Secure Fingerprinting on Sound Foundations

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr. Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

von
Ahmad-Reza Sadeghi

Gutachter:
Prof. Dr. Birgit Pfitzmann
Dr. Moti Yung

Dekan:
Prof. Dr. Philipp Slusallek

Einreichung:
24. März 2003

Promotions-Kolloquium:
2. Mai 2003

Saarbrücken, 2003

To my parents

Abstract

The rapid development and the advancement of digital technologies open a variety of opportunities to consumers and content providers for using and trading digital goods. In this context, particularly the Internet has gained a major ground as a worldwide platform for exchanging and distributing digital goods. Beside all its possibilities and advantages digital technology can be misused to breach copyright regulations: unauthorized use and illegal distribution of intellectual property cause authors and content providers considerable loss. Protection of intellectual property has therefore become one of the major challenges of our information society.

Fingerprinting is a key technology in copyright protection of intellectual property. Its goal is to deter people from copyright violation by allowing to provably identify the source of illegally copied and redistributed content. As one of its focuses, this thesis considers the design and construction of various fingerprinting schemes and presents the first explicit, secure and reasonably efficient construction for a fingerprinting scheme which fulfills advanced security requirements such as collusion-tolerance, asymmetry, anonymity and direct non-repudiation.

Crucial for the security of such cryptographic systems is a careful study of the underlying cryptographic assumptions. In case of the fingerprinting scheme presented here, these are mainly assumptions related to discrete logarithms. The study and analysis of these assumptions is a further focus of this thesis. Based on the first thorough classification of assumptions related to discrete logarithms, this thesis gives novel insights into the relations between these assumptions. In particular, depending on the underlying probability space we present new results on the reducibility between some of these assumptions as well as on their reduction efficiency.

Kurzzusammenfassung

Die Fortschritte im Bereich der Digitaltechnologien bieten Konsumenten, Urhebern und Anbietern große Potentiale für innovative Geschäftsmodelle zum Handel mit digitalen Gütern und zu deren Nutzung. Das Internet stellt hierbei eine interessante Möglichkeit zum Austausch und zur Verbreitung digitaler Güter dar. Neben vielen Vorteilen kann die Digitaltechnik jedoch auch missbräuchlich eingesetzt werden, wie beispielsweise zur Verletzung von Urheberrechten durch illegale Nutzung und Verbreitung von Inhalten, wodurch involvierten Parteien erhebliche Schäden entstehen können. Der Schutz des geistigen Eigentums hat sich deshalb zu einer der besonderen Herausforderungen unseres Digitalzeitalters entwickelt.

Fingerprinting ist eine Schlüsseltechnologie zum Urheberschutz. Sie hat das Ziel, vor illegaler Vervielfältigung und Verteilung digitaler Werke abzuschrecken, indem sie die Identifikation eines Betrügers und das Nachweisen seines Fehlverhaltens ermöglicht. Diese Dissertation liefert als eines ihrer Ergebnisse die erste explizite, sichere und effiziente Konstruktion, welche die Berücksichtigung besonders fortgeschrittener Sicherheitseigenschaften wie Kollusionstoleranz, Asymmetrie, Anonymität und direkte Unabstreitbarkeit erlaubt.

Entscheidend für die Sicherheit kryptographischer Systeme ist die präzise Analyse der ihnen zugrunde liegenden kryptographischen Annahmen. Den im Rahmen dieser Dissertation konstruierten Fingerprintingsystemen liegen hauptsächlich kryptographische Annahmen zugrunde, welche auf diskreten Logarithmen basieren. Die Untersuchung dieser Annahmen stellt einen weiteren Schwerpunkt dieser Dissertation dar. Basierend auf einer hier erstmals in der Literatur vorgenommenen Klassifikation dieser Annahmen werden neue und weitreichende Kenntnisse über deren Zusammenhänge gewonnen. Insbesondere werden, in Abhängigkeit von dem zugrunde liegenden Wahrscheinlichkeitsraum, neue Resultate hinsichtlich der Reduzierbarkeit dieser Annahmen und ihrer Reduktionseffizienz erzielt.

Zusammenfassung

Die rapiden Entwicklungen im Bereich der Informationsverarbeitung und -übertragung stellen neue Herausforderungen für digitale Medien dar, denen in angemessener Weise begegnet werden muss. Digitale Inhalte können praktisch kostenfrei ohne Verlust vervielfältigt und sehr schnell ausgetauscht bzw. verteilt werden. In diesem Zusammenhang spielt das Internet eine bedeutende Rolle, da es eine universelle Plattform mit globaler Reichweite für die Nutzung und Verbreitung digitaler Güter bietet. Darüber hinaus sind viele Anwendungen mittlerweile einfach bedienbar, so dass das Internet zu einem festen Bestandteil des alltäglichen Lebens geworden ist. Leider werden die technologischen Vorzüge der Digitaltechnik zunehmend zum Verstoß gegen das Urheberrecht missbraucht: Unautorisierte Nutzung und illegale Verbreitung des geistigen Eigentums verursachen den Urhebern erhebliche Schäden. Der Schutz des geistigen Eigentums hat sich daher zu einer besonderen Herausforderung unseres Digitalzeitalters entwickelt.

Es wurden bisher verschiedene technische und rechtliche Maßnahmen zum Urheberschutz vorgeschlagen, die häufig unter dem Begriff *Digital Rights Management* (DRM) zusammengefasst werden. Das Ziel von DRM-Systemen besteht allgemein darin, geeignete Umgebungen und Rahmenbedingungen für den Handel und die Nutzung digitaler Werke zu schaffen, so dass die Interessen und Sicherheitsanforderungen aller beteiligter Parteien in sinnvoller Weise berücksichtigt werden können.

Ein Schwerpunkt dieser Dissertation beschäftigt sich mit Fingerprintingverfahren, welche eine wichtige technische Maßnahme zum Urheberschutz in DRM-Systemen darstellen. Um die Rolle und Bedeutung der Fingerprintingverfahren innerhalb von DRM-Systemen einordnen zu können, werden im Folgenden einige zentrale Aspekte dieser Systeme kurz erläutert.

Die Komponenten von DRM-Systemen können in zwei funktionale Klassen unterteilt werden: Komponenten zur Handhabung von Rechten auf digitalen Werken (digitale Rechte) und Komponenten zur Handhabung von Missbräuchen.

Handhabung von Rechten: Die wesentliche Aufgabe dieser Klasse von Komponenten ist die Verwaltung von Urheber- und Nutzungsrechten für digitale Werke. In diesem Zusammenhang ist zu beachten, dass diese Rech-

te sich nicht nur auf Originalwerke, sondern auch auf dazu *ähnliche* Werke (*Werkklassen*) beziehen.[1] Auf diesen im technischen Urheberschutz häufig übersehenen Aspekt wurde zum ersten Mal in der Arbeit von Adelsbach, Pfitzmann, and Sadeghi (2000) hingewiesen, die u.a. verschiedene praktische Methoden zur automatischen Überprüfung der Ähnlichkeit zwischen Werken (Ähnlichkeitstest) vorschlägt. Ein weiterer wichtiger Aspekt besteht darin, dass Rechte selbst wiederum durch verschiedene Transaktionen wie Transfer und Lizenzierung weitergegeben werden können. Zu diesen Transaktionen gehören auch die Protokolle zum Urheberschaftsbeweis. Diese ermöglichen es den Urhebern, ihre rechtmäßige Urheberschaft auf digitale Werke einer anderen Partei direkt zu beweisen (siehe Adelsbach, Pfitzmann, and Sadeghi (2000)). Diese Aspekte stellen eine notwendige Grundlage für einen sinnvollen Handel mit digitalen Werken dar.

Handhabung von Missbrauch: Diese Klasse von Komponenten stellt die technischen Maßnahmen bereit, die zum Schutz der Urheberrechte gegen Missbrauch eingesetzt werden.[2] Diese können wiederum nach *präventiven* und *abschreckenden* Maßnahmen klassifiziert werden.

Präventive Maßnahmen haben das Ziel, den Missbrauch (z.B. unautorisierte Nutzung, Anfertigung von Kopien oder deren Verbreitung) zu verhindern, in dem Sinne, dass der Missbrauch mit heute zur Verfügung stehenden technischen Mitteln nicht möglich sein soll. Idealerweise benötigt man eine vertrauenswürdige Plattform, welche den Zugriff auf digitale Inhalte nach bestimmten Sicherheitsstrategien kontrolliert.[3] Die Realisierung einer solchen Plattform setzt manipulationssichere Hardware bzw. Software voraus und ist in der Praxis nur schwer realisierbar.[4] Abhilfe bieten die im Folgenden diskutierten abschreckenden Maßnahmen.

Abschreckende Maßnahmen sind komplementär zu präventiven Maßnahmen und sollen die Nutzer wegen zu befürchtender Konsequenzen vor illegalen Aktionen abhalten, indem sie die Identifikation eines Betrügers und das Nachweisen seines Fehlverhaltens ermöglichen. In diesem Kontext stellen *Fingerprintingverfahren* eine Schlüsseltechnologie dar. Ihr Ziel ist es, vor illegaler Vervielfältigung und Verteilung digitaler Werke abzuschrecken. Hierbei werden robuste *steganographische*[5] Verfahren (*Watermarking*) verwen-

[1] Ähnliche Werke entstehen durch nachvollziehbare Transformationen des Originalwerks, wie beispielsweise eine JPEG-Komprimierung eines Bildes.

[2] Dies ist keine zwingende Einschränkung, da sie auch nicht-technische Maßnahmen wie gesetzliche Richtlinien und zugrunde liegende Geschäftsmodelle beinhalten können.

[3] Eine der ersten Ideen in diesem Kontext war der Ansatz zur Superdistribution für Softwareprodukte. Hierbei soll die Software zwar frei verteilt werden, allerdings soll sie nur lauffähig sein, wenn die entsprechend notwendige Superdistributionskomponente, beispielsweise ein Dongle, installiert ist (Mori and Kawahara (1990), Cox (1996)).

[4] Siehe z.B. Anderson (2001) für Probleme, die Hardware betreffen und Barak et al. (2001) für das Problem der Softwarelösungen mittels sog. "code obfuscation".

[5] Mit Steganographie bezeichnet man die Methoden zur verdeckten Kommunikation,

det, um die Identifizierungsinformation des Käufers, den sog. *Fingerprint*, in das zugrunde liegende Werk einzubetten, bevor dieses an den Käufer weitergegeben wird. Die Anforderungen an das Watermarkingverfahren sind *Unwahrnehmbarkeit* und *Robustheit* des Wasserzeichens. Ersteres fordert, dass die eingebettete Information von dem Käufer nicht wahrgenommen werden soll und letzteres, dass das eingebettete Wasserzeichen nicht gelöscht werden kann, ohne zugleich das digitale Werk zu zerstören. Findet man nun eine illegale Kopie, die ähnlich zu einem der Originalwerke ist, so sollte ein Fingerprintingverfahren es ermöglichen, die Identifizierungsinformation zu extrahieren und somit die Quelle der illegalen Verteilung zu identifizieren.

Fingerprintingverfahren können unterschiedliche Sicherheitseigenschaften haben. Eine besonders wichtige und nicht trivial erreichbare Eigenschaft ist die Resistenz gegen sog. *Kollusionsangriffe*. Bei diesem Angriff schließt sich eine bestimmte Anzahl von Besitzern gekennzeichneter Kopien eines Werkes (eine *Kollusion*) zusammen und versucht, aus ihren Kopien eine neue Kopie zu erzeugen, aus der sich keine zur Käuferidentifikation verwertbare Information extrahieren lässt. Eine andere für die Praxis sehr bedeutende, aber ebenso nicht triviale Eigenschaft, ist die *Asymmetrie* von Fingerprintingverfahren. Das Ziel besteht hierbei in der Beweisbarkeit einer Identifizierung. Wenn der Verkäufer das gekennzeichnete Werk ebenfalls kennt, reicht die alleinige Vorlage der Identifizierungsinformation in der Realität nicht als Beweis aus, um einen neutralen Dritten (z.B. ein Gericht) von einer unerlaubten Verbreitung des gekennzeichneten Werkes zu überzeugen. Fingerprintingverfahren können um weitere wichtige und komplexe Sicherheitseigenschaften wie *Käuferanonymität* und *direkte Unabstreitbarkeit* erweitert werden. Käuferanonymität bedeutet, dass der Käufer digitale Werke anonym erwerben kann und auch weiterhin anonym bleibt, solange er die erworbenen Werke nicht widerrechtlich verteilt. Im Falle einer unerlaubten Weitergabe kann jedoch die Anonymität des betrügerischen Käufers aufgehoben werden. Direkte Unabstreitbarkeit erlaubt dem Händler, einem Dritten das Fehlverhalten eines betrügerischen Käufers direkt nachzuweisen, ohne dass hierzu die Beteiligung des betroffenen Käufers erforderlich ist. Diese Eigenschaft ist beispielsweise in einem Rechtsstreit von besonderer praktischer Relevanz. Die Realisierung der genannten Eigenschaften benötigt den Einsatz anspruchsvoller kryptographischer Techniken, welche im Rahmen dieser Dissertation detailliert vorgestellt werden. Des Weiteren werden verschiedene Fingerprintingsysteme und die von ihnen erfüllten Sicherheitsanforderungen systematisch und modular beschrieben. Insbesondere wird die erste explizite, modulare, effiziente und beweisbar sichere Konstruktion für ein Fingerprintingverfahren vorgestellt, welche alle oben genannten Sicherheitseigenschaften bietet.

Fingerprintingsysteme verwenden unterschiedliche kryptographische

wobei die zu kommunizierende Information in einem Trägermedium versteckt wird.

Protokolle und Primitive, deren Sicherheit auf kryptographischen Annahmen beruhen. Hierbei wird angenommen, dass bestimmte — meist zahlentheoretische — Probleme mit vertretbarem Aufwand, d.h. in polynomieller Zeit, nicht gelöst werden können. Ein in diesem Zusammenhang weithin bekanntes Problem beruht auf der Schwierigkeit, diskrete Logarithmen (DL) in bestimmten algebraischen Strukturen berechnen zu können (McCurley 1990). Die Sicherheit vieler kryptographischer Systeme beruht auf solchen DL-basierten Annahmen (Maurer 2001). Neben der reinen DL-Annahme existieren zahlreiche verwandte Annahmen wie beispielsweise die Diffie-Hellman (DH) Annahme und die Decisional Diffie-Hellman (DDH) Annahme (Diffie and Hellman 1976; Brands 1994; Boneh 1998). Bei der konkreten Formulierung dieser Annahmen hat man jedoch gewisse Freiheitsgrade, die durch verschiedene Aspekte wie das Berechnungsmodell, den Problemtyp (Computational, Decisional oder Matching), den zugrunde liegenden Wahrscheinlichkeitsraum sowie die tolerierbare Erfolgswahrscheinlichkeit des Angreifers gegeben sind. Allerdings sind einige dieser Aspekte und ihre Auswirkungen auf die Sicherheitsgarantien in der Literatur nicht ausreichend beachtet. Eine tiefgehende Analyse DL-basierter Annahmen bildet den anderen Schwerpunkt dieser Dissertation. Hierbei werden die kryptographisch relevanten Parameter identifiziert und zum ersten Mal ein mathematisches Fundament zur präzisen Definition DL-basierter Annahmen aufgebaut. Die Annahmen werden, basierend auf diesem neuen Fundament, klassifiziert und analysiert. Insbesondere werden sie in Abhängigkeit eines neu eingeführten Parameters untersucht, der sog. Granularität, welche den einer Annahme zugrunde liegenden Wahrscheinlichkeitsraum bestimmt. Dies stellt eine vollständig neuartige Analyse dar, welche im Rahmen dieser Dissertation gemacht wurde und neue Resultate und hilfreiche Erkenntnisse über die Zusammenhänge zwischen diesen Annahmen liefert.

Überblick

In Kapitel 1 wird zunächst eine kurze Einführung in die Schwerpunkte und Resultate dieser Dissertation gegeben. Insbesondere werden Einblicke in die wichtigsten Aspekte des technischen Urheberschutzes für digitale Werke gegeben, um schließlich die Rolle der Fingerprintingverfahren in diesem Kontext einzuordnen.

In Kapitel 2 werden die kryptographischen Annahmen basierend auf diskreten Logarithmen (DL) betrachtet. Dabei werden zunächst die zur präzisen Definition kryptographischer Annahmen notwendigen mathematischen Aspekte untersucht. In diesem Zusammenhang werden die hierfür wichtigsten Parameter identifiziert und eingehend beschrieben. Darauf aufbauend werden die DL-basierten Annahmen klassifiziert und ihre wichtigsten Eigenschaften und Varianten unter besonderer Berücksichtigung der identifizierten Parameter diskutiert. Insbesondere wird der Einfluss der Granularität des

Wahrscheinlichkeitsraums untersucht. Für einige DL-Annahmen wird bewiesen, dass diese für niedrige Granularität zueinander äquivalent sind, während dies im generischen Sinne für ihre hochgranularen Varianten nicht gilt. Hierbei werden Reduktionen zwischen der Diffie-Hellman (DH) Annahme, der Square Exponent (SE) Annahme und der in dieser Arbeit neu vorgestellten Inverse Exponent (IE) Annahme für beide Problemtypen, Computational und Decisional, betrachtet. Erste Ergebnisse über DL-basierte Annahmen wurden bereits in der Arbeit Sadeghi and Steiner (2001) veröffentlicht. Allerdings werden diese Resultate in der vorliegenden Dissertation auf allgemeinere Annahmen, d.h. für Gruppen mit nicht primer Ordnung, ausgeweitet und bewiesen.

In Kapitel 3 werden weitere Notationen eingeführt, die bei den vorgestellten Fingerprintingverfahren verwendet werden. Darüber hinaus gibt dieses Kapitel einen detaillierten Überblick über die notwendigen kryptographischen Primitive und Unterprotokolle wie Commitment-, Verschlüsselungs- und Signaturverfahren sowie interaktive Beweissysteme und ihre Zero-Knowledge Eigenschaft.

In den Kapiteln 4 und 5 werden ein modulares Schema zur Konstruktion verschiedener Typen von Fingerprintingverfahren (symmetrisch, asymmetrisch, anonym) vorgestellt und die Sicherheitsanforderungen an diese Verfahren formuliert. Anschließend wird die erste explizite Konstruktion für ein anonymes asymmetrisches kollusionstolerantes Fingerprintingverfahren entwickelt und dessen Sicherheitsbeweis angegeben. Hierbei befindet sich in Kapitel 5 die detaillierte Konstruktion des hierzu entwickelten Kernmoduls, das Einbettungsprotokoll, das zur sicheren und verifizierbaren Einbettung der Identifizierungsinformation eingesetzt wird.

Die wesentlichen Ergebnisse bezüglich der neuen Fingerprintingprotokolle wurden bereits in den Arbeiten Pfitzmann and Sadeghi (1999) und Pfitzmann and Sadeghi (2000) veröffentlicht. In dieser Dissertation sind jedoch eine Reihe weiterer neuer Details hinzugekommen, wie beispielsweise zum Beweis der Anonymitätseigenschaft, welcher Simulationstechniken verwendet. Hierbei erfordern die Komplexität der Protokolle, die Interaktionen zwischen den Unterprotokollen sowie die Verwendung informationstheoretisch versteckender Commitments den Einsatz neuer Techniken zur Erreichung korrekter Simulation.

Resultate

Die wichtigsten Resultate der vorliegenden Dissertation lassen sich folgendermaßen zusammenfassen:

- Die Dissertation beinhaltet neue Einblicke in die Zusammenhänge zwischen den kryptographischen Annahmen, die auf diskreten Logarithmen (DL) beruhen. Insbesondere werden erstmalig die Zusammenhänge zwischen diesen Annahmen in Abhängigkeit eines neu eingeführten Parameters, der Granularität, untersucht, die den einer Annahme zugrunde liegenden Wahrscheinlichkeitsraum bestimmt. Es wird bewiesen, dass einige DL-basierte Annahmen für niedrige Granularität äquivalent sind, während dies im generischen Sinne für ihre hochgranularen Varianten nicht gilt. Weiterhin wird gezeigt, dass Reduktionen für niedrigere Granularität effizienter sind als für höhere Granularität. Dadurch wird die besondere Bedeutung und Berücksichtigung der Granularität bei der Definition der Annahmen unterstrichen.

  Die zugehörige Analyse basiert auf einer Klassifikation DL-basierter Annahmen, welche die präzise Formulierung dieser Annahmen erlaubt und über die bekannten Ansätze deutlich hinaus geht.

- Eine neue DL-basierte Inverse Exponent (IE) Annahme wird eingeführt und deren Zusammenhang zu anderen bekannten DL-basierten Annahmen, wie beispielsweise Diffie-Hellman (DH), durch konstruktive Reduktionsbeweise dargelegt. Dabei liefern die Beweise auch die konkrete Sicherheit, d.h. den Reduktionsaufwand. Die IE Annahme stellt die Grundlage für die Sicherheitsanalyse des in dieser Dissertation eingeführten anonymen Fingerprintingverfahrens dar.

- Die erste explizite, effiziente und beweisbar sichere Konstruktion eines Fingerprintingverfahrens wird vorgestellt, welche die besonderen Sicherheitseigenschaften wie Asymmetrie, Anonymität und direkte Unabstreitbarkeit bietet. Insbesondere wird eine detaillierte und explizite Konstruktion für das Kernmodul eines (anonymen) asymmetrischen Fingerprintingverfahrens, das Einbettungsprotokoll, entwickelt und als Zero-Knowledge und kollusionstolerant bewiesen.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction and Overview

This chapter gives an outline of the content of this thesis. In particular, it provides a summary of the major results: It presents the first concrete and reasonably efficient construction for a fingerprinting scheme fulfilling the most advanced and desirable security requirements. Further, it presents the first thorough classification and study of the underlying cryptographic assumptions that are related to discrete logarithm, and gives new insights into the relations among these assumptions.

FINGERPRINTING is a key technology supporting the copyright protection of intellectual property. The main goal of fingerprinting is to deter people from illegal redistribution of intellectual property by enabling the rightful copyright owner to trace the source of illegal redistribution. As one of its focuses, this thesis considers the design and construction of different types of fingerprinting schemes, in particular explicit construction of secure anonymous fingerprinting schemes.

Fingerprinting schemes employ various cryptographic primitives and protocols whose security relies on cryptographic assumptions that are usually based on number theoretic problems. For the fingerprinting schemes presented here, we are mainly concerned with assumptions based on solving discrete logarithms (DL) in certain algebraic structures. The precise definition of DL-related assumptions, and the study and analysis of their relations is another focus of this thesis.

Before going into the details of each of these topics, we will first give an overview of the main aspects and technical measures of copyright protection systems to get the grand picture, and to emphasis the role of the fingerprinting within such systems. Then we give an overview of the main aspects of the DL-based assumptions.

1.1 Copyright Protection

Digital technology and media offer content providers and producers many business opportunities and consumers many advantages for using, exchanging and trading digital goods (e.g., audio, video, etc.). In this context, the Internet has taken an important place as a global platform for using and distributing digital goods, and has become a fixed part of the daily life in the information society. However, all these technological possibilities face us also with challenging problems regarding copyright protection of intellectual property. **Digital Rights Management (DRM)** systems should provide the appropriate environment for trading digital works while protecting the rights and interests of *all* involved parties in the sense of multi-party security.

One may divide a DRM system into two main components, namely, **Rights Handling** and **Misuse Handling**.

1.1.1 Rights Handling

This component is responsible for handling the **copyrights** on digital works. Informally, a copyright, as a form of protection by law, provides an author or her legal assignees with exclusive rights to control certain uses (e.g., reproduction, performance, redistribution) of intellectual and expressive works, and to authorize others to do the same. Intellectual and expressive works include literary, dramatic, musical, and artistic works (see, e.g., US Copyright Office (2002)). Some important aspects in this context are:

- *Similarity:* Rights cannot only refer to original works, but also to a class of **similar** works which we call **work-class**. Similar works are those works derived from the author's original works by means of trivial (understandable) transformations, e.g., a JPEG compressed version of an image. Note that digital works can be treated on the *bit-level*, and on the *perceptual level*. On the bit-level, each digital work is represented and identified by its unique bit-string encoding, and works are considered equal, if and only if they have the same bit-representation. On the perceptual level, the notion of a work is more "fuzzy", since human perception of (digital) works is quite robust against modifications of works. Here, "fuzzy" means that "one work" on the perceptual level concerns *all works which are sufficiently similar with respect to the human perception* (e.g., rescaled, cropped, rotated or compressed versions of an image), and each having different bit-representations (e.g., PNG-, TIFF- or JPEG-encodings of the same image). All such similar works are included in a *work-class*. This issue was first noticed by Adelsbach, Pfitzmann, and Sadeghi (2000). They propose methods to efficiently represent work-classes, e.g., by means of robust **digital watermarks** or **robust hashes**. Note that the perceptual level is the more natural way of treating digital works. Therefore, the rights handling component should be capable of handling rights on digital works as understood on the perceptual level.

- *Transactions on digital rights*: Rights on work-classes can be treated as digital goods, and thus, they can be subject to various transactions such as **licensing** and **transfer**. For instance, consider a content provider who may sell the "right to access" (licenses to read or to listen) to certain objects (special online-news, music) in its database instead of selling the whole database to consumers. The provider may transfer certain rights or even all rights on its database to a costumer. Other types of transactions are **proof of authorship** on work-classes, and **proof of ownership** on rights. These transactions are essential for reasonable trading, since honest users would (and should) not purchase rights without ensuring that the selling party is either the rightful author or an authorized assignee (see Adelsbach, Pfitzmann, and Sadeghi (2000) and Adelsbach and Sadeghi (2002)).[1] Note that after purchasing digital rights the buyer does not become the legal owner of these rights, if this purchase *violates* the copyright conditions — even if the buyer has purchased that work in good faith!

  The rights handling component of a DRM system should enable authorized parties to perform the mentioned transactions on digital rights at any time.

1.1.2 Misuse Handling

This component is responsible for providing technical and non-technical measures to protect copyright owners against copyright violations. The non-technical aspects concern the legal terms, i.e., what the law permits, and the underlying business models (see also National Research Council (2000)). Our concern, however, is the technical copyright protection against misuse, i.e., against unauthorized copying, manipulating, redistributing, etc. A variety of approaches have been proposed, each having its advantages and

[1] One may think that prove of authorship is obsolete since there are a few global players in "Hollywood", and everyone knows their products (e.g., "Star wars Episode I"), and can easily recognize the copyright holders. However, some remarks are in place: On the one hand, even in this case, there may be business models involving small unknown resellers who are not immediately trusted a priori by potential customers, and therefore, have to prove that they are authorized to sell licenses. On the other hand, DRM systems should not be restricted to the distribution of the products of large music and motion picture industries. They should also support the distribution of "non-famous" works created by "non-famous" freelance artists, such as photographers, graphic designers or music bands. Unfortunately, most of today's DRM proposals do not support this, since they commonly were initiated by large content providers. Thus, from usual DRM proposals, it is obvious, that the interests of powerful authors and content providers dominated those of small ones.

limitations. The proposed technical measures may be classified in two main classes, namely, **preventive measures** and **deterrent measures**.

1.1.2.1 Preventive measures

The goal of these methods is to *prevent* misuse. Ideally, one requires a "trusted media player", also called "trusted viewer", enabling only the authorized users to "view" (watch, listen to, execute, etc.) digital works while controlling the use of works according to the specified rights policies.

In this context, one of the earliest approaches for efficient distribution of software was **super distribution**. The idea is to make software freely available without any restriction, but, the software would run on a system only if this system have installed the "super distribution technology" (see Mori and Kawahara (1990) and Cox (1996)).

The realization of such systems requires the combination and interplay of various components to achieve at least some of the desired properties of the ideal "trusted media player". In the following, we will shortly consider the main aspects and components of such a system:

One may consider the **access control** to information as the most fundamental form of technology for copyright protection of digital property; only the authorized parties are allowed to access the information or perform certain actions. In this context, one may classify **encryption** as a basic tool implementing a certain type of access control. Encryption schemes should protect digital content when they are stored or transmitted, and only those parties possessing the decryption key(s) are capable of accessing the content. A well-known application is pay-per-view television. Users who pay are given keys to be able to decrypt the encrypted content sent by the station. However, one cannot "prevent misuse", once the content is decrypted[2]: authorized parties may cheat, and reveal the decryption key or the decrypted content to unauthorized parties. Another basic form of access control is employed by today's operating systems, and can offer limited security.

The considerable advancements in the area of access control and the authorization languages (see e.g., Jajodia, Samarati, and Subrahmanian (1997) and Bonatti, de Capitani di Vimercati, and Samarati (2002)) may be exploited to formalize and express rights statements for digital properties. Rights expressions may get very complex and may include a variety of aspects such as permissions, constraints, obligations, rights owners, etc. For

[2]This is even the case if the decryption is just in time and on site, meaning that the content is not decrypted until just before it is used, no temporary copies are ever stored, and the decryption is physically very close to the usage site. However, skilled adversaries might still be able to manipulate I/O routines of the computer, and intercept the decrypted content just before it is used. Note that the I/O routines of the existing operating systems were not originally designed to hide the information they are processing (National Research Council 2000).

instance, consider Aisha and Bill who made a certain video (e.g., how to find oil in the dessert). George is only allowed (usage permission) to watch certain parts of the film for a certain period of time (constraint), and he has to pay usage fee (obligation to pay) to Aisha and Bill (rights owner).

However, existing access control methods offer partially what is really required for handling rights on digital properties. Note that the access and use conditions attached to digital properties may get very complex. Further, access control systems for digital properties must also be able to securely handle the access policies for long time periods, and deal with administrative tasks such as archiving a huge amount of different data and document types.

Recent solutions concern the design of **Rights Management Languages**. This, relatively new and promising, approach attempts to formulate rights and complex access and usage statements and policies (attached to digital properties) in a formal and machine-readable language enabling computers to decide whether the inquiries are permitted (see e.g., Gunter, Weeks, and Wright (2001)).

However, as for encryption, we are faced with the question of what should be done, once the digital content is in possession of a user. Dishonest users may let unauthorized users access the content, or they may be able to circumvent or manipulate rights policies. To protect the system in such an adversarial environment, solutions with **tamper-resistant** hardware and software are indispensable.

However, past hardware solutions based on tamper-resistant hardware appear to be costly, and ineffective for general-purpose computers.[3]

To avoid problems with special-purpose hardware, it was desired to design end-to-end systems based on software to control digital content. Such systems apply a variety of measures such as **secure digital containers**, e.g., in IBM's **cryptolope** system (Group 1997). Digital containers are cryptographically protected files which contain the content, access policies expressed in the rights management language, fingerprinting information (see below), etc. A customer purchasing a work in an open environment receives a cryptographic container containing the corresponding work, a contract on the purchased rights and other necessary information. Any usage must be confirmed by an observer software that checks the rights attached to

[3]For instance, a way for distributing software products is to use *dongles*, hardware devices plugged, e.g., into the printer port. The software does not work unless the dongle is installed. The most common type performs a challenge and response protocol between the dongle and the software. However, dongles turned out to be impractical for mass software market. Consumers did not accept them since for each application a separate dongle was needed. Moreover, applying similar preventive measures for mass market requires agreements among manufacturers of different products, e.g., to prevent producing CD or MP3 copies, CD burner devices or MP3 readers must have build-in features such as "copying not allowed" or "do not play a copy". However, such agreements are hard to achieve in practice as well as consumer acceptance to use such devices (see also Anderson (2001)).

the underlying work. However, implementing secure containers in software is a difficult task, if not impossible.[4]

A recent approach which can be applied in this context is **Trusted Computing Platforms** proposed by **Trusted Computing Group** (TCG), the former Trusted Computing Platform Alliance (TCPA) (2002), or by Microsoft's **Next-Generation Secure Computing Base** (NGSCB) (England et al. 2003), the former **Palladium**. In the TCG Specification[5] the tamper-resistance hardware is reasonably reduced to a small module called Trusted Platform Module (TPM). This technology offers functionalities that can be used to build much more robust DRM systems. However, without the appropriate operating system the TPM approach alone cannot prevent end-users from circumventing the protection measures (see, e.g., Sadeghi and Stüble (2003)).

1.1.2.2 Deterrent measures

As mentioned in Section 1.1.2.1, the realization of preventive measures in practice is reasonable only in conjunction with *tamper-resistance*. Tamper-resistance, however, is a strong assumption, and depending on the application, it may easily be broken by skilled adversaries (see also Anderson and Kuhn (1996)). Hence, we are back to the original question of what to do against misuse when the digital content is under the control of the adversary. Here, the only technical means of hindering misuse are *deterrence* measures. These are complementary to preventive measures with the main goal to *discourage* users from copyright violations. This means that users may copy or redistribute works illegally, however, copyright holders are provided with means to detect infringements after the fact (and then proceed with legal measures against the cheaters.)

A key deterrence technology is **fingerprinting**. Informally, a **fingerprint** is a characteristic of an object which distinguishes this object from other similar objects. Fingerprinting refers to the mechanism of combining a fingerprint with an object, or to identify fingerprints that are already intrinsic to an object. Fingerprints can be used in a variety of applications. Our concern in this thesis is fingerprinting digital content. Typical scenario for applying fingerprinting is as follows: A merchant, as the party who holds rights on digital works, secretly hides (embeds) identifying information (fin-

[4]Tamper-resistant software should resist manipulation or hide operations or secrets used by software. One of the methods to realize tamper-resistant software is **obfuscation**. Informally, an **obfuscator** is something like a (efficient) "compiler" that transforms a program to a new program which has the same functionality as the original program but "hard" to reverse-engineer (see, e.g., Goto et al. (2001), Hada (2000)). However, the effectiveness of the proposed methods are proven experimentally, and as Barak et al. (2001) show in their theoretical investigation of obfuscation, even under very weak formalizations of the above intuition, obfuscation is impossible.

[5]At the time of this writing there was no NGSCB specification available.

gerprint) in the work to be sold. Note that each copy of the original work obtains its unique fingerprint for the corresponding buyer. For the purpose of embedding one applies **steganographic** techniques, in particular **watermarking**, which are required to embed the fingerprint (watermark) imperceptibly and robustly. **Robustness** means that the watermark cannot be removed without rendering the underlying work useless. Later, if the merchant finds a redistributed work which is *similar* to one of her original works, he should be able to extract the (embedded) fingerprint, and identify the user(s) who illegally redistributed this work.

An example of a typical application is fingerprinting of digital audio, images and video to monitor piracy of digital content, and to trace the source of illegal redistributions. For instance, Pay-TV stations can use fingerprints to trace subscribers who illegally redistribute the content they (legally) receive.

There are several useful and advanced security properties one desires the Fingerprinting schemes to provide. The first non-trivial enhancement is resisting **collusion-attacks**. In a collusion attack, the adversary has access to a certain number of fingerprinted copies of the same work, and attempts to generate a copy which contains no usable information that leads to a successful identification. Another very important and non-trivial property is **asymmetry** of a fingerprinting scheme. Its goal is to provide provable identification: If the merchant knows the fingerprinted work, as it is the case for ordinary fingerprinting schemes, the extraction of identifying information is not sufficient as a proof of treachery (illegal redistribution) to an honest third party (e.g., an arbiter in a trial).

Further desirable enhancements of fingeprinting schemes are **anonymity** for buyers (users) and **direct non-repudiation**. Anonymity property allows a buyer to purchase works anonymously and remain anonymous as long as she does not illegally redistribute a work, otherwise her anonymity can be revoked. Direct non-repudiation enables the merchant to obtain enough evidence in case of treachery to convince any arbiter, and the accused buyer does not need to participate in any trial protocol to deny charges. This property makes 2-party trials possible instead of common 3-party solutions, and is very useful in practice. To implement the above properties, one needs to apply sophisticated cryptographic techniques.

In this thesis, we will consider different types of fingerprinting schemes and present the first concrete construction which offers the advanced security properties mentioned above.

1.2 Assumptions based on Discrete Logarithm

Fingerprinting schemes apply a variety of cryptographic primitives and techniques to achieve certain security properties. The security of these schemes relies, among others, on cryptographic assumptions on the computational difficulty of some particular number-theoretic problems. A well-known class of assumptions is related to the difficulty of computing **Discrete Logarithm (DL)** in cyclic groups (McCurley 1990; Maurer 2001). Among the various types of problems related to discrete logarithms, the best known ones, besides discrete logarithm, are the computational and decisional **Diffie-Hellman (DH)** assumptions (Diffie and Hellman 1976; Brands 1994; Boneh 1998).

In earlier days, DL-related assumptions were defined informally and imprecise. With the development of the modern theoretical cryptography, cryptographers started to formulate more precise definitions for these assumptions to be able to prove the security of cryptographic systems formally.

In the concrete formalizations of these assumptions, one has various degrees of freedom offered by parameters such as the underlying **computational model**, **problem type** (computational, decisional or matching), **probability space** or the tolerated **success probability of the adversary**. However, such aspects are often not precisely considered in the literature, and their impact on security guarantees are simply overlooked.

In this thesis, we address these aspects by identifying the parameters relevant to cryptographic assumptions. More precisely, we introduce the mathematical foundation required to precisely formulate and classify the assumption related to discrete logarithm. Based on this classification, we give new insights into the relations between some of these assumptions. In particular, we give the first analysis of the impact of the underlying probability space on the usefulness of these assumptions, on the existence of relations among them and on the efficiency of reductions between them.

1.3 Outline

Chapter 2 explores the cryptographic assumptions based on discrete logarithm, their mathematical foundation and their relations. More precisely, we identify parameters relevant to concrete mathematical formulation of cryptographic assumptions, and systematically classify the DL-related assumptions. The classification allows us to formulate general assumption statements in a compact form and to give novel insights into the relations between these assumptions. In particular, we investigate the impact of a newly introduced parameter called *granularity*. It describes the underlying probability space of an assumption. Depending on granularity, we prove that the medium-granular variants of some assumptions are equivalent whereas their high-granular variants are not equivalent in generic sense. Further, it is shown that reductions among lower granular assumptions are more efficient than among their higher granular counterparts. These relations are proven between a newly introduced assumption, called Inverse Exponent (IE), and known assumptions such as Diffie-Hellman (DH) for both computational and decisional problem types. IE plays an important role in the security analysis of the anonymous fingerprinting scheme proposed in Chapter 4.

Preliminary results on the DL-assumptions have been published in Sadeghi and Steiner (2001). In this thesis, these results are generalized to larger classes of assumptions (general group orders and not only for groups of prime order) and rigorously proven where the proofs are more involved.

In Chapter 3, we introduce further conventions and mathematical notations required in the context of fingerprinting schemes. Moreover, the chapter motivates and reviews in detail the main cryptographic primitives and protocols such as commitment schemes, (blind) signatures schemes, interactive proof systems and the zero-knowledge property.

In Chapters 4 and 5, we first give systematic modular constructions for different fingerprinting schemes (symmetric, asymmetric, anonymous). In particular, we give a framework for the embedding procedure which is the core part of asymmetric fingerprinting schemes. Finally, we introduce the first explicit construction for a reasonably efficient and secure fingerprinting scheme which fulfills the advanced security requirements collusion-tolerance, asymmetry, anonymity and direct non-repudiation. Hereby, Chapter 5 presents in detail the construction of the first explicit embedding protocol, which is the core module of the anonymous asymmetric fingerprinting scheme.

The main results of the fingerprinting part have been appeared in Pfitzmann and Sadeghi (1999) and Pfitzmann and Sadeghi (2000). However, new details have been added, in particular for the proof of anonymity (unlinkability) property which is the core security proof for the proposed anonymous fingerprinting scheme.

1.4 Summary of Major Results

The major results of this thesis are as follows:

- This thesis presents novel insights into the relations between cryptographic assumptions related to discrete logarithms (DL). In particular, it investigates for the first time the relation between these assumptions dependent on a newly identified parameter, "granularity", which describes the underlying probability space of an assumption.

  The strong impact of granularity is shown by proving surprising separability results: some DL-related assumptions are equivalent for their medium-granular variant whereas their high-granular variants are provably not reducible with respect to generic algorithms. Further, it is shown that reductions for medium granularity can achieve much better concrete security (reduction efficiency) than the counterpart high-granular reductions.

The analyses leading to these results are based on the first thorough classification of these assumptions introduced in Sadeghi and Steiner (2001). The classification allows compact and precise formulation of assumption statements.

- A new DL-related assumption, Inverse Exponent (IE), is introduced and its relation is shown to other known assumptions such as Diffie-Hellman (DH). The corresponding reduction proofs are constructive and give the concrete security (reduction cost). The IE assumption is useful in the context of the security analysis of the anonymous fingerprinting scheme introduced in this thesis.
- The first explicit and reasonably efficient construction for an anonymous fingerprinting scheme is introduced and proven secure. The construction is modular and offers, beside anonymity, also other advanced security properties such as collusion-tolerance, asymmetry and direct non-repudiation. In particular, the first concrete construction for the core module of an asymmetric fingerprinting scheme, namely the embedding protocol, is given. This protocol is applied to securely and verifiably embed the identifying information into the underlying digital content.

Acknowledgements

There is a long list of people I would like to thank, beginning with my friends from the college time!

My first thanks go to those co-authors of the papers on which this thesis is built: Birgit Pfitzmann, Michael Steiner and André Adelsbach. I also would like to thank the co-authors of other publications: André Adelsbach, Ammar Alkassar, Alexander Geraldy, Stefan Katzenbeisser, Matthias Schunter, Sandra Steinbrecher, Markus Schneider and Christian Stüble. It was a great pleasure and fun to work with you and of course, I hope we will continue this in the future.

Special thanks go to Birgit Pfitzmann. She offered me a research position and provided me with the opportunity to pursue a Ph.D. Birgit is a knowledgeable and friendly person, always open for questions and discussions. During the time I was doing this work I have learned much from her not only about cryptography, security and computer science in general but also about plants and birds – Birgit has a considerable knowledge in these areas as well! I also enjoyed the joint work on teaching and supervising master theses.

I am indebted to Moti Yung for being one of the referees of this thesis. Although he is a very busy man he was willing to read my tiny (300 pages) Ph.D. and to do all the time consuming work a referee has to do. I am very grateful for his willingness to do all this, and I promise to bake brownies next time we meet! Moti's extensive research work, particularly in the area of cryptography, is well-known to everyone in the community and has inspired me for my research work. Further, Moti is a kind and humorous guy, and it was real fun to have him over here.

I am deeply grateful that Reinhard Wilhelm and Nicola Wolpert took time to be in my defence steering committee.

Other special thanks go to Stefan Brands for very fruitful discussions and fun time specially during the time at Financial Cryptography. Stefan's interesting work and ideas on cryptographic protocols, particularly in the context of electronic payment systems, were a very useful basis for some parts of my research work. I hope, he would manage to come when I make brownies!

Next, I would like to thank a very special friend Michael Steiner, the "efficiency man". He is the kindest smart a.. I have ever met. I enjoyed doing research with him a lot during the time we worked at Saarland University. In particular, I enjoyed the passionate discussions with him not only on a variety of research topics but also on philosophical, social and particularly political[6] ones. Specially, we often discussed socially valuable subjects such as whether Ph.D. and natural "fun stuff" can ever be compatible. Michael's urge to comment everything and my urge to argue was sometimes nerve-killing but also very funny, inspiring and fruitful (lots of fruits). I hope, we could catch this up in the near future in good "old Europe".

Special credits go to Michael Waidner who was always supportive and open for discussions and solving problems of different nature. He is a good Ph.D.-therapist. He motivated me (and also other colleagues) during the Ph.D. "process", which can get very painful and frustrating. Particularly, I appreciate his comments on part of this thesis.

I thank my friend and colleague Matthias Schunter. His coolness and talent for organizing various things contributed heavily to the relaxing working environment. It was a very nice time to work and to do research with him, and in particular, to jointly interview candidates to be hired at our institute. One of our best choices was Petra Maschke to whom I am very thankful for helping us (i.e., me) through all nasty and complicated administrative stuff. I do not want to exaggerate, but we were a fantastic team.

My thanks go also to my good friend Markus Schneider who was the first one explaining correlation attacks to me[7], and who advised me not to waste my time with other research topics than cryptography. His motivating and smart "comments" were always inspiring. I appreciate his effort for organizing those nice weekend excursions to France with our mutual friends,

[6]I am confident that one day he will admit that I was right, and that the world is governed by a collusion of evil people.

[7]Although it has nothing to do with this thesis, I just wanted to mention it.

particularly Mr. Fuckard. The excursions were relaxing, because of those romantic French villages with extremely romantic cows! Markus taught me a lot about French wine, i.e., he bought it for me. It is fun to do research with him, at least from my perspective. I would like to thank him and André Adelsbach for commenting the German summary of this thesis, whereby Markus's comments were of high poetic (Goethe-like) quality.

What could I have done without my friend Jürgen Stemler! Without his help I could have not been able to find a new living place in Saarbrücken within "one" day. Jürgen introduced the new environment to me and made my life much easier during my stay in Saarland.

Next, I would like to thank André Adelsbach, Ammar Alkassar, Sandra Steinbrecher and Christian Stüble for being very nice colleagues, and for their close collaboration and many inspiring discussions. Oliver Altmeyer for his intensive assistance in teaching and for nagging about it, Tom Beiler for many arguments about everything particularly on object oriented philosophy, Jürgen Brauckmann for being a nice and patient colleague, Ingrid Biehl for fruitful discussions on fingerprinting schemes, Tanja Brüssow for the nice time after the defence, Alexander Geraldy for hard working on the ISDN-Encryptor, Stefan Groß and Thomas Groß for a good and pleasant collaboration, Eike Kiltz for some comments on a paper relevant for this thesis, Heike Neumann for many useful discussions on number theoretical issues, my good friend Jan Holger (Max) Schmidt for helpful discussions on electronic payment systems, Igor Shparlinski for the hint about his result on our Inverse Exponent Problem, and lastly Chen Fen for temporarily destroying my office, my nerves and for singing German folk songs while I was writing papers.

I would like to deeply thank Judith Forner for her support and strong friendship over many years and for being patient toward my bad attitudes and accompanying me through many crucial situations and decisions in my life. I hope that this friendship never ends.

I want to thank my family for their never-ending support, and specially my dear sister Shahnaz. Without her help and effort I would have never been able to begin with this work, and I hope that I would be able to do the same for her. Words cannot express my thankfulness. It was a hard time for her and me, but, we managed it.

And at last thanks to those friends who sacrifice their youth and lives for the sake of freedom, taking on the sorrow to make the road free for others. I wish you were here.

Chapter 2

Exploring DL-based Assumptions

The security of many cryptographic constructions relies on assumptions related to Discrete Logarithms (DL). In the concrete formalizations of these assumptions some degrees of freedom are offered by parameters such as computational model, problem type (computational, decisional) or success probability of the adversary. In this chapter we first present the mathematical foundations required to precisely formalize and classify assumptions related to discrete logarithm. Based on this framework we briefly review existing results on the relation between Diffie-Hellman (DH) and Square Exponent (SE) assumptions. We then introduce a new assumption, called Inverse Exponent (IE) assumption, and extend the existing results by proving similar relations between IE, DH and SE.

In particular, we investigate the effect of a newly introduced parameter "granularity", which describes the underlying probability space in an assumption. Varying this parameter can strongly influence the relations between different assumptions and leads to surprising results: We prove that two DL-related assumptions can be reduced to each other for the medium granularity whereas they are provably not reducible with respect to generic algorithms for high granularity. Further, we show that reductions for medium granularity can achieve much better concrete security than equivalent high-granularity reductions.

MOST modern cryptographic systems rely on assumptions on the computational difficulty of some particular number-theoretic problem.[1]

[1] The exceptions are information-theoretically secure systems and systems such as hash-functions or shared-key encryption relying on heuristic assumptions, e.g., the Random Oracle Model (Bellare and Rogaway 1993).

One well-known class of assumptions is related to the difficulty of computing discrete logarithms in cyclic groups (McCurley 1990). In this class a number of variants exists. The most prominent ones, besides **Discrete Logarithm (DL)**, are the computational and decisional **Diffie-Hellman (DH)** assumptions (Diffie and Hellman 1976; Brands 1994). Less known assumptions are **Matching Diffie-Hellman** (Frankel, Tsiounis, and Yung 1996), **Square Exponent (SE)** (Maurer and Wolf 1996), and **Inverse Exponent (IE)** (Pfitzmann and Sadeghi 2000), an assumption closely related to the **Inverted-Additive Exponent (IAE)** Problem introduced by MacKenzie (2001)[2] and also implicitly required for the security of the schemes proposed by Camenisch, Maurer, and Stadler (1996) and Davida et al. (1997). Further related assumptions mentioned in the sequel are **Generalized Diffie-Hellman (GDH)** (Shmuely 1985; Steiner, Tsudik, and Waidner 1996) and the **Representation Problem (RP)** (Brands 1994). Several additional papers have studied relations among these assumptions, e.g., (Shoup 1997; Maurer and Wolf 1998a; Maurer and Wolf 1998b; Biham, Boneh, and Reingold 1999; Wolf 1999).

In the concrete formalizations of these assumptions, one has various degrees of freedom offered by parameters such as computational model, problem type (computational, decisional or matching) or success probability of the adversary. However, such aspects are often not precisely considered in the literature and consequences are simply overlooked. In this chapter, we address these aspects by identifying the parameters relevant to cryptographic assumptions. Based on this, we present a formal framework and a concise notation for defining DL-related assumptions. This enables us to precisely and systematically classify these assumptions.

Among the specified parameters, we focus on a parameter we call *granularity* of the probability space which underlies an assumption. Granularity defines what part of the underlying algebraic structure (i.e., algebraic group and generator) is part of the probability space and what is fixed in advance: For high granularity, an assumption has to hold for all groups and generators; for medium granularity, the choice of the generator is included in the probability space, and for low granularity, the probability is taken over both, the choice of the group and the generator. Assumptions with lower granularity are weaker than those with higher granularity. Nonetheless, not all cryptographic settings can rely on the weaker variants: Only when the choice of the system parameters is guaranteed to be random, one can rely on a low-granularity assumption. For example, consider an anonymous payment system where the bank chooses the system parameters. To base the security of such a system a-priori on a low-granularity assumption

[2]Note that SE and IAE are originally called Squaring Diffie-Hellman (Wolf 1999) and Inverted-Additive Diffie-Hellman (MacKenzie 2001), respectively. They are renamed here for consistency and clarity reasons.

would not be appropriate. A cheating bank might try to choose a weak group with trapdoors (easy problem instances) to violate the anonymity of the customer. Such a cheater strategy might be possible even if the low-granular assumption holds: The assumption would ensure that the overall number of easy problem instances is asymptotically negligible (in respect to the security parameter). Nonetheless, it would not rule out that there are infinitely many weak groups! However, if we choose the system parameters of the payment system through a random yet verifiable process we can resort to a weaker assumption with lower granularity. To our knowledge no paper on anonymous payment systems addresses this issue properly. Granularity was also overlooked in different contexts, e.g., Boneh (1998) ignores the fact that low-granular assumptions are not known to be random self-reducible and comes to a wrong conclusion regarding the correctness of a certain self-corrector.

In this chapter, we show that varying granularity can lead to surprising results. We extend the results of Wolf (1999) to the problem class IE, i.e., we prove statements on relations between IE, DH and SE for both computational and decisional variants in the setting of Wolf (1999), which corresponds to the high-granular case. We then consider medium granularity (with other parameters unchanged) and show the impact: We prove that the decisional IE and SE assumptions are equivalent for medium granularity whereas this is provably not possible for their high-granular variants, at least not in the generic model (Shoup 1997). We also show that reductions between computational IE, SE and DH can offer much better concrete security for medium granularity than their high-granular analogues.

The rest of this chapter is structured as follows: In the next section, we define the basic terminology. Section 2.2 introduces the classification of discrete-logarithm-based assumptions, and in Section 2.3, we see how this classification can be used to concisely yet precisely describe assumptions and relations among them. Section 2.4 focuses on the granularity and its impact. In sections 2.5 and 2.6, we review existing results on the relation between Diffie-Hellman (DH) and Square Exponent (SE) assumptions. We introduce a new assumption, called Inverse Exponent (IE) assumption, and extend the existing results by proving similar relations between IE, DH and SE. Further, these sections consider the impact of the granularity parameter on the relation between assumptions.

2.1 Terminology

2.1.1 General Notational Conventions

By $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots\}$ and $(\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots)$ we denote the *set* and the *sequence* consisting of the elements a, b, c, ... By specifying a set as $\{f(v_1, \ldots, v_n) \mid \mathsf{pred}(v_1, \ldots, v_n)\}$ we mean the set of elements we get by evaluating the formula f with any instantiation of the n free variables $v_1, \ldots, v_n$ which fulfills the predicate pred, e.g., $\{(v, v^2) \mid v \in \mathbb{N}\}$ denotes the set of all tuples which contain a natural number and its square. Similarly, we define $(f(v_1, \ldots, v_n) \mid \mathsf{pred}(v_1, \ldots, v_n))$ to be the sequence of elements we get by evaluating the formula f with any instantiation of the n free variables $v_1, \ldots, v_n$ which fulfills the predicate pred. The elements are ordered according to some arbitrary but fixed order relation on the (instantiated) argument tuples $(v_1, \ldots, v_n)$. For example, $((v, v^2) \mid v \in \mathbb{N})$ denotes the infinite sequence of all tuples which contain a natural number and its square, and where the sequence is ordered, e.g., using the standard order $<$ on $\mathbb{N}$ and the value of v as the sort index.

The evaluation and following *assignment* of an expression expr to a variable v is denoted by $v \leftarrow \mathsf{expr}$. By $v \overset{\mathcal{R}}{\leftarrow} S$ we mean the assignment of a uniformly chosen random element from the set S to variable v. Similarly, $v \in_{\mathcal{R}} S$ denotes that v is a uniformly distributed random element from set S. Finally, by $\mathsf{t} := \mathsf{expr}$ we mean that by definition the term t is equal to expr.

Simple *random variables* are specified as $v \overset{\mathcal{R}}{\leftarrow} S$ as mentioned above. To specify more complicated random variables, we use the following notation: $(f(v_1, \ldots, v_n) :: \mathsf{assign}(v_1, \ldots, v_n))$. By this we mean the random variable having a structure as defined by the formula f and a *probability space* as induced by binding the n free variables $v_1, \ldots, v_n$ via the assignment rule assign, e.g., $((v, v^2) :: v \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_n)$ denotes the random variable consisting of a tuple which contains an integer and its square where the integer is uniformly chosen from $\mathbb{Z}_n$. Similarly, $\{f(v_1, \ldots, v_n) :: \mathsf{assign}(v_1, \ldots, v_n)\}$ defines an ensemble of random variables indexed by the free variables v_i which are left unspecified in the assignment rule assign and which have by definition domain $\mathbb{N}$, e.g., $\{(v, v^k) :: v \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_n\}$ denotes the ensemble of random variables consisting of a tuple which contain an integer and its k-th power where the integer is uniformly chosen from $\mathbb{Z}_n$ and the natural number k is the index of the ensemble. Finally, let v be some arbitrary random variable or random variable ensemble. Then, $[v]$ denotes the set of all possible values of v.

To specify *probabilities*, we use the notation $\mathbf{Prob}[\mathsf{pred}(v_1, \ldots, v_n) :: \mathsf{assign}(v_1, \ldots, v_n)]$. This denotes the probability that the predicate pred holds when the probability is taken over a probability space defined by the formula assign on the n free variables v_i of the predicate pred. For example, $\mathbf{Prob}[v \equiv 0 \pmod 2 :: v \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_n]$ denotes the probability that a random element of $\mathbb{Z}_n$ is even.

For convenience, by log we always mean the logarithm to the base two.

2.1.2 Asymptotics

Cryptographic assumptions are always expressed asymptotically in a **security parameter** $k \in \mathbb{N}$. To classify the asymptotic behavior of functions

$\mathbb{N} \rightarrow \mathbb{R}^*$ (with $\mathbb{R}^*$ denoting the set of all non-negative real numbers) we require the following definitions.

We can extend ordinary relation operators $op \in \{<, \leq, =, >, \geq\}$ on elements of $\mathbb{R}^*$ to asymptotic relation operators op_∞ on functions f_1 and f_2 defined as above as follows:

$$f_1(k) \;\; op_\infty \;\; f_2(k) \;:=\; \exists k_0 \, \forall k > k_0 : f_1(k) \; op \; f_2(k).$$

The corresponding negation of the asymptotic relation operators is then denoted by $\not<_\infty$, $\not\leq_\infty$, $\neq_\infty$, $\not\geq_\infty$, and $\not>_\infty$, respectively.

For example, $f_1(k) <_\infty f_2(k)$ means that f_1 is asymptotically strictly smaller than f_2 and $f_1(k) \not\geq_\infty f_2(k)$ means that f_1 is not asymptotically larger or equal to f_2, i.e., for each k_0 there is a $k_1 > k_0$ such that $f_1(k_1) < f_2(k_1)$. However, note that the $f_1(k) \not\geq_\infty f_2(k)$ does not imply $f_1(k) <_\infty f_2(k)$!

Let $\mathsf{poly}(v)$ be the class of **univariate polynomials** with variable v and non-negative coefficients, i.e., $\mathsf{poly}(v) := \{\sum_{i=0}^{d} a_i v^i \mid d \in \mathbb{N}_0 \;\wedge\; a_i \in \mathbb{N}_0\}$. Furthermore, let $\mathsf{poly}(v_1, \ldots, v_n)$ be the class of **multivariate polynomials** with n variables v_j and non-negative coefficients, i.e., $\mathsf{poly}(v_1, \ldots, v_n) := \{\sum_{i=0}^{d} \sum_{j=1}^{|D_i|} a_{ij} \prod_{l=1}^{n} {v_l}^{d_{ijl}} \mid d \in \mathbb{N}_0 \wedge a_{ij} \in \mathbb{N}_0 \wedge (d_{ij1}, \ldots, d_{ijn}) \in D_i^n\}$ where $D_i^n := \{(d_l \mid l \in \{1, \ldots, n\}) \mid d_l \in \mathbb{N}_0 \wedge \sum_{l=1}^{n} d_l = i\}$. Based on this we can define the following useful classes of functions:

A **negligible** function $\epsilon(k)$ is a function where the inverse of any polynomial is asymptotically an upper bound, i.e., $\forall d > 0 \; \exists k_0 \; \forall k > k_0 : \epsilon(k) < 1/k^d$. We denote this by $\epsilon(k) <_\infty 1/\mathsf{poly}(k)$. If $\epsilon(k)$ cannot be upper bounded in such a way, we say $\epsilon(k)$ is **not negligible** and we denote this by $\epsilon(k) \not<_\infty 1/\mathsf{poly}(k)$.

A **non-negligible** function $f(k)$ is a function which asymptotically can be lower bounded by the inverse of some polynomial, i.e., $\exists d > 0 \; \exists k_0 \; \forall k > k_0 : f(k) \geq 1/k^d$. We denote this by $f(k) \geq_\infty 1/\mathsf{poly}(k)$.[3] If $f(k)$ cannot be lower bounded in such a way we say $f(k)$ is **not non-negligible** and denote this by $f(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

Non-negligible functions are — when seen as a class — closed under multivariate polynomial composition, i.e., $\forall n \in \mathbb{N} \; \forall i \in \{1, \ldots, n\} \; \forall p \in \mathsf{poly}(v_1, \ldots, v_n) \setminus \{0_{\mathsf{poly}}\} \; \forall f_i \geq_\infty 1/\mathsf{poly}(k) : \;\; p(f_1, \ldots, f_n) \geq_\infty 1/\mathsf{poly}(k)$ where 0_{poly} denotes the null polynomial. This holds also for negligible functions if there is no non-zero constant term in the polynomial, i.e., we select only elements from the class $\mathsf{poly}(v_1, \ldots, v_n)$ where a_{01} is zero. For not negligible and not non-negligible functions this holds solely for univariate polynomial composition. Finally, the addition (multiplication) of a non-negligible and a not negligible function is a non-negligible (not negligible) function. Similarly, the addition of a negligible and a not non-negligible

[3]Note that not negligible is *not* the same as non-negligible, there are functions which are neither negligible nor non-negligible!

function is a not non-negligible function. The multiplication of a negligible and a not non-negligible function is a not non-negligible function or even a negligible function if the not non-negligible function can be upper bounded by some polynomial.

2.1.3 Computational Model

The computational model is based on the class $\mathcal{TM}$ of probabilistic *Turing machines* on the binary alphabet $\{0, 1\}$. The **runtime** of a Turing machine M is measured by the number of simple Turing steps from the initial state with given inputs until the machine reaches a final state. This is denoted by $\mathrm{RunTime}(\mathsf{M}(\mathit{inputs}))$. The complexity of a Turing machine is expressed as a function of the bit-length of the inputs encoded on its input tape and defined as the maximum runtime for any input of a given bit-length. To make the definition of the probability spaces more explicit, we model a probabilistic Turing machine always as a deterministic machine with the random coins given as an explicit input $\mathcal{C}$ chosen from the *uniform distribution of infinite binary strings* $\mathcal{U}$. However, we do not consider the randomness when calculating the length of the inputs. The important class of **polynomial-time Turing machines** is the class of machines with polynomial complexity:

$$\begin{aligned}\{\mathcal{A} \;\mid\; & \mathcal{A} \in \mathcal{TM};\\ & \forall d_1;\ \exists d_2;\ \forall k;\\ & \forall \mathit{inputs} \in \{0,1\}^{k^{d_1}};\ \forall \mathcal{C} \in \{0,1\}^{\infty};\\ & :\ \mathrm{RunTime}(\mathcal{A}(\mathcal{C}, \mathit{inputs})) \;<\; k^{d_2}\}\end{aligned}$$

When we use the term **efficient** in the context of algorithms or computation we mean a Turing machine with polynomial complexity. By a **hard problem** we mean the absence of any efficient algorithm (asymptotically) solving that problem.

In some situations, e.g., in a reduction, a machine M has access to some other machines $\mathcal{O}_1, \ldots, \mathcal{O}_n$ and can query them as **oracles**. We denote this by $\mathsf{M}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$. This means that the machine M can write the input tapes of all $\mathcal{O}_i$, run them on that input, and read the corresponding output tapes. However, M does not get access to the internal structure or state of the oracle.

2.1.4 Indistinguishability

Let two families of random variables $X := (X_k \mid k \in \mathbb{N})$ and $Y := (Y_k \mid k \in \mathbb{N})$ be defined over some discrete domain $\mathcal{D}$. They are said to be **computationally indistinguishable** iff there is no efficient distinguishing algorithm D which can distinguish the two asymptotically, i.e., $|\mathbf{Prob}[\mathsf{D}(1^k, X_k) = 1] - \mathbf{Prob}[\mathsf{D}(1^k, Y_k) = 1]|$ is a negligible function in k. This is denoted by $X \overset{c}{\approx} Y$.

X and Y are **statistically indistinguishable** iff the **statistical difference** $\Delta_{(X,Y)}(k) := \sum_{d \in \mathcal{D}} |\mathbf{Prob}[X_k = d] - \mathbf{Prob}[Y_k = d]|$ is a negligible function. This is written as $X \overset{s}{\approx} Y$.

2.1.5 Algebraic Structures

The following terms are related to the algebraic structures underlying an assumption.

Finite cyclic group G: A group is an algebraic structure with a set G of **group elements** and a binary **group operation** $* : G \times G \to G$ such that the following conditions hold:

- the group operation is **associative**, i.e., $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$,
- there is an **identity element** $1 \in G$ such that $a * 1 = a = 1 * a$ for all $a \in G$, and
- for each $a \in G$ there is an **inverse** $a^{-1} \in G$ such that $a * a^{-1} = 1 = a^{-1} * a$.

The **group order** is the cardinality of the set G and is denoted by $|G|$.

In the following, we write group operations always multiplicatively by juxtaposition of group elements; Nonetheless, note that the following results apply — with the appropriate adaption of notation — also to additive groups such as elliptic curves. The **exponentiation** a^x for $a \in G$ and $x \in \mathbb{N}_0$ is then defined as usual as $\overbrace{a \cdots a}^{x \text{ times}}$. The **discrete logarithm** of a given $b \in G$ with respect to a specified base $a \in G$ is the smallest $x \in \mathbb{N}_0$ such that $a^x = b$ or undefined if no such x exists. The **order of a group element** $b \in G$ is the least positive integer x such that $b^x = 1$ or ∞ if no such x exists.

A group G is **finite** if $|G|$ is finite. A group G is **cyclic** if there is a **generator** $g \in G$, such that $\forall b \in G\ \exists! x \in \mathbb{Z}_{|G|} : g^x = b$. The order of all elements in a finite cyclic group G divides $|G|$. In particular, there are exactly $\varphi(d)$ elements of order d (where d is any divisor of $|G|$).

All considered assumptions are based on finite cyclic groups. For brevity, however, we omit the “finite cyclic” in the sequel and refer to them simply as “groups”.

For more information on the relevant abstract algebra we refer you to the book of Lidl and Niederreiter (1997).

Algorithmically, the following is noteworthy: Finding generators can be done efficiently when the factorization of $|G|$ is known; it is possible to perform exponentiations in $O(\log(|G|))$ group operations; and computing inverses can be done in $O(\log(|G|))$ group operations under the condition that $|G|$

is known. For the corresponding algorithms and further algorithms for abstract or concrete groups we refer you to the books of Bach and Shallit (1996) and Menezes, van Oorschot, and Vanstone (1997).

Structure instance SI:A tuple $(G, g_1, \ldots, g_n)$ containing a group G as first element followed by a sequence of one or more generators g_i. This represents the structure underlying a particular problem. We can assume that the structure instance SI (though not necessarily properties thereof such as the order or the factorization of the order) is publicly known.

As a convention we abbreviate g_1 to g if there is only a single generator associated with a given structure instance.

2.1.6 Problems

The following two terms characterize a particular problem underlying an assumption.

Problem family $\mathcal{P}$: A family of abstract relations indexed by their underlying structure instance SI. An example is the family of Diffie-Hellman problems which relate two (secret) numbers x and y, the two (public) values g^x and g^y, and the value g^{xy} where all exponentiations are computed using the generator g specified in SI. We define a problem family by explicitly describing its problem instances as shown in the next paragraph.

Problem instance PI: A list of concrete parameters fully describing a particular instance of a problem family, i.e., a description of the structure instance SI and a tuple $(priv, publ, sol)$ where $priv$ is the tuple of values kept secret from adversaries, $publ$ is the tuple of information publicly known on that problem and sol is the set of possible solutions[4] of that problem instance. When not explicitly stated, we can assume that $priv$ always consists of elements from $\mathbb{Z}_{|G|}$, $publ$ consists of elements from G, and sol is either a set of elements from $\mathbb{Z}_{|G|}$ or from G.

If we take the aforementioned Diffie-Hellman problem for subgroups of $\mathbb{Z}_p^*$ of order q with p and q prime as an example, a problem instance PI_{DH} is defined by a tuple

$$(((\mathbb{Z}^*_{p/q}, p, q), (g)), ((x, y), (g^x, g^y), \{(g^{xy})\}))$$

where $\mathbb{Z}^*_{p/q}$ denotes the parameterized description of the group and its operation, and p, q are the corresponding group parameters. (More details on the group description and parameter are given below when group samplers are introduced.)

[4]The solutions might not be unique, e.g., multiple solution tuples match a given public value in the case of the Representation Problem (See Section 2.2, Parameter 1).

This presentation achieves a certain uniformity of description and allows a generic definition of types of problems, i.e., whether it is a decisional or computational variant of a problem. While this might not be obvious right now, it should become clear at the latest in Section 2.2 below when we give the explicit definition of the different problem families with Parameter 1 and the precise definition of problem types with Parameter 2.

For convenience, we define PI^{SI}, PI^{publ}, PI^{priv} and PI^{sol} to be the projection of a problem instance PI to its structure instance, public, private and solution part, respectively. Picking up again above example, this means ${PI_{DH}}^{SI} := ((\mathbb{Z}^*_{p/q}, p, q), (g))$, ${PI_{DH}}^{priv} := (x, y)$, ${PI_{DH}}^{publ} := (g^x, g^y)$, and ${PI_{DH}}^{sol} := \{g^{xy}\}$, respectively.

2.1.7 Samplers

In the following, we describe different probabilistic polynomial-time algorithms we use to randomly select (sample) various parameters. Note that these samplers cannot be assumed to be publicly known, i.e., to sample from the corresponding domains adversaries have to construct their own sampling algorithms from publicly known information.

Group sampler $SG_{\mathcal{G}}$: A function which, when given a security parameter k as input, randomly selects a group G and returns a corresponding group index. We assume that a group sampler selects groups only of similar nature and type, i.e., there is a general description of a Turing machine which, based on a group index as parameter, implements at least the group operation and the equality test, and specifies how the group elements are represented. An example are the groups pioneered by Schnorr (1991) in his identification and signature schemes and also used in the Digital Signature Standard (DSS) (National Institute of Standards and Technology (NIST) 2000), i.e., unique subgroups of $\mathbb{Z}^*_p$ of order q with p and q prime. The group index would be (p, q) and the description of the necessary algorithms would be taken, e.g., from Menezes, van Oorschot, and Vanstone (1997). Note that, in this example, the group index allows the derivation of the group order and the factorization thereof. However, it cannot be assumed that the group index — the only information besides the description of the Turing machine which will be always publicly known about the group — allows to derive such knowledge on the group order in general.

The set of groups possibly returned by a group sampler, i.e., $[SG_{\mathcal{G}}]$, is called in the sequel a **group family** $\mathcal{G}$ and is required to be infinite. To make the specific group family $\mathcal{G}$ more explicit in the sampler we often label the sampler accordingly as $SG_{\mathcal{G}}$, e.g., for above example the sampler would be named $SG_{\mathbb{Z}^*_{p/q}}$.

Furthermore, the set of possible groups G returned by $SG_{\mathcal{G}}$ for a given fixed security parameter k, i.e., $[SG_{\mathcal{G}}(1^k)]$, is called **group siblings** $\mathcal{G}_{SG(k)}$. This represents the groups of a given family $\mathcal{G}$ with approximately the same "security". We assume that the group operation and equality test for the groups in $\mathcal{G}_{SG(k)}$ can be computed efficiently (in k); yet the underlying problem is supposedly asymptotically hard.

Slightly restricting the class of samplers, we require that the order $|G|$ of all $G \in \mathcal{G}_{SG(k)}$ is approximately the same. In particular, we assume that the order can be bounded in the security parameter, i.e., $\exists d_1, d_2 > 0\ \forall k > 1\ \forall G \in \mathcal{G}_{SG(k)}\ :\ k^{d_1} \leq \log(|G|) \leq k^{d_2}$.[5] For Schnorr signatures, in the example given above, a group sampler might choose the random primes p and q with $|q| \approx 2k$ and $p = rq + 1$ for an integer r sufficiently large to make DL hard to compute in security parameter k. See Menezes, van Oorschot, and Vanstone (1997) and Odlyzko (2000) for the state-of-the-art algorithms for computing discrete logarithms and Lenstra and Verheul (2001) for a methodology on how to choose parameters (as a function of the security parameter k), illustrated concretely for group families such as $\mathbb{Z}_p^*$ or elliptic curves.

Generator sampler Sg: A function which, when given a description of a group G for a fixed group family, randomly selects a generator $g \in G$. We assume that Sg has always access somehow, e.g., via an oracle, to the factorization of the group order. This information is required by the sampler as the group index might not be sufficient to find generators efficiently. This covers the situation where an honest party chooses the group as well as the generator but keeps the factorization of the group order secret. However, it also implies that the factorization of the order should in general be public when the adversary chooses the generators.

Note that the number of generators is $\varphi(|G|)$ and, due to requirements on group orders mentioned above, always super-polynomial in the security parameter k: Given the lower bound $\forall n \geq 5\ :\ \varphi(n) > n/(6 \log(\log(n)))$ (Fact 2.102, Menezes, van Oorschot, and Vanstone 1997) and our size restrictions on $|G|$ we have asymptotically the following relation: $\varphi(|G|)/|G| > 1/O(\log k) > 1/k$.

Problem instance sampler $SPI_{\mathcal{P}}$: A function indexed by a problem family $\mathcal{P}$ which, when given a description of a structure instance SI as input, randomly selects a problem instance PI. Similarly to Sg, we assume that $SPI_{\mathcal{P}}$ gets always access to the factorization of the group order. Furthermore, $SPI_{\mathcal{P}}$ gets also access to the discrete logarithms among the different

[5]This restriction is mainly for easier treatment in various reductions and is not a hindrance in practical applications: On the one hand, the upper bound is tight (larger groups cannot have efficient group operations). On the other hand, the common approach in choosing a safe group order, e.g., as proposed by Lenstra and Verheul (2001), will relate the group order closely to the negligible probability of guessing a random element correctly, and hence result in exponential order.

generators in SI. This is required for some problem families, e.g., IE and RP(n).[6] In most cases and in all examples considered here, this corresponds to randomly selecting *priv* and deriving *publ* and *sol* from it. For example, a problem instance sampler SPI_{DH} for the Diffie-Hellman problem family would return a tuple $(SI, ((x, y), (g^x, g^y), \{(g^{xy})\}))$ with x and y randomly picked from $\mathbb{Z}_{|G|}$ and g taken from SI. When the specific problem family $\mathcal{P}$ is not relevant or clear from the context we abbreviate $SPI_{\mathcal{P}}$ to SPI.

Note that the running time of the samplers is always polynomially bounded in the security parameter k.[7]

If not stated explicitly we can always assume a uniform distribution of the sampled elements in the corresponding domains, as done in most cases of cryptographic applications. The rare exceptions are cases such as the c-DLSE assumption (Patel and Sundaram 1998; Gennaro 2000), an assumption on the difficulty of taking discrete logarithms when the random exponents are taken only from a small set, i.e., $\mathbb{Z}_{2^c}$ with $c = \omega(\log\log|G|)$ instead of $\mathbb{Z}_{|G|}$, or the Diffie-Hellman Indistinguishability (DHI) assumptions introduced by Canetti (1997). The difficulty of these assumptions is not necessarily their individual specification, e.g., c-DLSE could be defined by suitably restricting the domain of the *sol* part of a DL problem instance. The deeper problem is that proving relations among these and other assumptions seems to require very specific tools, e.g., for randomization and analysis of resulting success probabilities, and are difficult to generalize as desirable for a classification as presented here. However, it might be worthwhile to investigate in future work whether these cases can be addressed by treating the sampling probability distribution as an explicit parameter of the classification. To make this extension promising, one would have to first find a suitable categorization of sampling probability distributions which: (1) captures the assumptions currently not addressed, and (2) offers tools assisting in proving reductions in a generalizable fashion.

2.2 Parameters of DL-based Assumptions

In defining assumptions, a cryptographer has various degrees of freedom related to the concrete mathematical formulation of the assumption, e.g.,

[6]As a practical consequence, it means that for such problem families either this information has to be public, e.g., the group index should allow the derivation of the factorization of the order, or the group and generators are chosen by the same party which samples the problem instance.

[7]For SG this holds trivially as we required samplers to be polynomial-time in their inputs. The input of Sg are the outputs of a single call of a machine (SG) polynomially bounded by k and, therefore, can be polynomially upper bounded in k. As the class of polynomials is closed under polynomial composition this holds also for Sg and, using similar reasoning, also for SPI.

what kind of attackers are considered or over what values the probability spaces are defined.

To shed some light in these degrees of freedom we classify intractability assumptions for problems related to DL and relevant to many cryptographic applications. We identify the following orthogonal parameters. Additionally, we give for each of these parameters in a corresponding sublist different values which can produce significantly different assumptions.

1. **Problem family**: The following problem families are useful (and often used) for cryptographic applications. As mentioned in Section 2.1.6 we define the problem family (or more precisely their problem instances) by a structure instance SI (described abstractly by G and g_i's) and a tuple $(priv, publ, sol)$:

   DL (Discrete Logarithm):

   $$PI_{\mathrm{DL}} \;:=\; ((G,g),((x),(g^x),\{(x)\})).$$

   DH (Diffie-Hellman):

   $$PI_{\mathrm{DH}} \;:=\; ((G,g),((x,y),(g^x,g^y),\{(g^{xy})\}))$$

   GDH(n) (Generalized Diffie-Hellman for $n \geq 2$):

   $$\begin{aligned} PI_{\mathrm{GDH}(n)} \;:=\; & ((G,g),((x_i \mid i \in \{1,\dots,n\}),\\ & (g^{\prod_{i\in I} x_i} \mid I \subset \{1,\dots,n\}),\ \{(g^{\prod_{i=1}^{n} x_i})\})), \end{aligned}$$

   where n is a fixed parameter.[8]

   SE (Square-Exponent):

   $$PI_{\mathrm{SE}} \;:=\; ((G,g),((x),(g^x),\{(g^{x^2})\})).$$

   IE (Inverse-Exponent):

   $$PI_{\mathrm{IE}} \;:=\; ((G,g),((x),(g^x),\{(g^{x^{-1}})\})).$$

   Note that for elements $x' \in \mathbb{Z}_{|G|} \setminus \mathbb{Z}^*_{|G|}$ the value x^{-1} is not defined. Therefore, $PI_{\mathrm{IE}}{}^{priv}$ $(= (x))$ has to contain an element of $\mathbb{Z}^*_{|G|}$, contrary to the previously mentioned problem families where $priv$ consists of elements from $\mathbb{Z}_{|G|}$.

[8]A slightly generalized form GDH($n(k)$) would allow n to be a function in k. However, this function can grow at most logarithmically (otherwise the tuple would be of super-polynomial size!)

RP(n) (Representation Problem for $n \geq 2$):

$$\begin{aligned} PI_{\mathrm{RP}(n)} &:= ((G, g_1, \ldots, g_n),\ ((x_i \mid i \in \{1, \ldots, n\}),\ (\prod_{i=1}^{n} g_i^{x_i}), \\ &\qquad \{(x'_i \mid i \in \{1, \ldots, n\}) \mid (x'_i \in \mathbb{Z}_{|G|}) \wedge (\prod_{i=1}^{n} g_i^{x'_i} = \prod_{i=1}^{n} g_i^{x_i})\})), \end{aligned}$$

where n is a fixed parameter.[9]

IAE (Inverted Additive Exponent Problem):

$$PI_{\mathrm{IAE}} := ((G, g), ((x, y), (g^{1/x}, g^{1/y}), \{(g^{1/(x+y)})\})).$$

Similar to IE, $PI_{\mathrm{IAE}}{}^{priv}$ $(= (x, y))$ consists of elements from $\mathbb{Z}^*_{|G|}$. Additionally, it has to hold that $x + y \in \mathbb{Z}^*_{|G|}$.

SDH (Strong Diffie-Hellman Problem):

$$PI_{\mathrm{SDH}} := ((G, g), ((x, y), (g^x, g^{1/x}, g^y), \{(g^{xy})\})).$$

Similar to IE, the domain of x is restricted to $\mathbb{Z}^*_{|G|}$.

2. **Problem type**: Each problem can be formulated in three variants.

C (Computational): For a given problem instance PI an algorithm $\mathcal{A}$ succeeds if and only if it can solve PI, i.e., $\mathcal{A}(\ldots, PI^{publ}) \in PI^{sol}$. For the Diffie-Hellman problem family this means that $\mathcal{A}$ gets g^x and g^y as input and the task is to compute g^{xy}.
There is a small twist in the meaning of $\mathcal{A}(\ldots, PI^{publ}) \in PI^{sol}$: As $|G|$ is not necessarily known, $\mathcal{A}$ might not be able to represent elements of $\mathbb{Z}_{|G|}$ required in the solution set uniquely in their "principal" representation as elements of $\{0, \ldots, |G| - 1\}$. Therefore, we allow $\mathcal{A}$ in these cases to return elements of $\mathbb{Z}$ and we implicitly reduce them $\bmod |G|$.

D (Decisional): For a given problem instance PI_0, a random problem instance PI_1 chosen with the same structure instance using the corresponding problem instance sampler and a random bit b, the algorithm $\mathcal{A}$ succeeds if and only if it can decide whether a given solution chosen randomly from the solution set of one of the two problem instances corresponds to the given problem instance, i.e., $\mathcal{A}(\ldots, PI^{publ}, sol_c)) = b$ where $sol_c \overset{\mathcal{R}}{\leftarrow} PI_b{}^{sol}$.[10] For the Diffie-Hellman problem family this means that $\mathcal{A}$ gets g^x, g^y and g^c (where c is either xy or $x'y'$ for $x', y' \in_{\mathcal{R}} \mathbb{Z}_{|G|}$) as input and the task is to decide whether g^c is g^{xy} or not.

[9]Similar to GDH(n) one can also define here a slightly generalized form RP($n(k)$). In this case, one can allow $n(k)$ to grow even polynomially.

[10]This definition differs subtly from most other definitions of decisional problems: Here the distribution of the challenge sol_c is for $b = 1$, i.e., the random "wrong" challenge,

M (Matching): For two given problem instances PI_0 and PI_1 and a random bit b, the algorithm $\mathcal{A}$ succeeds if and only if it can correctly associate the given solutions with their corresponding problem instances, i.e., $\mathcal{A}(\ldots, PI_0{}^{publ}, PI_1{}^{publ}, sol_b, sol_{\bar{b}}) = b$ where $sol_0 \xleftarrow{\mathcal{R}} PI_0{}^{sol}$ and $sol_1 \xleftarrow{\mathcal{R}} PI_1{}^{sol}$. For the Diffie-Hellman problem family this means that $\mathcal{A}$ gets g^{x_0}, g^{y_0}, g^{x_1}, g^{y_1}, $g^{x_b y_b}$ and $g^{x_{\bar{b}} y_{\bar{b}}}$ as input and the task is to predict b.

Initially, only computational assumptions, which follow naturally from informal security requirements, were considered in cryptography. For example, a key exchange protocol should prevent the complete recovery of the key which is usually the solution part of an assumption. However, the later formalization of security requirements, in particular semantic security (Goldwasser and Micali 1984), requires often the indistinguishability of random variables. Taking again the example of a key exchange protocol, it was realized that if you do not want to make strong requirements on the particular use of exchanged keys but allow the modular and transparent composition of key exchange protocols with other protocols, e.g., for secure sessions, it is essential that an exchanged key is indistinguishable from random keys, i.e., not even partial information on the key is leaked. While this does not necessarily imply decisional assumptions, such assumptions might be indispensable for efficient systems: There is an efficient encryption scheme secure against adaptive adversaries under the Decisional Diffie-Hellman assumption (Cramer and Shoup 1998). Nonetheless, no system is known today which achieves the same security under a similar computational assumption in the standard model.[11] Finally, the matching variant was introduced by Frankel, Tsiounis, and Yung (1996) where it showed to be a useful tool to construct fair off-line cash. Handschuh, Tsiounis, and Yung (1999) later showed that the matching and the decisional variants of Diffie-Hellman are equivalent, a proof which is adaptable also to other problem families.

3. **Group family**: Various group families are used in cryptographic applications. The following list contains some of the more common ones.

according to the distribution of *sol* induced by *SPI* whereas most others consider it to be a (uniformly chosen) random element of G. Taking DIE or DDH with groups where the order has small factors these distributions are quite different! Conceptually, the definition here seems more reasonable, e.g., in a key exchange protocol you distinguish a key from an arbitrary key, not an arbitrary random value. It also addresses nicely the case of samplers with non-uniform distributions.

[11]There are efficient schemes known in the random oracle model (Bellare and Rogaway 1993), e.g., OAEP (Bellare and Rogaway 1995; Boneh 2001; Shoup 2001; Fujisaki et al. 2001). However, this model is strictly weaker than the standard model and has a number of caveats (Canetti, Goldreich, and Halevi 1998).

For brevity we do not mention the specific parameter choice as a function of k. We refer you to, e.g., Lenstra and Verheul (2001), for concrete proposals:

$\mathbb{Z}_p^*$: The multiplicative groups of integers modulo a prime p with group order $\varphi(p)$ having at least one large prime factor. The group index is p.

$\mathbb{Z}_{p/q}^*$: The subgroups of $\mathbb{Z}_p^*$ of prime order q. The group index is the tuple (p, q).

$\mathbb{Z}_n^*$: The multiplicative groups of integers modulo a product n of two (or more) large primes p and q with $p-1$ and $q-1$ containing at least one large prime factor. The group index is n.[12]

$\mathbb{QR}_n^*$: The subgroups of $\mathbb{Z}_n^*$ formed by the quadratic residues with n product of two large safe[13] primes. The group index is n.

$E_{a,b}/\mathbb{F}_p$: The elliptic curves over $\mathbb{F}_p$ with p and $|E_{a,b}|$ prime with group index (a, b, p).

The concrete choice of a group family has significant practical impact on aspects such as computation or bandwidth efficiency or suitability for a particular hardware but discussing this goes beyond the scope of this document, namely comparing assumptions. In this scope, it is mostly sufficient to classify simple and abstract properties of the chosen family and the public knowledge about a given group. We established the following two general criteria:

(a) The factorization of the group order contains

lprim: large prime factors (at least one). Formally, it has to hold that (with $\mathbb{P}$ being the set of prime numbers):

$$\forall d > 0\, \exists k_0\, \forall k > k_0\, \forall G \in \mathcal{G}_{SG(k)}\, \exists p \in \mathbb{P}\, \exists r \in \mathbb{N} : |G| = pr \wedge p > k^d,$$

nsprim: no small prime factor. Formally, the following has to hold:

$$\forall d > 0\, \exists k_0\, \forall k > k_0\, \forall G \in \mathcal{G}_{SG(k)}\, \nexists p \in \mathbb{P}\, \exists r \in \mathbb{N} : |G| = pr \wedge p < k^d,$$

prim: only a single and large prime factor.

Note that this is a strict hierarchy and later values imply earlier ones. There would also be an obvious fourth value, namely the order contains no large factor. However, in such cases no reasonable DL based assumption seems possible (Pohlig and Hellman 1978; Pollard 1978).

[12] This means that the order of the group is secret if we assume factoring n is hard.

[13] A prime p is a safe prime when $p - 1 = 2p'$ and $p' \in \mathbb{P}$.

(b) The group order is publicly

$\overline{\text{o}}$: unknown,

o: known,

fct: known including its complete[14] factorization.

We assume any such public knowledge to be encoded in the description returned by a group sampler SG. Note that in practice the group order is never completely unknown: at least an efficiently computable upper bound $B(|G|)$ can always be derived, e.g., from the bit-length of the representation of group elements. This can be exploited, e.g., in achieving **random self-reducibility**[15] (Blum and Micali 1984) for DDH even in the case where the order is not known (Boneh 1998).

The cryptographic application will determine which of above properties hold, e.g., a verifiable group generation will quite likely result in a publicly known factorization.

Furthermore, note that the group families given above implicitly fix the properties of the group order factorization ($\mathbb{Z}_p^*$: lprim; $\mathbb{Z}_{p/q}^*$: prim; $\mathbb{Z}_n^*$: lprim; $\mathbb{QR}_n^*$: nsprim; $E_{a,b}/\mathbb{F}_p$: prim), and the public knowledge about it ($\mathbb{Z}_p^*$: o; $\mathbb{Z}_{p/q}^*$: fct; $\mathbb{Z}_n^*$: $\overline{\text{o}}$; $\mathbb{QR}_n^*$: $\overline{\text{o}}$; $E_{a,b}/\mathbb{F}_p$: fct).

4. **Computational capability of adversary**: Potential algorithms solving a problem have to be computationally limited for number-theoretic assumptions to be meaningful (otherwise we could never assume their nonexistence). Here, we only consider probabilistic polynomial-time algorithms (called **adversaries** in the following). The adversary can be of

u (Uniform complexity): There is a single probabilistic Turing machine $\mathcal{A}$ which for any given finite input returns a (not necessarily correct) answer in polynomial time in its input length. As the complexity of Turing machines is measured in the bit-length of the inputs the inputs should be neither negligible nor super-polynomial in the security parameter k, otherwise the algorithm might not be able to write out the complete desired output or might become too powerful. To address this issue one normally passes an additional input 1^k to $\mathcal{A}$ to lower bound the complexity and makes sure that the other inputs can be polynomially upper

[14]If the order is known then small prime factors can always be computed. Insofar the case here extends the knowledge about the factorization also to large prime factors.

[15]Informally, a problem is random self-reducible if solving *any* problem instance can be reduced to solving the problem on a *random* instance, i.e., when given an instance x we can efficiently randomize it to a random instance $x_{\mathcal{R}}$ and can efficiently derive (derandomize) the solution for x from the solution returned by an oracle call on $x_{\mathcal{R}}$.

bounded in k. In all cases considered here, the inputs in the assumptions are already proportional to the security parameters, see remarks on the size of groups and on the runtime of samplers in Section 2.1.7. Therefore we can safely omit 1^k in the inputs of $\mathcal{A}$.

n (Non-uniform complexity): There is an (infinite) family of Turing machines $(\mathcal{A}_k \mid k \in \mathbb{N})$ with description size and running time of $\mathcal{A}_k$ bounded by a polynomial in the security parameter k.[16] Equivalent alternatives are a (single) Turing Machine with polynomial running time and an additional (not necessarily computable) family of auxiliary inputs polynomially bounded by the security parameter, or families of circuits with the number of gates polynomially bounded by the security parameter,[17] respectively.

Uniform assumptions are (in many cases strictly) weaker than corresponding non-uniform assumptions as any uniform algorithm is also a non-uniform one. Furthermore, all uniform black-box reductions map to the non-uniform case (but not necessarily vice-versa!) and henceforth most uniform proofs should map to their non-uniform counterpart. This makes uniform assumptions preferable over non-uniform assumptions (e.g., honest users are normally uniform and weaker assumptions are always preferable over stronger ones). However, uniform assumptions also assume uniform adversaries which is a weaker adversary model than the model considering non-uniform adversaries. Furthermore, there are proofs which only work in a non-uniform model.

Further, potentially interesting yet currently ignored, attacker capabilities would be bounds on space instead of (or in addition) to time. Adaptive adversaries do not seem of concern for pure assumptions.

Ideally, one would consider larger, i.e., less restricted, classes of adversaries than the strictly polynomial-time one following from the definition from Section 2.1.3. It would seem more natural, e.g., to require polynomial behavior only on inputs valid for a given assumption or to allow algorithms, e.g., Las Vegas algorithms, with no a-priori bound on the runtime.[18] Unfortunately, such classes are difficult to define properly and even harder to work with. However, as for each adversary of these classes, there seems to be a closely related (yet not necessarily black-box constructible) strictly polynomial-time adversary with

[16] The remarks on input length and runtime mentioned above for uniform complexity also apply here.

[17] In the case of circuits the bound on the running time automatically follows and does not have to be explicitly restricted.

[18] However, we would have to restrict the considerations to polynomial time runs when measuring the success probability of adversaries.

similar success probability, this restriction seems of limited practical relevance.

5. **"Algebraic knowledge"**: A second parameter describing the adversary's computational capabilities relates to the adversary's knowledge on the group family. It can be one of the following:

   $\boldsymbol{\sigma}$ (Generic): This means that the adversary does not know anything about the structure (representation) of the underlying algebraic group. More precisely this means that all group elements are represented using an **encoding function** $\sigma(\cdot)$ drawn randomly from the set $\Sigma_{G,g}$ of bijective[19] functions $\mathbb{Z}_{|G|} \to G$. Group operations can only be performed via the addition and inversion[20] oracles $\sigma(x+y) \leftarrow \sigma_+(\sigma(x), \sigma(y))$ and $\sigma(-x) \leftarrow \sigma_-(x)$ respectively, which the adversary receives as a black box (Shoup 1997; Nechaev 1994) together with $\sigma(1)$, the generator.

   If we use σ in the following, we always mean the (not further specified) random encoding used for generic algorithms with a group G and generator g implied by the context. In particular, by $\mathcal{A}^\sigma$ we refer to a generic algorithm. To prevent clutter in the presentation, we do not explicitly encode group elements passed as inputs to such generic algorithms. However, they should all be considered suitable encoded with σ.

   (marked by absence of σ) (Specific): In this case the adversary can also exploit special properties (e.g., the encoding) of the underlying group.

   This separation is interesting for the following reasons:

   - Tight lower bounds on the complexity of some DL-based assumptions can lead to provably hard assumptions in the generic model (Shoup 1997; Maurer and Wolf 1998b). No such results are known in the standard model. However, similar to the random oracle model (Bellare and Rogaway 1993) the generic model is idealized and related pitfalls lure when used in a broader context than simple assumptions (Fischlin 2000).

[19] Others, e.g., Babai and Szemerédi (1984) and Boneh and Lipton (1996), considered the more general case where elements are not necessarily unique and there is a separate equality oracle. However, that model is too weak to cover some important algorithms, e.g., Pohlig and Hellman (1978), which are intuitively "generic". Furthermore, the impossibility results mentioned later still hold when transfered to the more general case.

[20] Computing inverses is usually efficient only when the group order is known. However, note that all impossibility results — the main use of generic adversaries — considered later hold naturally also without the inversion oracle.

- A number of algorithms computing discrete logarithms are generic in their nature. Two prominent ones are Pohlig-Hellman (1978) and Pollard-ρ (1978) paired with Shanks Baby-Step Giant-Step optimization. Furthermore, most reductions are generic.
- However, exploiting some structure in the group can lead to faster algorithms, e.g., for finite fields there is the class of index-calculus methods and in particular the generalized number field sieve (GNFS) (Gordon 1993b; Schirokauer 1993) with sub-exponential expected running time.
- Nonetheless, for many group families, e.g., elliptic curves, no specific algorithms are known which compute the discrete logarithms better than the generic algorithms mentioned above.

Note that a generic adversary can always be transformed to a specific adversary but not necessarily vice-versa. Therefore, a reduction between two generic assumptions is also a reduction between the specific counterparts of the two assumptions. However, proofs of the hardness of generic assumptions or the non-existence of relations among them do *not* imply their specific counterparts!

6. **"Granularity of probability space"**: Depending on what part of the structure instance is a-priori fixed (i.e., the assumption has to hold for all such parameters) or not (i.e., the parameters are part of the probability space underlying an assumption) we can distinguish among the following situations:

l (Low-granular): The group family (e.g., prime order subgroups of $\mathbb{Z}_p^*$) is fixed but not the specific structure instance (e.g., parameters p, q and generators g_i for the example group family given above).

m (Medium-granular): The group (e.g., p and q) but not the generators g_i are fixed.

h (High-granular): The group as well as the generators g_i are fixed.

An assumption defines a family of probability spaces $\mathcal{D}_i$, where the index i is the tuple of k and, depending on granularity, group and generator, i.e., all parameters with an all-quantifier in the assumption statement. Each probability space $\mathcal{D}_i$ is defined over problem instances, random coins for the adversary, and, again depending on granularity, groups and generators. Note that for a given k there are exponentially many $\mathcal{D}_i$. In the sequel we use the term **probability space instance (PSI)** for a single probability space $\mathcal{D}_i$.

7. **Success probability**: This parameter gives an (asymptotic) upper bound on how large a success probability we tolerate from an adversary. The success probability is measured over the family of probability space instances $\mathcal{D}_i$. Violation of an assumption means that there exists an algorithm $\mathcal{A}$ whose success probability $\alpha(k)$ reaches or exceeds this bound for infinitely many k in respect to at least one of the corresponding probability space instances $\mathcal{D}_i$.

   The upper bound and the corresponding adversary can be classified in the following types:

   1 (Perfect): The strict upper bound on the success probability is 1. Therefore, a perfect adversary algorithm $\mathcal{A}$ with success probability $\alpha(k)$ has to solve the complete probability mass of infinitely many $\mathcal{D}_i$, i.e., $\alpha(k) \not<_\infty 1$.

   $(1-1/\mathsf{poly}(k))$ (Strong): The bound is defined by the error probability which has to be non-negligible. Therefore, a strong adversary algorithm $\mathcal{A}$ has to be successful for infinitely many $\mathcal{D}_i$ with overwhelming probability., i.e., if $\alpha(k)$ is the success probability of $\mathcal{A}$ then $1 - \alpha(k) \not>_\infty 1/\mathsf{poly}(k)$.

   ϵ (Invariant): The strict upper bound is a fixed and given constant $0 < \epsilon < 1$. Therefore, the success probability $\alpha(k)$ of an invariant adversary algorithm $\mathcal{A}$ has to be larger than ϵ for infinitely many $\mathcal{D}_i$, i.e., $\alpha(k) \not<_\infty \epsilon$.

   $1/\mathsf{poly}(k)$ (Weak): All non-negligible functions are upper bounds, i.e., only negligible success probabilities are tolerated. Therefore, a weak adversary algorithm $\mathcal{A}$ has to be successful with a not negligible fraction of the probability mass of $\mathcal{D}_i$ for infinitely many $\mathcal{D}_i$, i.e., if $\alpha(k)$ is the success probability of $\mathcal{A}$ then $\alpha(k) \not<_\infty 1/\mathsf{poly}(k)$.

   An assumption requiring the nonexistence of perfect adversaries corresponds to worst-case complexity, i.e., if the assumption holds then there are at least a few hard instances. However, what is a-priori required in most cases in cryptography is a stronger assumption requiring the nonexistence of even weak adversaries, i.e., if the assumption holds then most problem instances are hard.

The classification given above is certainly not exhaustive. The exploration of new problem families, e.g., related to arbitrary multivariate functions in the exponents as investigated by Kiltz (2001), might require additional values for the existing parameters. This can be done without much impact on the classification itself and other results. However, the need for a new dimension such as adding probability distributions as a separate parameter (see Section 2.1.7) would be of much larger impact. Nevertheless, from the current experience, above classification seems quite satisfactory.

2.3 Defining Assumptions

Using the parameters and corresponding values defined in the previous section, we can define intractability assumptions in a compact and precise way.

The notation for a given assumption is

$$\text{\$s-\$t\$}\mathcal{P}^{\text{\$a}}(\text{c:\$c; g:\$g; f:\$}\mathcal{G})$$

where for each parameter there is a placeholder $X which is instantiated by the labels corresponding to the value of that parameter in the given assumption. The placeholders and values (with $-$ denoting that this value can be absent in the notation and has the same meaning as a corresponding wild card) are as follows:

- $s: The algorithm's success probability ($\$s \in \{1, (1-1/\mathsf{poly}(k)), \epsilon, 1/\mathsf{poly}(k)\}$).
- $t: The problem type ($\$t \in \{\mathrm{C}, \mathrm{D}, \mathrm{M}\}$).
- $$\mathcal{P}$: The problem family ($\$\mathcal{P} \in \{\mathrm{DL}, \mathrm{DH}, \mathrm{GDH}(n), \mathrm{SE}, \mathrm{IE}, \mathrm{RP}(n), \mathrm{IAE}, \mathrm{SDH}\}$).
- $a: The algebraic knowledge of the algorithm ($\$a \in \{\sigma, -\}$).
- $c: The algorithm's complexity ($\$c \in \{u, n\}$).
- $g: The granularity of the probability space ($\$g \in \{h, m, l\}$).
- $$\mathcal{G}$: The group family ($\$\mathcal{G} \in \{\mathrm{lprim}, \mathrm{nsprim}, \mathrm{prim}, -\} \times \{\overline{\mathrm{o}}, \mathrm{o}, \mathrm{fct}, -\} \times \{\mathbb{Z}_p^*, \mathbb{Z}_{p/q}^*, \mathbb{Z}_n^*, \mathbb{QR}_n^*, E_{a,b}/\mathbb{F}_p, -\}$).[21]

This is best illustrated in an example: The term

$$1/\mathsf{poly}(k)\text{-DDH}^{\sigma}(\text{c:u; g:h; f:prim})$$

denotes the decisional (D) Diffie-Hellman (DH) assumption in prime-order groups (f:prim) with weak success probability ($1/\mathsf{poly}(k)$), limited to generic algorithms (σ) of uniform complexity (c:u), and with high granularity (g:h).

To refer to classes of assumptions we use **wild cards** ($*$) and sets ($\{\cdots\}$) of parameter values, e.g.,

[21] The parameters for $\mathcal{G}$ are not completely orthogonal in the sense that some combinations do not exist, e.g., $(\mathrm{prim}, \cdot, \mathbb{QR}_n^*)$, and some result in nonsensical assumptions, e.g., $(\cdot, \mathrm{fct}, \mathbb{Z}_n^*)$. Nonetheless, the assumptions still can be defined and insofar this is not really of concern here.

$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CDH}^{\sigma}(\text{c:u; g:h; f:}*)$$

denotes the class of computational (C) Diffie-Hellman (DH) assumptions with uniform complexity (c:u), limited to generic algorithms (σ), with high-granular probability space (g:h), with some error ($\{(1 - 1/\mathsf{poly}(k)), \epsilon, 1/\mathsf{poly}(k)\}$) and based on an arbitrary group family (f:$*$).

Let us turn now to the meaning of an assumption described by above notation: By stating that an assumption \$s-\$t\$$\mathcal{P}^{\$\text{a}}$(c:\$c; g:\$g; f:\$$\mathcal{G}$) holds, we believe that asymptotically no algorithm of complexity \$c and algebraic knowledge \$a can solve (random) problem instances of a problem family \$$\mathcal{P}$ with problem type \$t chosen from groups in \$$\mathcal{G}$ with sufficient (as specified by \$s) success probability where the probability space is defined according to granularity \$g.

The precise and formal definitions follow naturally and quite mechanically. In defining an assumption we always require a bound k_0 for the asymptotic behavior which says that beyond that bound no adversary will be successful. As further "ingredients" there are polynomials defined by their maximal degree d_1, d_2 and d_3 which bind the error probability, time and description of programs, respectively. Finally, we require a machine (or family thereof) $\mathcal{A}$ ($\mathcal{A}_i$) trying to solve the problem, and various quantifiers specifying (using the various samplers) the required parameters for a problem instance PI to solve.

Finally, we denote the class of uniform complexity adversaries by $\mathcal{UPTM}$ and the corresponding class of generic adversaries by $\mathcal{UPTM}^{\sigma}$. The class of non-uniform complexity and generic non-uniform complexity adversaries is denoted similarly by $\mathcal{NPTM}$ and $\mathcal{NPTM}^{\sigma}$, respectively.

To illustrate the formal details of assumptions and to provide a feel for the various parameters we offer three sets of examples. In each set we vary one of the parameters, namely: (1) the computational complexity, (2) the less obvious and often overlooked granularity parameter, and (3) the success probability. The complete details on how to derive the formal assumption statement from the parameters can be found in Appendix A:

1. Weak computational DL assumptions in the generic model, a group order with at least one large prime factor and the two variants of complexity measures (see Parameter 4). Remember that $PI_{\text{DL}} := (SI, ((x), (g^x), \{(x)\}))$, $PI_{\text{DL}}{}^{publ} := (g^x)$ and $PI_{\text{DL}}{}^{sol} := \{(x)\}$. Further, let $SG_{\mathcal{G}}$ be a group sampler of some group family $\mathcal{G}$ where the groups have an order with at least one large prime factor.

   (a) Assumption $1/\mathsf{poly}(k)$-CDL$^{\sigma}$(c:u; g:h; f:lprim), i.e., the uniform complexity variant:

$\forall \mathcal{A}^\sigma \in \mathcal{UPTM}^\sigma;$
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$
$\forall G \in [SG_\mathcal{G}(1^k)];$
$\forall g \in [Sg(G)];$
$SI \leftarrow (G, g);$

$\mathbf{Prob}[\mathcal{A}^\sigma(\mathcal{C}, SI, PI_{\mathrm{DL}}{}^{publ}) \in PI_{\mathrm{DL}}{}^{sol} ::$
$\quad \sigma \xleftarrow{\mathcal{R}} \Sigma_{G,g};$
$\quad PI_{\mathrm{DL}} \leftarrow SPI_{\mathrm{DL}}(SI);$
$\quad \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
$\ ] < 1/k^{d_1}.$

(b) Same setting as above except now with a non-uniform adversary ($1/\mathsf{poly}(k)$-CDL^σ(c:n; g:h; f:lprim)):

$\forall (\mathcal{A}_i^\sigma \mid i \in \mathbb{N}) \in \mathcal{NPTM}^\sigma;$
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$
$\forall G \in [SG_\mathcal{G}(1^k)];$
$\forall g \in [Sg(G)];$
$SI \leftarrow (G, g);$

$\mathbf{Prob}[\mathcal{A}_k^\sigma(\mathcal{C}, SI, PI_{\mathrm{DL}}{}^{publ}) \in PI_{\mathrm{DL}}{}^{sol} ::$
$\quad \sigma \xleftarrow{\mathcal{R}} \Sigma_{G,g};$
$\quad PI_{\mathrm{DL}} \leftarrow SPI_{\mathrm{DL}}(SI);$
$\quad \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
$\ ] < 1/k^{d_1}.$

2. Weak decisional DH assumption variants for prime order subgroups of $\mathbb{Z}_p^*$ with varying granularity. Recall that $PI_{\mathrm{DH}} := (SI, ((x, y), (g^x, g^y), \{(g^{xy})\}))$, $PI_{DH}{}^{publ} := (g^x, g^y)$ and $PI_{DH}{}^{sol} := \{(g^{xy})\}$.

   (a) Assumption $1/\mathsf{poly}(k)$-DDH(c:u; g:h; f:$\mathbb{Z}_{p/q}^*$), i.e., with high granularity:

```
$\forall \mathcal{A} \in \mathcal{UPTM};$
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$
$\forall G \in [SG_{\mathbb{Z}^*_{p/q}}(1^k)];$
$\forall g \in [Sg(G)];$
$SI \leftarrow (G, g);$

$(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, {PI_{\mathrm{DH}/0}}^{publ}, sol_{\mathrm{DH}/c}) = b ::$
    $b \xleftarrow{\mathcal{R}} \{0,1\};$
    $PI_{\mathrm{DH}/0} \leftarrow SPI_{\mathrm{DH}}(SI);$
    $PI_{\mathrm{DH}/1} \leftarrow SPI_{\mathrm{DH}}(SI);$
    $sol_{\mathrm{DH}/c} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/b}}^{sol};$
    $\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
  $] - 1/2\,| \cdot 2)\ < 1/k^{d_1}.$
```

(b) As above except now with medium granularity ($1/\mathsf{poly}(k)$-DDH(c:u; g:m; f:$\mathbb{Z}^*_{p/q}$)):

```
$\forall \mathcal{A} \in \mathcal{UPTM};$
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$
$\forall G \in [SG_{\mathbb{Z}^*_{p/q}}(1^k)];$

$(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, {PI_{\mathrm{DH}/0}}^{publ}, sol_{\mathrm{DH}/c}) = b ::$
    $g \leftarrow Sg(G);$
    $SI \leftarrow (G, g);$
    $b \xleftarrow{\mathcal{R}} \{0,1\};$
    $PI_{\mathrm{DH}/0} \leftarrow SPI_{\mathrm{DH}}(SI);$
    $PI_{\mathrm{DH}/1} \leftarrow SPI_{\mathrm{DH}}(SI);$
    $sol_{\mathrm{DH}/c} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/b}}^{sol};$
    $\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
  $] - 1/2\,| \cdot 2)\ < 1/k^{d_1}.$
```

(c) As above except now with low granularity ($1/\mathsf{poly}(k)$-DDH(c:u; g:l; f:$\mathbb{Z}^*_{p/q}$)):

$\forall \mathcal{A} \in \mathcal{UPTM};$
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$

$(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, {PI_{\mathrm{DH}/0}}^{publ}, sol_{\mathrm{DH}/c}) = b ::$
$\quad G \leftarrow SG_{\mathbb{Z}^*_{p/q}}(1^k);$
$\quad g \leftarrow Sg(G);$
$\quad SI \leftarrow (G, g);$
$\quad b \overset{\mathcal{R}}{\leftarrow} \{0,1\};$
$\quad PI_{\mathrm{DH}/0} \leftarrow SPI_{\mathrm{DH}}(SI);$
$\quad PI_{\mathrm{DH}/1} \leftarrow SPI_{\mathrm{DH}}(SI);$
$\quad sol_{\mathrm{DH}/c} \overset{\mathcal{R}}{\leftarrow} {PI_{\mathrm{DH}/b}}^{sol};$
$\quad \mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U}$
$\ ] - 1/2\,| \cdot 2)\ < 1/k^{d_1}.$

3. Matching IE assumptions in $\mathbb{QR}^*_n$ with varying success probability. Recall that $PI_{\mathrm{IE}} := (SI, ((x), (g^x), \{(g^{x^{-1}})\}))$, ${PI_{IE}}^{publ} := (g^x)$ and ${PI_{IE}}^{sol} := \{(g^{x^{-1}})\}$.

   (a) Assumption $1/\mathsf{poly}(k)$-MIE(c:u; g:h; f:$\mathbb{QR}^*_n$), i.e., the variant with weak success probability:

   $\forall \mathcal{A} \in \mathcal{UPTM};$
   $\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$
   $\forall G \in [SG_{\mathbb{QR}^*_n}(1^k)];$
   $\forall g \in [Sg(G)];$
   $SI \leftarrow (G, g);$

   $(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, {PI_{\mathrm{IE}/0}}^{publ}, {PI_{\mathrm{IE}/1}}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::$
   $\quad b \overset{\mathcal{R}}{\leftarrow} \{0,1\};$
   $\quad PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI);$
   $\quad PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI);$
   $\quad sol_{\mathrm{IE}/0} \overset{\mathcal{R}}{\leftarrow} {PI_{\mathrm{DH}/0}}^{sol};$
   $\quad sol_{\mathrm{IE}/1} \overset{\mathcal{R}}{\leftarrow} {PI_{\mathrm{DH}/1}}^{sol};$
   $\quad \mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U}$
   $\ ] - 1/2\,| \cdot 2)\ < 1/k^{d_1}.$

   (b) Same setting as above except now with invariant success probability ϵ (ϵ-MIE(c:u; g:h; f:$\mathbb{QR}^*_n$)):

$\forall \mathcal{A} \in \mathcal{UPTM};$
$\exists k_0;\ \forall k > k_0;$
$\forall G \in [SG_{\mathbb{QR}^*_n}(1^k)];$
$\forall g \in [Sg(G)];$
$SI \leftarrow (G, g);$

$(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, {PI_{\mathrm{IE}/0}}^{publ}, {PI_{\mathrm{IE}/1}}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::$
$\quad b \overset{\mathcal{R}}{\leftarrow} \{0,1\};$
$\quad PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI);$
$\quad PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI);$
$\quad sol_{\mathrm{IE}/0} \overset{\mathcal{R}}{\leftarrow} {PI_{\mathrm{DH}/0}}^{sol};$
$\quad sol_{\mathrm{IE}/1} \overset{\mathcal{R}}{\leftarrow} {PI_{\mathrm{DH}/1}}^{sol};$
$\quad \mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U};$

$\quad] - 1/2\ | \cdot 2)\ < \epsilon.$

(c) Same setting as above except now with strong success probability ($(1-1/\mathsf{poly}(k))$-MIE(c:u; g:h; f:$\mathbb{QR}^*_n$)):

$\forall \mathcal{A} \in \mathcal{UPTM};$
$\exists d_1 > 0;\ \exists k_0;\ \forall k > k_0;$
$\forall G \in [SG_{\mathbb{QR}^*_n}(1^k)];$
$\forall g \in [Sg(G)];$
$SI \leftarrow (G, g);$

$(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, {PI_{\mathrm{IE}/0}}^{publ}, {PI_{\mathrm{IE}/1}}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::$
$\quad b \overset{\mathcal{R}}{\leftarrow} \{0,1\};$
$\quad PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI);$
$\quad PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI);$
$\quad sol_{\mathrm{IE}/0} \overset{\mathcal{R}}{\leftarrow} {PI_{\mathrm{DH}/0}}^{sol};$
$\quad sol_{\mathrm{IE}/1} \overset{\mathcal{R}}{\leftarrow} {PI_{\mathrm{DH}/1}}^{sol};$
$\quad \mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U}$
$\quad] - 1/2\ | \cdot 2)\ < (1 - 1/k^{d_1}).$

(d) Same setting as above except with no tolerated error, i.e., perfect success probability (1-MIE(c:u; g:h; f:$\mathbb{QR}^*_n$)):

$$
\begin{array}{l}
\forall \mathcal{A} \in \mathcal{UPTM};\\
\exists k_0;\ \forall k > k_0;\\
\forall G \in [SG_{\mathbb{QR}_n^*}(1^k)];\\
\forall g \in [Sg(G)];\\
SI \leftarrow (G, g);\\[1em]
(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, {PI_{\mathrm{IE}/0}}^{publ}, {PI_{\mathrm{IE}/1}}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::\\
\quad b \xleftarrow{\mathcal{R}} \{0,1\};\\
\quad PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI);\\
\quad PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI);\\
\quad sol_{\mathrm{IE}/0} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/0}}^{sol};\\
\quad sol_{\mathrm{IE}/1} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/1}}^{sol};\\
\quad \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}\\
\ ]-1/2\ | \cdot 2)\ < 1.
\end{array}
$$

To express relations among assumptions, we use the following operators where P and Q are assumptions as previously defined:

$P \implies Q$ means that if assumption P holds, so does assumption Q, i.e., P (Q) is a stronger (weaker) assumption than Q (P). Vice-versa, it also means that if there is a polynomially-bounded algorithm $\mathcal{A}_Q$ breaking assumption Q then there is also another polynomially-bounded algorithm $\mathcal{A}_P$ which breaks assumption P. Usually, this is shown in a **black-box reduction** where $\mathcal{A}_P$, or more precisely $\mathcal{A}_P^{\mathcal{A}_Q}$, breaks assumption P with oracle access to $\mathcal{A}_Q$. As a special case for invariant assumptions, we mean with ϵ-$P \implies \epsilon$-Q that it should hold that $\forall \epsilon' \in]0,1[\ \exists \epsilon'' \in]0,1[\ :\ \epsilon''\text{-}P \implies \epsilon'\text{-}Q$.

$P \iff Q$ means that $P \implies Q$ and $Q \implies P$, i.e., P and Q are assumptions of the same (polynomial) complexity.

$P \xRightarrow{\alpha' \geq f_\alpha(t,\alpha,|G|,\ldots);\ t' \leq f_t(t,\alpha,|G|,\ldots)} Q$ is used to specify the quality of the reduction, i.e., the concrete security. It means that if assumption Q can be broken in time t and with success probability α, we can break P in time t' and with success probability α' bounded by functions f_t and f_α, respectively. To measure time, we consider group operations and equality tests having unit-cost each and oracle calls having cost t. Obviously, the cost of group operations, the runtime and the success probability of the oracle, and the size of the groups are not constant but functions depending on the security parameter k, e.g., α should be written more precisely as $\alpha(k)$. However, for better readability we omit this and all asymptotic aspects in the presentation. For the identical reason, we also cautiously use the $O(\cdot)$ notation even if we slightly lose precision.

Let us illustrate this with the following result from Maurer and Wolf (1996) (for more information on this result see also page 47):

$$\epsilon\text{-CDH(c:u; g:h; f:o)} \xRightarrow{\alpha'=\alpha^3;\ t'=3t+O(\log(|G|)^2)} \epsilon\text{-CSE(c:u; g:h; f:o)}$$

This means that with three calls to an oracle breaking ϵ-CSE(c:u; g:h; f:o) and additional $O(\log(|G|)^2)$ group operations we can achieve a success probability of at least α^3 in breaking ϵ-CDH(c:u; g:h; f:o) where t and α are the runtime and the success probability of the oracle, respectively.

For simple assumptions, above is interpreted without syntactical conditions on P and Q, i.e., they may be arbitrary assumptions. If a relation refers to assumption classes, i.e., they contain some parameters which are not fully specified and contain wild cards or sets, there is the following syntactical constraint: The parameters which are not fully specified have to be equal for both assumptions P and Q. The meaning is as follows: The relation P OP Q holds for any assumption P' and Q' we can instantiate from P and Q by fixing all not fully specified parameters to any matching value with the additional condition that these values are identical for P' and Q'. To give an example,

$$*\text{-CDH}^*\text{(c:}*\text{; g:\{h,m\}; f:o)} \implies *\text{-CSE}^*\text{(c:}*\text{; g:\{h,m\}; f:o)}$$

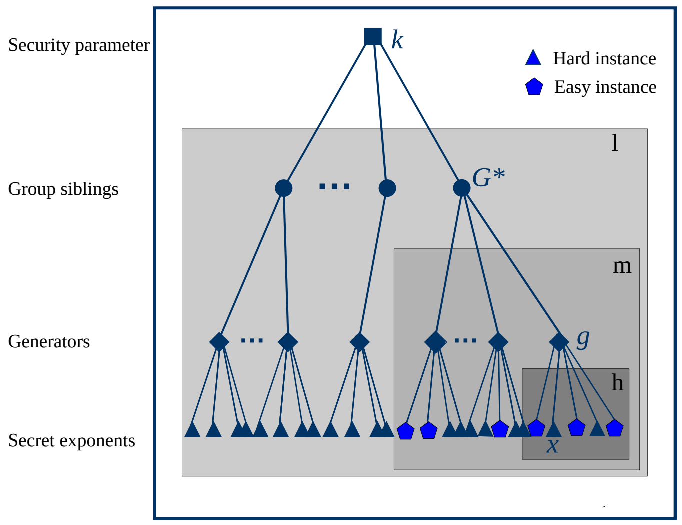
illustrates that the result from Maurer and Wolf mentioned above can be generalized — as proven later in this thesis — to high and medium granularity with arbitrary success probability, complexity and algebraic knowledge.

Furthermore, if we are referring to oracle-assumptions, i.e., assumptions where we give adversaries access to auxiliary oracles, we indicate it by listing the oracles at the end of the list in the assumption term. For example, the assumption $1/\mathsf{poly}(k)$-CDL$^\sigma$(c:u; g:h; f:lprim; $\mathcal{O}_{\text{1-CDL(c:u; g:h; f:lprim)}}$) corresponds to the first assumption statement given in the example list above except that now the adversary also gets access to an oracle breaking the 1-CDL(c:u; g:h; f:lprim) assumption.

2.4 The Impact of Granularity

In Section 2.2, we have identified several parameters to be considered when defining intractability assumptions. In this section, we will focus on granularity parameter and its impact. Before stating the actual results, let us first briefly repeat the practical relevance of granularity as alluded in the introduction. Figure 2.1 illustrates exemplarily different variants of the probability space for a given security parameter k. The areas labeled with l, m and

Figure 2.1 The impact of granularity



h represent the algebraic parameters over which low-, medium- and high-granular probability spaces are defined.[22] Assumptions with lower granularity are weaker, and are therefore more desirable in principle. However, not all cryptographic settings can rely on the weaker variants: Consider, for instance, an escrowed anonymous payment system where the bank chooses the system parameters.[23] It would not be appropriate to base the security of such a system a-priori on a low-granular assumption. This is because a cheating bank might try to choose a weak group with trapdoors (easy problem instances) to violate the anonymity of the customer. This case is shown in Figure 2.1 with a trapdoor group G^* and its corresponding easy instances. Such a strategy might be possible even if the low-granular assumption holds: The assumption would ensure that the overall number of easy problem instances is asymptotically negligible with respect to the security parameter. In Figure 2.1 weak instances (in area m) represent exemplarily a negligible portion of the instances of the low-granular space in area l. Yet, the assumption would not rule out that there are infinitely many weak groups. Therefore, there might not exist a sufficiently large k for which the bank cannot break the assumption.

In contrast, a high-granular (medium-granular) assumption does not hold in our example because as shown in the figure, the fraction of weak in-

[22] Recall that high-granular probability space is defined over the private parts (secret exponents), the medium-granular over the generators and private parts and finally the low-granular over groups, generators and the private parts.

[23] These are electronic payment systems where a third party can revoke the anonymity of the users under certain circumstances. Because of the revocation ability such systems can offer the users only computationally secure anonymity.

stances in area h (m) is not negligible. However, if a high-granular (medium-granular) assumption holds then the trapdoor groups G^* in the above example would not exist and the bank could not cheat.

Thus, which of the granularity variants is appropriate in cryptographic protocols depends on how and by whom the parameters are chosen. A priori we have to use a high-granular assumption. Yet, in the following situations we can resort to a weaker less granular assumption: The security requirements of the cryptographic system guarantee that it's in the best (and only) interest of the chooser of the system parameters to choose them properly; the system parameters are chosen by a mutually trusted third party; or the system parameters are chosen in a verifiable random process.[24] Also, at most in these cases we can reasonably assume a group family with the group order and its factorization to be hidden from the public and the adversary. As a consequence, it would seem strange to base a cryptographic system on a high-granular assumption with unknown order factorization: either the system parameters are chosen by an honest party and we could resort to a weaker assumption with lower granularity, or the knowledge of the order and its factorization has to be assumed to be known to the adversary. Furthermore, care has to be taken for DL-related high- and medium-granular assumptions in $\mathbb{Z}_p^*$ and its subgroups. Unless we further constrain the set of valid groups with (expensive) tests as outlined by Gordon (1993a), we require, for a given security parameter, considerably larger groups than for the low granular counterpart of the assumptions. As informally mentioned above, assumptions with lower granularity are weaker than assumption of higher granularity. Formally, this is stated and proven in the following theorem:

Theorem 2.1

$$*\text{-}{*}{*}^*(\text{c:}*;\text{g:h};\text{f:}*) \implies *\text{-}{*}{*}^*(\text{c:}*;\text{g:m};\text{f:}*) \implies *\text{-}{*}{*}^*(\text{c:}*;\text{g:l};\text{f:}*)$$

□

Proof. Assume we are given an adversary $\mathcal{A}$ breaking a low-granular assumption for some group and problem family, some problem type, computational complexity, arbitrary algebraic knowledge and success probability. Furthermore, we are given an input I corresponding to an assumption of high- or medium-granular but otherwise identical parameters.

For the reduction, call $\mathcal{A}$ on this input I and return the result. To see that this achieves the desired attack on the medium- or high-granular assumption, note that inputs to an adversary breaking a high- or medium-granular

[24]This can be done either through a joint generation using random coins (Cachin, Kursawe, and Shoup 2000) or using heuristics such as the one used for DSS key generation (National Institute of Standards and Technology (NIST) 2000).

assumption are also valid inputs to a low-granular adversary. Therefore, this reduction is a legitimate attacker from a runtime perspective exactly in the case where the oracle itself is a legitimate attacker. Furthermore, the probability space instances defined by a high- or medium-granular assumption always partition the probability space instances of a low-granular assumption. Therefore, it is clear that for a perfect adversary $\mathcal{A}$ the reduction breaks certainly the high- or medium-granular probability space instances which are part of the low-granular probability space instances which $\mathcal{A}$ breaks. As there are by definition of $\mathcal{A}$ infinitely many such low-granular probability space instances, it automatically follows that for the perfect case the high- and medium-granular assumption is broken, too. By a counting argument this also easily extends to the case of strong, invariant and weak adversaries, i.e., at least some of the high- or medium-granular probability space instances which are part of the low-granular probability space instances broken by $\mathcal{A}$, are broken with the necessary success probability as well.

By an identical argument, it follows that a high-granular assumption can be reduced to the corresponding medium-granular assumption. This concludes the theorem. ■

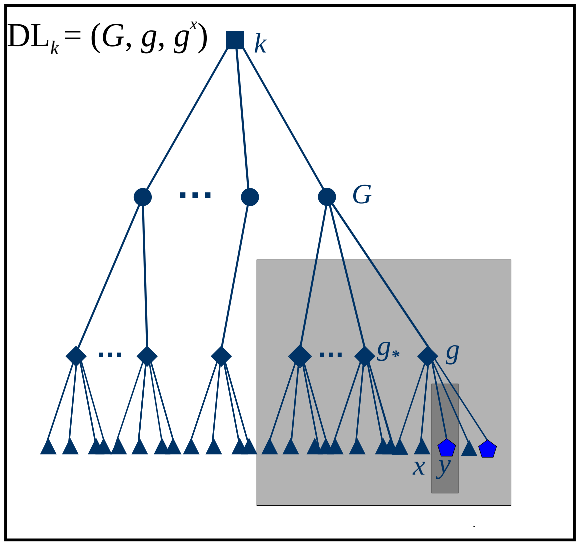
Remark 2.1. Note that the inverse of above result, a low-granular assumption implies the corresponding high-granular one, does not hold in general: There are always super-polynomially many of the higher-granular probability space instances contained in a given lower-granular instance. Therefore, there might be situations where infinitely many high-granular probability space instances — and henceforth the corresponding high-granular assumption — are broken, yet they form only a negligible subset of the enclosing lower-granular probability space instances and the low-granular assumption can still hold.

However, if for a given granularity there exists a random self-reduction (Blum and Micali 1984), then the inverse reduction exists also from that granularity to all higher granularities. As random self-reductions are known for all mentioned problem families and problem types in their medium granularity variant, this equates the medium- and high-granular cases. Unfortunately, no random self-reduction is yet known for low-granular assumptions and achieving such "full" random self-reducibility seems very difficult in general (if not impossible) in number-theoretic settings (Boneh 2000) contrary to, e.g., lattice settings used by Ajtai and Dwork (1997). ◦

2.5 Computational DH, SE and IE

Maurer and Wolf (1996) proved the equivalence between the computational SE and DH assumptions in their uniform and high-granular variant for both perfect and invariant success probabilities.

Figure 2.2 Random Self Reducibility (Medium granularity)



We briefly review their results, and show that they also hold for weak and strong success probabilities. We then extend these results to medium granularity and prove similar relations between IE and DH.

First however, we look at two important aspects: random self-reducibility and self-correction.

2.5.1 Random Self-Reduction

Informally **random self-reducibility** is a property which can be used to show that a problem is uniformly hard if it is hard at all. A problem is random self-deducible if solving *any* problem instance can be reduced to solving the problem on a *random* instance, i.e., when given an instance I of a probability space instance P, we can efficiently randomize it to a random instance $I_{\mathcal{R}}$ of P, and can efficiently recover (derandomize) the solution for I from the solution returned by an oracle call on $I_{\mathcal{R}}$.

We give an informal example to demonstrate this property for discrete logarithm. Figure 2.2 illustrates the corresponding groups G and generators g for a security parameter k. Suppose, we are given an oracle $\mathcal{O}_{DL}$ which with success probability α, computes the discrete logarithm x of randomly chosen values $b \in G$ with respect to the given generator g. We construct a probabilistic algorithm $\mathcal{A}^{\mathcal{O}_{DL}}$ to compute the discrete logarithm for any specific DL instance $((G, g), (a))$ where $a = g^y$ (shown in Figure 2.2). Further, the success probability of $\mathcal{A}^{\mathcal{O}_{DL}}$ is the same as $\mathcal{O}_{DL}$.

For this, we transform a given DL instance $a = g^y$ for a given generator $g \in G$ into a random DL instance $b = {g_*}^x$ for a random generator $g_* \in G$

as follows: Select $r \in_{\mathcal{R}} \mathbb{Z}_{|G|}$, $r_g \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$ and call $\mathcal{O}_{DL}$ with the input $((G, g_*), (b))$ where

$$g_* := g^{r_g}, \quad b := a^{r_g} g^{r r_g} = (g^y)^{r_g} g^{r r_g} = g^{r_g(y+r)} = (g^{r_g})^{(y+r)} = {g_*}^x$$

and $x{:=}y{+}r$. Note that b is a randomly and uniformly distributed value over G since g_* is a random group generator, and r is randomly and uniformly chosen value from $\mathbb{Z}_{|G|}$. When $\mathcal{O}_{DL}$ answers x, $\mathcal{A}^{\mathcal{O}_{DL}}$ outputs $y = x - r$, a unique number in $\mathbb{Z}_{|G|}$. The output y is a correct discrete logarithm of a if and only if x is a correct discrete logarithm of b with respect to g. One can verify this as follows:

$$a = (b/{g_*}^r)^{r_g^{-1}} = ({g_*}^x {g_*}^{-r})^{r_g^{-1}} = ({g_*}^{x-r})^{r_g^{-1}} = ({g_*}^{r_g^{-1}})^{x-r} = g^{x-r}.$$

Thus, $\mathcal{A}^{\mathcal{O}_{DL}}$ is successful if and only if $\mathcal{O}_{DL}$ is successful. Since there is a single call to $\mathcal{O}_{DL}$ and the input distribution to $\mathcal{O}_{DL}$ is correct, the success probability of $\mathcal{A}^{\mathcal{O}_{DL}}$ is also α. If $\mathcal{O}_{DL}$ is an efficient algorithm, so is $\mathcal{A}^{\mathcal{O}_{DL}}$.[25]

The above example illustrates random self-reducibility for DL over the medium-granular probability space (see Figure 2.2). For the random self-reducibility over high-granular probability space we always set $r_g = 1$.

2.5.2 Self-Correction

In the following sections, we are mostly concerned with faulty oracles, i.e., oracles which answer with a certain success probability to the legal inputs, i.e., inputs with correct distribution over the oracle's input domain. If an oracle has a small but not negligible success probability, one is interested in constructing an efficient algorithm which improves this success probability such that the answers to the legal inputs are almost certainly correct. In other words, one is interested in performing **self-correction** on the faulty oracle.

In our considerations we need to self-correct the faulty CDH oracle to determine the success probability of certain reductions which appear in later sections. Suppose, we are given a faulty CDH oracle $\mathcal{O}_{CDH}$ which on input $((G, g), (g^x, g^y))$ outputs g^{xy} with a not negligible probability α. Then we can construct an efficient algorithm for CDH which outputs the correct answer almost certainly for all legal inputs. One may ask, why not running such an oracle for $O(1/\alpha)$ times until we get a correct answer! However, this is of no help, since in general we cannot determine (decide) whether the output of the oracle is the correct Diffie-Hellman solution or not – this would mean solving Decisional Diffie-Hellman (DDH) Problem which is assumed to be hard in the underlying group.

[25] Note that we assume, the oracle has polynomial running time on all inputs (see also Section 2.1.3).

Thus, other (more complicated) approaches have been taken to construct self-correctors for computational problems such as CDH.

Maurer and Wolf (1996) and Shoup (1997) give different constructions for CDH self-correctors. For our considerations we will use the result from Shoup 1997 which is formulated in the following lemma.

Lemma 2.1 (Shoup 1997) *Given a CDH oracle with success probability α, one can construct a probabilistic algorithm for CDH which, for a given $0 < \beta < 1$, answers correctly to all inputs with probability at least $\alpha' = 1 - \beta$ making $O(\frac{\log(1/\beta)}{\alpha})$ queries to the faulty oracle and performing additional $O(\frac{\log(1/\beta)}{\alpha} \log|G| + (\log|G|)^2)$ group operations.* $\square$

Note that Lemma 2.1 does not consider the success probabilities in the asymptotic framework as we introduced in Section 2.1.2. Thus, for our considerations we suitably adjust this result when self-correcting a weak or invariant oracle to a strong oracle.[26] This is summarized in the following corollary:

Corollary 2.1

$$\{(1-1/\mathsf{poly}(k))\}\text{-CDH(c:}*\text{; g:}\{\text{h,m }\}\text{; f:o)} \xRightarrow{\alpha' \geq 1-1/2^k;\ t'=O(\frac{tk}{\alpha})+O(\frac{k\log|G|}{\alpha}+(\log|G|)^2)} \{\epsilon, 1/\mathsf{poly}(k)\}\text{-CDH(c:}*\text{; g:}\{\text{h,m }\}\text{; f:o)}$$

$\square$

Proof. We give the proof for weak oracles, and the proof for the invariant oracle immediately follows.

Assume, we are given a CDH oracle with weak success probability $\alpha(k)$. In our framework for success probabilities it is reasonable to self-correct this oracle to a strong oracle, i.e., an oracle with success probability $\alpha'(k)$ where $1 - \alpha'(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

By the straight forward (and naive) application of Lemma, 2.1 one may set $\beta(k) := 1/2^k$ (k security parameter) and self-correct oracle's success probability to $\alpha'(k) \geq 1 - 1/2^k$ implying $1 - \alpha'(k) \leq 1/2^k$. Since $1/2^k$ is asymptotically smaller than the inverse of any polynomial, we can write $1/2^k <_\infty 1/\mathsf{poly}(k)$. It follows that $1 - \alpha'(k) <_\infty 1/\mathsf{poly}(k)$. According to Lemma 2.1 the self-correction requires $O(\frac{k}{\alpha(k)})$ calls to the weak oracle and $O(\frac{k\log|G|}{\alpha(k)} + (\log|G|)^2)$ group operations where we used $\log(1/\beta(k)) = \log(2^k) = k$.

However, this approach is not conform to our framework of success probabilities and does not work directly. The reason is that in general the above

[26] Recall that in the asymptotic notion the success probabilities are in fact functions in the security parameter and for weak and invariant oracles we have $\alpha(k) \not<_\infty 1/\mathsf{poly}(k)$ and $\alpha(k) \not<_\infty \epsilon$ (see Section 2.2)

self-correction may not provide us with a polynomial time algorithm. It is guaranteed to be polynomial only for those values of k (infinitely many k_i by definition) where the success probability $\alpha(k)$ of the weak oracle can be lower bounded by the inverse of some polynomial $p(\cdot)$, but not necessarily for other values of k.[27] To handle this problem, one can define a family of algorithms indexed by a polynomial $p_j(\cdot)$ which runs the self-correction with $p_j(k)$ rounds (oracle calls). Thus, all members of this family have the run time $O\big(p_j(k)\big) + O\big(p_j(k)\log(|G|) + (\log|G|)^2\big)$, and therefore are polynomial. Moreover, there are members of this family which satisfy the condition for strong success probability. These are exactly the members for which $kp(\cdot) <_\infty p_j(\cdot)$ holds. In particular, this holds for the same k values for which the given weak success probability holds.

However, this is an existential argument and not constructive, as in general, neither the function $\alpha(k)$ or the k_i values are known beforehand nor they can be approximated by querying the oracle in polynomial time.

Hence, in our framework it suffices to self-correct $\mathcal{O}_{CDH}$ such that $\alpha'(k) \not<_\infty 1 - 1/2^k$ holds. This implies $1 - \alpha'(k) \not>_\infty 1/2^k$, and since $1/2^k <_\infty 1/\mathsf{poly}(k)$, it follows $1 - \alpha'(k) \not\geq_\infty 1/\mathsf{poly}(k)$ (i.e., $\alpha'(k)$ is strong.) This completes the proof. ■

Remark 2.2. In the proof of his self-corrector Shoup (1997) assumes that the group order is known. However, by a closer inspection of the proof and the deployment of the techniques used in Remark 2.16, we can drop this requirement. Thus, Corollary 2.1 also holds without requiring the knowledge of the group order. ○

2.5.3 CSE versus CDH

2.5.3.1 High Granular

We start with the result of Maurer and Wolf (1996) on the equivalence between the computational SE and DH assumptions in their uniform and high-granular variant for perfect and invariant success probabilities. This is formulated in our convention in the following theorem.

Theorem 2.2 (Maurer and Wolf 1996)

$$\epsilon\text{-CSE(c:u; g:h; f:o)} \xRightarrow{\alpha'=\alpha;\ t'=t+O(\log|G|)} \epsilon\text{-CDH(c:u; g:h; f:o)}$$

$$\epsilon\text{-CSE(c:u; g:h; f:o)} \xLeftarrow{\alpha'=\alpha^3;\ t'=3t+O(\log|G|)} \epsilon\text{-CDH(c:u; g:h; f:o)}$$

□

[27]Note that we might have $\alpha(k) = 0$ for infinitely many k values, and since the self-correction costs are proportional to $1/\alpha(k)$, this would lead to exponentially high number of oracle calls and group operations.

Proof. Let $0 < \epsilon_1 < 1$, $0 < \epsilon_2 < 1$ be arbitrary constants. Then the following statements hold:

(a) Given a CDH oracle $\mathcal{O}_{CDH}$ which breaks ϵ-CDH(c:u; g:h; f:o) with success probability $\alpha_{CDH}(k) \not<_\infty \epsilon_1$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CDH}}$ which breaks ϵ-CSE(c:u; g:h; f:o) with success probability $\alpha_{CSE}(k) \not<_\infty \epsilon_1$, using a single call to $\mathcal{O}_{CDH}$ and $O(\log |G|)$ group operations.

(b) Given a CSE oracle $\mathcal{O}_{CSE}$ which breaks ϵ-CSE(c:u; g:h; f:o) with success probability $\alpha_{CSE}(k) \not<_\infty \epsilon_2$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CSE}}$ which breaks ϵ-CDH(c:u; g:h; f:o) with success probability $\alpha_{CDH}(k) \not<_\infty {\epsilon_2}^3$, using 3 calls to $\mathcal{O}_{CSE}$ and $O(\log |G|)$ group operations.

From these reductions the theorem immediately follows. Above reductions are achieved as follows:

Case (a) is quite straightforward as the problem instances of SE are a proper subset of the problem instances of DH and the answer can be retrieved in one call to the oracle. In the case of perfect CDH oracle (perfect success probability) the oracle returns g^{x^2} on the input $((G, g), (g^x, g^x))$. However, in the case of invariant oracle (faulty oracle) care has to be taken that the inputs to the CDH oracle are uniformly distributed over oracle's input domain.[28] This can easily be achieved by randomizing a given tuple $((G, g), (g^x, g^x))$ to a random CDH tuple $((G, g), (g^{x'}, g^{y'}))$, i.e., randomly self-reducing the problem as follows (see also Section 2.5.1): Choose $r_x, r_y \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ and set $x' := x + r_x$, and $y' := x + r_y$. The elements $g^{x'}, g^{y'}$ are randomly and uniformly distributed over G, since due to the randomization, x', y' are randomly and uniformly spread over $\mathbb{Z}_{|G|}$.[29] Note that in the high-granular case an $SI = (G, g)$ fixes a probability space instance (PSI).

Using oracle's answer we can determine the desired result as follows:

$$g^{x^2} = \frac{\mathcal{O}_{CDH}(g^{x'}, g^{y'})}{g^{r_x x + r_y x + r_x r_y}} = \frac{g^{x'y'}}{g^{r_x x + r_y x + r_x r_y}}.$$

Success probability: There is a single oracle call, and thus, for the success probability of $\mathcal{A}^{\mathcal{O}_{CDH}}$ we have $\alpha_{CSE}(k) = \alpha_{CDH}(k)$. Since $\alpha_{CDH}(k) \not<_\infty \epsilon_1$ it follows $\alpha_{CSE}(k) \not<_\infty \epsilon_1$.

Efficiency: There is only one oracle call and to solve the DSE instance we need to compute g^{r_x}, g^{r_y}, $(g^x)^{r_x}, (g^x)^{r_y}, g^{r_x r_y}$ and $(g^{r_x x + r_y x + r_x r_y})^{-1}$. For

[28]Note that the success probability of a faulty oracle holds for randomly and uniformly chosen inputs from the oracle's input domain.

[29]that is, $\forall (x', y') \in \mathbb{Z}^2_{|G|}$ and $\forall (x, y) \in \mathbb{Z}^2_{|G|}$ there exists exactly one pair $(r_x, r_y) \in \mathbb{Z}^2_{|G|}$ such that the equations $x' = x + r_x$ and $y' = x + r_y$ hold.

exponentiations we can use, e.g., the square and multiply method requiring $O(\log |G|)$ group operations. If we assume that the group order is known (denoted by the place holder "f:o"), we can efficiently compute the inverse of the group elements using $O(\log |G|)$ group operations.

Case (b) is slightly more involved. The key observation is that

$$g^{(x+y)^2} = g^{2xy} g^{x^2} g^{y^2}.$$

This implies

$$(g^{xy})^2 = g^{2xy} = g^{(x+y)^2} (g^{x^2})^{-1} (g^{y^2})^{-1} = g^{(x+y)^2 - x^2 - y^2}.$$

Therefore, we can solve CDH with three oracle calls (one for each of $g^{(x+y)^2}$, g^{x^2} and g^{y^2}), the computation of inverses of g^{x^2} and g^{y^2}, and the square root of $g^{2xy} = (g^{xy})^2$.

As before, for the faulty oracle we have to uniformly spread the given input over oracle's input domain using randomization. Furthermore, we have to make sure now that all oracle calls are (statistically) independent to be able to make concrete statements on the success probability of $\mathcal{A}^{\mathcal{O}_{CSE}}$. Both we achieve with independent blinding factors $r_i \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ and computing g^{u^2} as $\mathcal{O}_{CSE}(g^{(u+r_i)})/(g^u)^{2r_i} g^{r_i^2}$. Note that $g^{(u+r_i)}$ are randomly and uniformly distributed group elements, since $u + r_i$ are randomly and uniformly distributed over $\mathbb{Z}_{|G|}$.

Success probability: There are 3 independent calls to the CSE oracle, and thus, the success probability of $\mathcal{A}^{\mathcal{O}_{CSE}}$ is $\alpha_{CDH}(k) = \alpha_{CSE}(k)^3$. Since $\alpha_{CSE}(k) \not<_\infty \epsilon_2$ it follows $\alpha_{CDH}(k) \not<_\infty {\epsilon_2}^3$.

Efficiency: Assuming the group order $|G|$ is known, one can efficiently compute the inverse and square roots of elements in G. Computing the inverse of elements requires $O(\log |G|)$ group operations. For computing square roots there are two possible cases:

(i) The group order $|G|$ is odd, i.e., $\gcd(|G|, 2) = 1$. In this case, we can use the following general result: For d with $\gcd(|G|, d) = 1$ and $a \in G$ the equation $x^d = a$ has the unique solution $x = a^c$ where $dc \equiv 1 \bmod |G|$. The required number of group operations for computing this is in the order of $O(\log |G|)$.

Thus, one can determine the (unique) square root of $a = (g^{xy})^2$ by computing $c \equiv 2^{-1} \bmod |G|$ and $g^{xy} = a^c$. As mentioned above, the total cost is in the order of $O(\log |G|)$ group operations.

(ii) The group order $|G|$ is even. Thus, there exist two square roots. To compute these roots, one can apply the methods from Wolf 1999

(Lemma 11.4 and Theorem 11.5) where the (maximal) cost is in the order of $O(\log |G|)$ group operations.

The two square roots of a are g^{xy} and $g^{xy+|G|/2}$. To find out which one is the correct square root of a, we proceed as follows: Assume, $|G| = 2^e s$ where $\gcd(s, 2) = 1$. Apply the Pohlig-Hellman algorithm to compute x, y and $xy \bmod 2^e$. This requires $O(\log |G|)$ group operations.[30] Since $2^e \not| \, |G|/2$ we have $xy \not\equiv xy + |G|/2 \bmod 2^e$, and so we can determine the correct root g^{xy} by computing the discrete logarithm of one of the roots $\bmod 2^e$.

The costs per oracle call are in the order of $O(\log |G|)$ group operations for computing the square roots, and $O(\log |G|)$ group operations for exponentiation and computing the inverses. Hence, for 3 oracle calls the total costs can be expressed by $O(\log |G|)$ group operations.

■

Next, we extend this result to all other variants related to success probability (weak, strong) and adversary's computational complexity (non-uniform) as stated in the following Theorem:

Theorem 2.3

$$*\text{-CSE(c:}*\text{; g:h; f:o)} \quad \xrightarrow{\alpha'=\alpha;\ t'=t+O(\log |G|)} \quad *\text{-CDH(c:}*\text{; g:h; f:o)}$$

$$*\text{-CSE(c:}*\text{; g:h; f:o)} \quad \xleftarrow{\alpha'=\alpha^3;\ t'=3t+O(\log(|G|))} \quad *\text{-CDH(c:}*\text{; g:h; f:o)}$$

□

Proof. We consider only the variants related to the weak and strong success probabilities since the other variants (perfect and invariant) are handled by Theorem 2.2.

Weak oracles ($\alpha_{CDH}(k) \not<_\infty 1/\mathsf{poly}(k)$): The resulting success probability in both reductions is also weak: In the first reduction, we have $\alpha_{CSE}(k) = \alpha_{CDH}(k)$ implying $\alpha_{CSE}(k) \not<_\infty 1/\mathsf{poly}(k)$. In the second reduction, we have $\alpha_{CDH}(k) = \alpha_{CSE}(k)^3$ which is a power of a not negligible function resulting in a not negligible function. It follows $\alpha_{CDH}(k) \not<_\infty 1/\mathsf{poly}(k)$.

Strong oracles ($1 - \alpha_{CDH}(k) \not\geq_\infty 1/\mathsf{poly}(k)$): The resulting success probability in both cases is also strong: We use the result of Lemma 2.2 stating that

[30]Let $\prod p_i^{e_i}$ be the prime factorization of the group order $|G|$. Then using the so-called Pohlig-Hellman decomposition (Pohlig and Hellman 1978) combined with baby-step giant-step one can compute the discrete logarithm x of $b = g^x$ in G by $O(\sum e_i(\log |G| + \sqrt{p_i} \log p_i))$ group operations if the memory space for storing $\lceil \sqrt{p_i} \rceil$ group elements is available (see also Wolf (1999)) Here, we want to compute the discrete logarithm modulo 2^e, i.e., for $p_i = 2$ and $e_i = e$. For this, one requires $O(e(\log |G| + \sqrt{2} \log 2)) = O(\log |G|)$ group operations.

if a strong oracle is called polynomially (and independently) many times, the resulting success probability is also strong: In the first reduction there is a single oracle call, and in the second case there is a constant number of oracle calls (3 calls). Hence, the error probability of the algorithm is in both cases not non-negligible, i.e., $1 - \alpha_{CDH}(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

Efficiency: is the same as in Theorem 2.2. ■

The following lemma formulates the fact that if a strong oracle is called polynomially (and independently) many times, the resulting success probability is also strong.

Lemma 2.2 *Let $k \in \mathbb{N}$, $\alpha(k)$ be a function $\mathbb{N} \rightarrow [0,1]$ and $b, d > 0$ some real constants. Then the following holds:*

$$1 - \alpha(k) \not\geq_\infty 1/\mathsf{poly}(k) \implies 1 - \alpha(k)^{bk^d} \not\geq_\infty 1/\mathsf{poly}(k).$$

□

Proof. Due to the definition we have

$$1 - \alpha(k) \not\geq_\infty 1/\mathsf{poly}(k) \implies \forall c > 0\ \forall k_0\ \exists k_1 > k_0 : \alpha(k_1) > 1 - \frac{1}{k_1^c}.$$

It follows

$$\forall b > 0\ \forall d > 0\ \forall c > 0\ \forall k_0\ \exists k_1 > k_0 : \alpha(k_1)^{bk_1^d} > (1 - \frac{1}{k_1^c})^{bk_1^d}.$$

Now, for any $b > 0$ and any $d' > d > 0$ there exists $k_0' \in \mathbb{N}$ such that for all $k > k_0'$ the relation $1 \leq bk^d \leq k^{d'}$ holds. It follows

$$\forall b > 0\ \ \forall d > 0\ \forall d' > d > 0\ \forall c > 0\ \forall k_0 > k_0'\ \exists k_1 > k_0 : \alpha(k_1)^{bk_1^d} > (1 - \frac{1}{k_1^c})^{k_1^{d'}}.$$

According to Lemma 2.3 below, for $k \in \mathbb{N}$ and $c \geq d'$ the following holds: $(1 - \frac{1}{k^c})^{k^{d'}} \geq 1 - \frac{1}{k^{c-d'}}$. Since c is arbitrary and since we can write $c' := c - d' > 0$, it follows:

$$\forall b > 0\ \forall d > 0\ \forall c' > 0\ \forall k_0'\ \exists k_1 > k_0' : \alpha(k_1)^{bk_1^d} > 1 - \frac{1}{{k_1}^{c'}}.$$

This implies $1 - \alpha(k)^{bk^d} \not\geq_\infty 1/\mathsf{poly}(k)$ and the proof is completed. ■

The next lemma provides us with a useful lower bound which we apply in some proofs (as in the proof of Lemma 2.2).

Lemma 2.3 *Let $k \in \mathbb{N}$. Then for all real constants $d' > 0$ and $c > 0$ with $c > d'$ the following holds:*

$$(1 - \frac{1}{k^c})^{k^{d'}} \geq 1 - \frac{k^{d'}}{k^c} = 1 - \frac{1}{k^{c-d'}}.$$

□

Proof. First, we stress that for $a \in \mathbb{R}, a > -1$ and $n \in \mathbb{N}$, one can apply the Bernoulli inequality $(1 + a)^n \geq 1 + na$, and the claim follows immediately (set $a := -\frac{1}{k^c}$ and $n := k^{d'}$.)

We prove the claim for $n := k^{d'} \in \mathbb{R}$: For $k = 1$ this relation obviously holds. One way to see that it also holds for $k > 1$ is as follows: Consider the expressions $k^{d'} \ln(1 - 1/k^c)$ and $\ln(1 - 1/k^{c-d'})$. We expand them using

$$\ln(1 - x) = -[x + x^2/2 + x^3/3 + \cdots + x^n/n + \cdots]$$

for $-1 \leq x < 1$.

The expansion of the first expression is

$$\begin{aligned} k^{d'} \ln(1 - \frac{1}{k^c}) &= -k^{d'}[\frac{1}{k^c} + \frac{1}{2k^{2c}} + \frac{1}{3k^{3c}} + \cdots \frac{1}{nk^{nc}} + \cdots] \\ &= -[\frac{1}{k^{c-d'}} + \frac{1}{2k^{2c-d'}} + \frac{1}{3k^{3c-d'}} + \cdots + \frac{1}{nk^{nc-d'}} + \cdots] \end{aligned}$$

where $x := 1/k^c < 1$, i.e., $k^c > 1$.

Expanding the other expression we obtain

$$\ln(1 - \frac{1}{k^{c-d'}}) = -[\frac{1}{k^{c-d'}} + \frac{1}{2k^{2(c-d')}} + \frac{1}{3k^{3(c-d')}} + \cdots + \frac{1}{nk^{n(c-d')}} + \cdots].$$

where $x := 1/k^{c-d'} < 1$, i.e., $k^{d'} < k^c$.

Next, we compute the difference between these expansions:

$$\begin{aligned} \Delta(k) &:= k^{d'} \ln(1 - \frac{1}{k^c}) - \ln(1 - \frac{1}{k^{c-d'}}) \\ &= [\underbrace{(-\frac{1}{2k^{2c-d'}} + \frac{1}{2k^{2(c-d')}})}_{\delta_2} + \underbrace{(-\frac{1}{3k^{3c-d'}} + \frac{1}{3k^{3(c-d')}})}_{\delta_3} \\ &\qquad + \cdots + \underbrace{(-\frac{1}{nk^{nc-d'}} + \frac{1}{nk^{n(c-d')}})}_{\delta_n} + \cdots]. \end{aligned}$$

For $k > 1, c > d' > 0$ each difference term δ_n is positive, i.e.,

$$\delta_n = (-\frac{1}{nk^{nc-d'}} + \frac{1}{nk^{n(c-d')}}) = \frac{1}{nk^{nc-d'}}(-1 + (k^{d'})^{n-1}) > 0,$$

and we can conclude $\Delta > 0$. Thus, for $k > 1$ and $c > d'$ we can write

$$k^{d'} \ln(1 - \frac{1}{k^c}) > \ln(1 - \frac{1}{k^{c-d'}}),$$

and by applying the exponential function we obtain

$$e^{k^{d'} \ln(1-1/k^c)} = (1 - \frac{1}{k^c})^{k^{d'}} > e^{\ln(1-1/k^{c-d'})} = 1 - \frac{1}{k^{c-d'}}.$$

and this completes the proof. ■

2.5.3.2 Medium Granular

Until now we have proved the equivalence between CDH and CSE for their high-granular variants. The next theorem shows that this relation also holds for medium granularity.

Theorem 2.4

$$*\text{-CSE}(\text{c:}*; \text{g:m}; \text{f:o}) \xRightarrow{\alpha'=\alpha;\ t'=t+O(\log|G|)} *\text{-CDH}(\text{c:}*; \text{g:m}; \text{f:o})$$
$$*\text{-CSE}(\text{c:}*; \text{g:m}; \text{f:o}) \xLeftarrow{\alpha'=\alpha^3;\ t'=3t+O(\log|G|)} *\text{-CDH}(\text{c:}*; \text{g:m}; \text{f:o})$$

□

Proof. The proof idea of Theorem 2.3 can also be applied here. The only thing we have to show is that the necessary randomization in the reduction steps can be extended to the medium granularity variants of CDH and CSE. Note that for the medium-granular probability space a group G fixes a probability space instance (PSI).

CDH: We transform a given CDH input tuple $((G, g), (g^x, g^y))$ for a given generator $g \in G$ into a random CDH input tuple $((G, g_*), ({g_*}^{x'}, {g_*}^{y'}))$ for a random generator $g_* \in G$ as follows:

1. We choose $r_g \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}, r_x, r_y \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ and set $g_* := g^{r_g}$, $x' := x + r_x$, and $y' := y + r_y$.
2. We compute the public part of the input to the CDH oracle as

$$(g^x)^{r_g} g^{r_g r_x} = g^{r_g(x+r_x)} = (g^{r_g})^{(x+r_x)} = {g_*}^{x'}$$

and

$$(g^y)^{r_g} g^{r_g r_y} = g^{r_g(y+r_y)} = (g^{r_g})^{(y+r_y)} = {g_*}^{y'}.$$

The tuple $((G, g_*), (g_*, {g_*}^{x'}, {g_*}^{y'}))$ has the correct distribution for CDH oracle. This is because (i) g_* is a random group generator, and (ii) ${g_*}^{x'}, {g_*}^{y'}$ are randomly and uniformly distributed elements of G since, due to the randomization, x', y' are randomly and uniformly spread over $\mathbb{Z}_{|G|}$.[31]

3. We unblind the result of the CDH oracle as

$$\begin{aligned}({g_*}^{x'y'})^{r_g^{-1}}/((g^x)^{r_y}(g^y)^{r_x}g^{r_xr_y}) &= (g^{x'y'})^{r_gr_g^{-1}}/((g^x)^{r_y}(g^y)^{r_x}g^{r_xr_y}) \\ &= g^{(xy+xr_y+yr_x+r_xr_y)}/g^{(xr_y+yr_x+r_xr_y)} \\ &= g^{xy}.\end{aligned}$$

CSE: We transform a given CSE input $((G,g),(g^x))$ for a given generator $g \in G$ into a random CSE input $((G,g_*),({g_*}^{x'}))$ for a random generator $g_* \in G$ as follows:

1. We choose $r_g \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}, r_x \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ and set $g_* := g^{r_g}$ and $x' := x + r_x$.
2. We compute the public part of the input to the CSE oracle as

$$(g^x)^{r_g} g^{r_g r_x} = g^{r_g(x+r_x)} = (g^{r_g})^{(x+r_x)} = {g_*}^{x'}.$$

Similar to the above case, the tuple $((G,g_*),(g_*,{g_*}^{x'}))$ has the correct input distribution for the DSE oracle. This is because (i) g_* is a random group generator, and (ii) ${g_*}^{x'}$ is a random element of G since, due to the randomization, x' is randomly and uniformly spread over $\mathbb{Z}_{|G|}$.

3. We unblind the result of the CSE oracle as

$$\begin{aligned}({g_*}^{x'^2})^{r_g^{-1}}/((g^x)^{2r_x}g^{{r_x}^2}) &= (g^{x'^2})^{r_gr_g^{-1}}/((g^x)^{2r_x}g^{{r_x}^2}) \\ &= g^{(x^2+2xr_x+{r_x}^2)}/g^{(2xr_x+{r_x}^2)} \\ &= g^{x^2}.\end{aligned}$$

The rest of the proof remains the same as the proof of Theorems 2.2 and 2.3. ■

Remark 2.3. Reduction proofs of a certain granularity can in general be easily applied to the lower granularity variants of the corresponding assumptions. A sufficient condition is that all involved randomizations extend to the wider probability space associated with the lower granularity parameter. In all the mentioned problem families the random self-reducibility exists for medium granularity and we can transform proofs from a high-granular variant to the corresponding medium-granular variant. However, it does not seem to extend to low-granular variants, since this would require to randomize not only over the public part of the problem instance PI and the generator g but also over the groups G with the same associated security parameter k; this seems impossible to do in the general case and is easily overlooked and can lead to wrong conclusions, e.g., the random self-reducibility as stated by Boneh (1998) doesn't hold as the assumptions are (implicitly) given in their low-granular form. ○

[31] that is $\forall(x',y') \in \mathbb{Z}^2_{|G|}, \forall(x,y) \in \mathbb{Z}^2_{|G|}$ there exists exactly one pair $(r_x, r_y) \in \mathbb{Z}^2_{|G|}$ such that the equations $x' = x + r_x$, and $y' = y + r_y$ hold.

2.5.4 CDH versus CIE

2.5.4.1 High Granular

In the following, we prove that similar relations as between CDH and CSE also exist between CDH and CIE. As before, we show equivalence between the high-granular CDH and CIE assumptions: In Lemma 2.4 we prove high-granular reduction from CIE to CDH assumption for their different variants with respect to success probability. However, for weak and invariant CDH oracle the reduction does not work directly, and we need to self-correct the CDH oracle first.

For the converse reduction (i.e., from CDH to CIE), we first reduce CSE to CIE (Lemma 2.6), and then apply Theorem 2.3. Finally, we prove that the same relations hold also for the medium-granular versions, however, we can achieve them much more efficiently.

Lemma 2.4

$\{1,\ (1-1/\mathsf{poly}(k))\}$-CIE(c:$*$; g:h; f:fct)

$$\xRightarrow{\alpha'=\alpha^{O(\log|G|)};\ t'=O(t\log|G|)+O((\log|G|)^2)}$$

$\{1,\ (1-1/\mathsf{poly}(k))\}$-CDH(c:$*$; g:h; f:fct);

$\{(1-1/\mathsf{poly}(k))\}$-CIE(c:$*$; g:h; f:fct)

$$\xRightarrow{\alpha'\geq 1-1/2^k;\ t'=O(tk/\alpha+t\log|G|)+O(k\log|G|/\alpha+(\log|G|)^2)}$$

$\{\epsilon,1/\mathsf{poly}(k)\}$-CDH(c:$*$; g:h; f:fct)

□

Proof. The following statements hold:

(a) Given a CDH oracle $\mathcal{O}_{CDH}$ which breaks $\{1,\ (1-1/\mathsf{poly}(k))\}$-CDH(c:$*$; g:h; f:fct) with success probability $\alpha_{CDH}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CDH}}$ which breaks $\{1,\ (1-1/\mathsf{poly}(k))\}$-CIE(c:$*$; g:h; f:fct) with success probability $\alpha_{CIE}(k) = \alpha_{CDH}(k)^{O(\log|G|)}$, using $O(\log|G|)$ oracle calls and $O((\log|G|)^2)$ group operations.

(b) Given a CDH oracle $\mathcal{O}_{CDH}$ which breaks $\{\epsilon,1/\mathsf{poly}(k)\}$-CDH(c:$*$; g:h; f:fct) with success probability $\alpha_{CDH}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CDH}}$ which breaks $\{(1-1/\mathsf{poly}(k))\}$-CIE(c:$*$; g:h; f:fct) with success probability $\alpha_{CIE}(k) \not<_\infty 1-1/2^k$, using $O(k/\alpha_{CDH}(k)+\log|G|)$ oracle calls and $O(k\log|G|/\alpha_{CDH}(k)+(\log|G|)^2)$ group operations.

Case (a): Given the CDH input tuple $((G,g),(g^x))$, compute $g^{x^{-1}} = g^{x^{\varphi(|G|)-1}}$. This can be done, e.g., by applying the square and multiply method which requires $O(\log|G|)$ calls to $\mathcal{O}_{CDH}$. Note that for this, $\varphi(|G|)$

and consequently the factorization of $|G|$ must be known (This fact is indicated by the place holder f:fct in the assumption.) Further, note that each time $\mathcal{O}_{CDH}$ is called its inputs component g^u must be randomized to obtain oracle calls with properly distributed and statistically independent inputs.[32]

Success probability: Since there are $O(\log |G|)$ independent oracle calls the resulting success probability is $\alpha_{CIE} = (\alpha_{CDH}(k))^{O(\log |G|)}$. Depending on $\alpha_{CDH}(k)$ we have the following cases:

Perfect oracle ($\alpha_{CDH}(k) \not<_\infty 1$): Clearly, the resulting success probability is also perfect, i.e., $\alpha_{CIE}(k) \not<_\infty 1$.

Strong oracle ($1-\alpha_{CDH}(k) \not\geq_\infty 1/\mathsf{poly}(k)$): The resulting success probability is also strong: Set $f(|G|) := O(\log |G|)$. It follows $f(|G|) \leq b \log |G|$ for a constant $b > 0$. As discussed in Section 2.1.7, we can assume that the group order can be bounded in the security parameter, i.e., $|G| \leq 2^{k^d}$ for some $d > 0$. It follows $\log |G| \leq k^d$ and we can write

$$\alpha_{CIE}(k) = \alpha_{CDH}(k)^{f(|G|)} \geq \alpha_{CDH}(k)^{bk^d}.$$

According to Lemma 2.2, a polynomial power of a strong success probability is itself strong, i.e., $1 - \alpha_{CDH}(k)^{bk^d} \not\geq_\infty 1/\mathsf{poly}(k)$ and thus, it follows $1 - \alpha_{CIE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

Efficiency: There are $O(\log |G|)$ oracle calls, and per oracle call $O(\log |G|)$ group operations are required for exponentiations and computing inverses. This makes the total cost of $O((\log |G|)^2)$ group operations.

Case (b): The proof is similar to the case (a), except that for the weak and invariant CDH oracle the resulting success probability $\alpha_{CIE}(k)$ cannot be polynomially bounded, and the above reduction does not work directly. The success probability of $\mathcal{O}_{CDH}$ has to be improved first by means of self-correction (see Section 2.5.2), a task expensive both in terms of oracle calls and group operations.

Success probability: As mentioned above, we first self-correct the success probability of the invariant (weak) CDH oracle to strong success probability. This is done by applying Corollary 2.1. Thus, we have $1 - \alpha_{CDH}(k) \not\geq_\infty 1/\mathsf{poly}(k)$. Then it follows from Lemma 2.2 that $\alpha_{CDH}(k)^{bk^d}$ is strong, i.e.,

[32] This is done as follows: Due to square and multiply method the input tuple to $\mathcal{O}_{CDH}$ at given step is either of the form (g^{x^a}, g^x) (multiplication) or of the form (g^{x^a}, g^{x^a}) (squaring) for some a. The inputs are randomized by choosing $r, s \in_\mathcal{R} \mathbb{Z}_{|G|}$ and inputing the tuple (g^{x^a+r}, g^{x+s}) or (g^{x^a+r}, g^{x^a+s}) to $\mathcal{O}_{CDH}$. The desired outputs are then computed as $g^{x^{a+1}} = \frac{g^{(x^a+r)(x+s)}}{g^{sx^a+rx+rs}}$ or $g^{x^{2a}} = \frac{g^{(x^a+r)(x^a+s)}}{g^{(r+s)x^a+rs}}$.

we have $1 - \alpha_{CDH}(k)^{bk^d} \not\geq_\infty 1/\mathsf{poly}(k)$. Since $\alpha_{CIE}(k) \geq \alpha_{CDH}(k)^{bk^d}$ it follows $1 - \alpha_{CIE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

Efficiency: Due to Corollary 2.1 the additional costs for self-correcting are $O(k/\alpha_{CDH}(k))$ oracle calls and $O(k \log |G|/\alpha_{CDH}(k) + (\log |G|)^2)$ group operations. Thus, the total costs are: $O(k/\alpha_{CDH}(k) + \log |G|)$ oracle calls and $O(k \log |G|/\alpha_{CDH}(k) + (\log |G|)^2)$ group operations. ■

In the following Lemma we analyze the behavior of $\frac{\varphi(|G|)}{|G|}$ for group orders containing no small prime factors. This will be helpful when proving relations between certain assumptions in the sequel.

Lemma 2.5 *Let $SG_{\mathcal{G}}$ be a group sampler generating a family $\mathcal{G}$ of groups whose orders contain no small prime factors. Let $\mathcal{G}_{SG(k)}$ be the corresponding group siblings (the set of groups G returned by $SG_{\mathcal{G}}$ for a security parameter k.) Further, let $f : \mathbb{N} \mapsto \mathcal{G}$ be a function such that $f(k) \in \mathcal{G}_{SG(k)}$ and $\forall\, G' \in \mathcal{G}_{SG(k)}\,, \frac{\varphi(|G'|)}{|G'|} \geq \frac{\varphi(|f(k)|)}{|f(k)|}$. Then it follows $1 - \frac{\varphi(|f(k)|)}{|f(k)|} <_\infty 1/\mathsf{poly}(k)$.*
□

Proof. Let $|f(k)| = |G| = \prod_{i=1}^m p_i^{e_i}$ be the prime factorization of the group order $|G|$ and $p = \min(p_1, \cdots, p_m)$ be the smallest prime factor of $|G|$. Then it follows $|G| = \prod_{i=1}^m p_i \geq p^m$ and $\log |G| \geq m \log p$, and thus, $m \leq \log |G|/\log p \leq \log |G|$ for $\log p \geq 1$ (i.e., for $p \geq 2$). Moreover, as discussed in Section 2.1.7, we can assume that the group order can be upper bounded in security parameter, i.e., $|G| \leq 2^{k^d}$ for $k > 1$ and some $d > 0$. It follows $m \leq \log |G| \leq k^d$. Hence, we can write

$$\frac{\varphi(|G|)}{|G|} = \prod_{i=1}^{m}(1 - \frac{1}{p_i}) \geq (1 - \frac{1}{p})^m > (1 - \frac{1}{p})^{k^d}.$$

Since $|G|$ contains no small prime factors, it follows from the definition of no small prime (see Section 2.2) that for any real constant $c > 0$, there exists a k_0 such that for all $k > k_0$, $1/p < 1/k^c$. Thus, we can write

$$\frac{\varphi(|G|)}{|G|} \geq (1 - \frac{1}{p})^{k^d} > (1 - \frac{1}{k^c})^{k^d}.$$

According to Lemma 2.3, the relation $(1 - 1/k^c)^{k^d} \geq 1 - 1/k^{c-d}$ holds for $c > d$ and $k \in \mathbb{N}$. Since c is arbitrary, and since for all $c > d$ we can write $c' := c - d > 0$, it follows that for all $c' > 0$, there exists a k_0 such that for all $k > k_0$, $\frac{\varphi(|G|)}{|G|} > 1 - 1/k^{c'}$ and consequently $1 - \frac{\varphi(|G|)}{|G|} < 1/k^{c'}$. This means $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/\mathsf{poly}(k)$. ■

In the following lemma, we prove the reduction from CSE to CIE assumption for their high-granular version. This lemma will be helpful later when establishing the relation between CIE and CDH assumptions.

Lemma 2.6

$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CSE(c:}*\text{; g:h; f:nsprim,o)} \xrightarrow[]{\alpha' \geq \frac{2\varphi(|G|)-1}{|G|}\alpha^3;\ t'=3t+O(\log|G|)} \{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CIE(c:}*\text{; g:h; f:nsprim,o)}$$

□

Proof. We prove the following statement: Given a CIE oracle $\mathcal{O}_{CIE}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CIE(c:$*$; g:h; f:nsprim,o) with success probability $\alpha_{CIE}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CIE}}$ which breaks the assumption $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CSE(c:$*$; g:h; f:nsprim,o) with success probability $\alpha_{CSE}(k) \geq \frac{2\varphi(|G|)-1}{|G|}\alpha_{CIE}(k)^3$, using 3 oracle calls and $O(\log|G|)$ group operations.

We proceed as follows:

(i) Select $b, r_1, r_2 \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$, compute $(g^x g^{-b})^{r_1} = g^{(x-b)r_1}$ and $(g^x g^b)^{r_2} = g^{(x+b)r_2}$, and query $\mathcal{O}_{CIE}$ with $((G,g),(g^{(x-b)r_1}))$ and $((G,g),(g^{(x+b)r_1}))$. One can expect correct oracle answers with probability $\alpha_{CIE}(k)$ only if oracle inputs are legal, i.e., only if $x \pm b \in \mathbb{Z}^*_{|G|}$. This event occurs with a certain probability which will be determined later. Assuming these inputs are legal, the oracle calls are statistically independent since the input elements are randomized with r_1, r_2.[33] The oracle answers are $g^{\frac{1}{r_1(x-b)}} = \mathcal{O}_{CIE}\big(g^{(x-b)r_1}\big)$ and $g^{\frac{1}{r_2(x+b)}} = \mathcal{O}_{CIE}\big(g^{(x+b)r_2}\big)$, each time with probability $\alpha_{CIE}(k)$.

(ii) Using the oracle's answers in Step (i) compute:

$$\frac{\left(g^{\frac{1}{r_1(x-b)}}\right)^{r_1}}{\left(g^{\frac{1}{r_2(x+b)}}\right)^{r_2}} = g^{\left(\frac{1}{x-b}-\frac{1}{x+b}\right)} = g^{\frac{2b}{x^2-b^2}}.$$

The exponent $\frac{2b}{x^2-b^2}$ is an element of $\mathbb{Z}^*_{|G|}$ because of the following reasons: $b \in \mathbb{Z}^*_{|G|}$, and as assumed above, $x \pm b \in \mathbb{Z}^*_{|G|}$ which implies $x^2 - b^2 \in \mathbb{Z}^*_{|G|}$. Further, $|G|$ is odd as the group families do not have any small prime factors in the order.

(iii) Select $r_3 \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$ and query $\mathcal{O}_{CIE}$ with $\big(\,(G,g),\big(\,\big(g^{\frac{2b}{x^2-b^2}}\big)^{r_3}\,\big)\big)$ where r_3 is used for randomization to obtain statistically independent oracle call. The oracle's answer is $g^{\frac{x^2-b^2}{2br_3}} = \mathcal{O}_{CIE}\big(g^{\frac{2br_3}{x^2-b^2}}\big)$ with success probability $\alpha_{CIE}(k)$.

[33]Note that for $x \pm b \in \mathbb{Z}^*_{|G|}$ the multiplication with $r_1, r_2 \in \mathbb{Z}^*_{|G|}$ spreads $x \pm b$ randomly and uniformly over $\mathbb{Z}^*_{|G|}$, and consequently $\big(g^{(x-b)}\big)^{r_1}, \big(g^{(x+b)}\big)^{r_2}$ are randomly and uniformly distributed over input domain of $\mathcal{O}_{CIE}$, implying statistically independent oracle calls.

(iv) Compute the desired CSE instance by using the oracle's answer in Step (iii)

$$g^{x^2} = \big(g^{\frac{x^2-b^2}{2br_3}} \big)^{2br_3} g^{b^2}.$$

Success probability: The probability that both events $E_1 : x+b \in \mathbb{Z}^*_{|G|}$ and $E_2 : x-b \in \mathbb{Z}^*_{|G|}$ occur (in Step (i)) is

$$\begin{aligned}\mathbf{Prob}[E_1 \wedge E_2] &= \mathbf{Prob}[E_1] + \mathbf{Prob}[E_2] - \mathbf{Prob}[E_1 \vee E_2] \\ &= \frac{2\varphi(|G|)}{|G|} - \mathbf{Prob}[E_1 \vee E_2] \\ &\geq \frac{2\varphi(|G|)}{|G|} - 1\end{aligned}$$

where we set $\mathbf{Prob}[E_1 \vee E_2] = 1$. Obviously, this is a worst case lower bound.[34]

Each time with probability $\alpha_{CIE}(k)$ the oracle outputs the correct value. There are 3 statistically independent calls to the oracle, and so the resulting success probability of $\mathcal{A}^{\mathcal{O}_{CIE}}$ is:

$$\alpha_{CSE}(k) \geq (\frac{2\varphi(|G|)}{|G|} - 1)\alpha_{CIE}(k)^3.$$

In the following, we set $\lambda(k) := \alpha_{CIE}(k)^3$ and $\gamma(k) := \frac{2\varphi(|G|)}{|G|} - 1$ (Note that $|G|$ is a function of the security parameter k, see also Lemma 2.5).

Depending on the oracle's success probability $\alpha_{CIE}(k)$ we have the following cases:

Perfect oracle ($\alpha_{CIE}(k) \not<_\infty 1$): The resulting success probability cannot be perfect because there is a non-zero error probability when querying the CIE oracle.

Weak oracle ($\alpha_{CIE}(k) \not<_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) weak: Since $|G|$ contains no small prime factors, it follows from Lemma 2.5 that $1 - \gamma(k) = 2(1 - \frac{\varphi(|G|)}{|G|}) <_\infty 1/\mathsf{poly}(k)$. Thus, we can write $\gamma(k) >_\infty 1 - 1/\mathsf{poly}(k)$, meaning that $\gamma(k)$ is non-negligible. Further, we have $\alpha_{CIE}(k) \not<_\infty 1/\mathsf{poly}(k)$ that implies $\lambda(k) \not<_\infty 1/\mathsf{poly}(k)$. It follows $\gamma(k)\lambda(k) \not<_\infty 1/\mathsf{poly}(k)$ (see also Section 2.1.2). Finally, since $\alpha_{CSE}(k) \geq \gamma(k)\lambda(k)$, it follows $\alpha_{CSE}(k) \not<_\infty 1/\mathsf{poly}(k)$.

Invariant oracle ($\alpha_{CIE} \not<_\infty \epsilon_1$): The resulting success probability is (asymptotically) invariant: As shown in the weak case, we can write $\gamma(k) >_\infty 1 - 1/\mathsf{poly}(k)$. More precisely, for any $\epsilon' > 0$ there exist a k_0 such that for

[34]For this (worst case) lower bound to be meaningful (non-negative) the relation $\frac{\varphi(|G|)}{|G|} > \frac{1}{2}$ must hold, and clearly this is the case as $|G|$ contains no small prime factors.

all $k > k_0$, $\gamma(k) > 1 - \epsilon'$. Since $\alpha_{CIE}(k) \not<_\infty \epsilon_1$, for any k_0' there exists a $k_1 > k_0'$ such that $\alpha_{CIE}(k_1) \geq \epsilon_1$ and consequently $\lambda(k_1) \geq \epsilon_1^3$. Hence, for any $k_0' > k_0$ there exists a $k_1 > k_0'$ such that $\alpha_{CSE}(k_1) \geq \epsilon_2$ where $\epsilon_2 := (1 - \epsilon')\epsilon_1{}^3$. This means $\alpha_{CSE}(k) \not<_\infty \epsilon_2$.

Strong oracle ($1-\alpha_{CIE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) strong. For this, we first prove that $1 - \gamma(k)\lambda(k) \not\geq_\infty 1/\mathsf{poly}(k)$: As shown in the weak case, we can write $\gamma(k) >_\infty 1 - 1/\mathsf{poly}(k)$. Further, from Lemma 2.2 follows $1-\lambda(k) \not\geq_\infty 1/\mathsf{poly}(k)$.[35] Applying Lemma 2.7, we obtain $1-\gamma(k)\lambda(k) \not\geq_\infty 1/\mathsf{poly}(k)$. Since $1-\alpha_{CSE}(k) \leq 1-\gamma(k)\lambda(k)$, it follows $1 - \alpha_{CSE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

■

Lemma 2.7 *Let* $f(k)$, $g(k)$ *be functions* $\mathbb{N} \to [0,1]$. *If* $1 - f(k) \not\geq_\infty 1/\mathsf{poly}(k)$ *and* $1 - g(k) <_\infty 1/\mathsf{poly}(k)$, *then* $1 - f(k)g(k) \not\geq_\infty 1/\mathsf{poly}(k)$. □

Proof. Set $P(k) := (1 - f(k))(1 - g(k))$. Obviously, $P(k) \geq 0$ for all k, and we can write

$$\begin{aligned}(1 - f(k))(1 - g(k)) &= 1 + f(k)g(k) - f(k) - g(k)\\ &= -(1 - f(k)g(k)) + (1 - f(k)) + (1 - g(k))\\ &= -(1 - f(k)g(k)) + S(k) \geq 0\end{aligned}$$

where we set $S(k) := (1 - g(k)) + (1 - f(k))$. $S(k)$ is a not non-negligible function since it is the sum of a negligible and a not non-negligible functions. Thus, we can write $S(k) \not\geq_\infty 1/\mathsf{poly}(k)$. From equations above we have $1 - f(k)g(k)) \leq S(k)$. Since $S(k) \not\geq_\infty 1/\mathsf{poly}(k)$, it follows $1 - f(k)g(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

Efficiency: There are 3 calls to $\mathcal{O}_{CIE}$, and $O(\log|G|)$ group operations are required for the exponentiations and computing the inverse elements. ■

Remark 2.4. For groups of prime order, the resulting success probability covers also perfect success probability as the special case of elements not in $\mathbb{Z}_p^*$ (i.e., 0) can be tested and handled. ◦

Using the previous results we can now prove the relation between CIE and CDH assumptions in their high-granular variant.

[35] $\lambda(k)$ is a (polynomial) power of a strong success probability.

Theorem 2.5

$*$-CIE(c:$*$; g:h; f:fct)

$$\xRightarrow{\alpha' \geq 1-1/2^k;\ t' = O(\frac{tk}{\alpha} + t\log|G|) + O(\frac{k\log|G|}{\alpha} + (\log|G|)^2)}$$

$*$-CDH(c:$*$; g:h; f:fct);

$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CIE(c:$*$; g:h; f:nsprim,o)

$$\xLeftarrow{\alpha' \geq (\frac{2\varphi(|G|)-1}{|G|})^3\alpha^9;\ t' = 9t + O(\log|G|)}$$

$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CDH(c:$*$; g:h; f:nsprim,o)

□

Proof. The following statements hold:

(a) Given a CDH oracle $\mathcal{O}_{CDH}$ which breaks $*$-CDH(c:$*$; g:h; f:fct) with success probability $\alpha_{CDH}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CDH}}$ which breaks $*$-CIE(c:$*$; g:h; f:fct) with success probability $\alpha_{CIE}(k) = \alpha_{CDH}(k)^{O(\log|G|)}$, using at most $O(\frac{k}{\alpha_{CDH}(k)} + \log|G|)$ calls to $\mathcal{O}_{CDH}$ and $O(\frac{k\log|G|}{\alpha_{CDH}(k)} + (\log|G|)^2)$ group operations.

(b) Given a CIE oracle $\mathcal{O}_{CIE}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CIE(c:$*$; g:h; f:nsprim,o) with success probability $\alpha_{CIE}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CIE}}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CDH(c:$*$; g:h; f:nsprim,o) with success probability $\alpha_{CDH}(k) \geq (\frac{2\varphi(|G|)-1}{|G|})^3\alpha_{CIE}(k)^9$, using 9 oracle calls and $O(\log|G|)$ group operations.

Case (a): Follows immediately from lemma 2.4.

Case (b): According to Theorem 2.3 there is a reduction from $*$-CDH(c:$*$; g:h; f:o) to $*$-CSE(c:$*$; g:h; f:o) with success probability $\alpha_{CDH}(k) = \alpha_{CSE}(k)^3$, using 3 calls to $\mathcal{O}_{CDH}$ and $O(\log|G|)$ group operations. Further, according to Lemma 2.6 there is a reduction from $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CSE(c:$*$; g:h; f:nsprim,o) to $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CIE(c:$*$; g:h; f:nsprim,o) with success probability $\alpha_{CSE}(k) \geq \frac{2\varphi(|G|)-1}{|G|}\alpha_{CIE}(k)^3$, using 3 oracle calls and $O(\log|G|)$ group operations. Combining these results we obtain a reduction from $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CDH(c:$*$; g:h; f:nsprim,o) to $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CIE(c:$*$; g:h; f:nsprim,o) with the resulting success probability $\alpha_{CDH}(k) \geq (\frac{2\varphi(|G|)-1}{|G|})^3\alpha_{CIE}(k)^9$, using 9 oracle calls and $O(\log|G|)$ group operations. This completes the proof.

■

Remark 2.5. Theorem 2.5 concerns only group orders with no small prime factors, and does not cover the gap between the group orders with at least one large prime factor and those containing no small primes. Note that for group orders with only small prime factors these problems are easy to solve, since one can apply well-known algorithms for solving the discrete logarithms (see Shoup (1997)).

Remark 2.6. The CDH oracle can be used to multiply two discrete logarithms without knowing them explicitly (e.g., to compute g^{x^2} without knowing x). Using CDH oracle one can compute $g^{p(x)}$ for a polynomial $p(x)$ with integer coefficients or to compute $g^{h(x)}$ for any rational function of the form $h(x) = f(x)/g(x)$ where $f(x), g(x)$ are polynomials with integer coefficients. This fact was also mentioned shortly by Maurer (1994). As a consequence one can use CDH oracle to compute any multivariate polynomial $p(x_1, x_2, \cdots x_n)$ or rational function $h(x_1, x_2, \cdots, x_n)$ in the exponent.

Let us, for brevity, consider bivariate expressions in exponents. Assume we are given an oracle which, on input g^x, g^y, outputs $g^{p(x,y)}$ with a certain probability, where $p(x, y)$ is a known (fixed) bivariate polynomial whose degree and form is appropriately defined. We may call this oracle CPE (Computational Polynomial Exponent) and want to analyze its relation to CDH oracle. Due to the discussion above, we can easily construct a CPE oracle using a CDH oracle, however, the converse (reduction CDH to CPE) is not obvious. This direction was shown in Kiltz (2001): First show the equivalence between CDH oracle and a CPE oracle which computes polynomials of degree 2 (according to the underlying polynomial definition). We denote such oracle with CPE(2). Next prove that a CPE(n), i.e, a CPE oracle outputting $p(x, y)$ of degree n, can be inductively reduced to CPE(2). ∘

2.5.4.2 Medium Granular

Next, we prove the above equivalence (Theorem 2.5) also for medium granularity. Similar to Theorem 2.4, we could argue that due to the existence of a randomization the result immediately follows also for the medium-granular case. However, we will show that this reduction can be performed much more efficiently in the medium-granular case than in the high-granular case; thereby we improve the concrete security considerably. We start with the following lemma.

Lemma 2.8

$*$-CIE(c:$*$; g:m; f:o)

$$\xRightarrow{\alpha'=\alpha;\ t'=t+O(\log|G|)}$$

$*$-CSE(c:$*$; g:m; f:o)*;*

$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CIE(c:$*$; g:m; f:nsprim)

$$\xLeftarrow{\alpha'\geq\frac{\varphi(|G|)}{|G|}\alpha;\ t'=t}$$

$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CSE(c:$*$; g:m; f:nsprim)

$\square$

Proof. We prove that the following statements hold:

(a) Given a CSE oracle $\mathcal{O}_{CSE}$ which breaks $*$-CSE(c:$*$; g:m; f:o) with success probability $\alpha_{CSE}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CSE}}$ that breaks $*$-CIE(c:$*$; g:m; f:o) with success probability $\alpha_{CIE}(k) = \alpha_{CSE}(k)$, using 1 oracle call and $O(\log|G|)$ group operations.

(b) Given a CIE oracle $\mathcal{O}_{CIE}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CIE(c:$*$; g:h; f:nsprim) with success probability $\alpha_{CIE}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{CIE}}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-CSE(c:$*$; g:h; f:nsprim) with success probability $\alpha_{CSE}(k) \geq \frac{\varphi(|G|)}{|G|}\alpha_{CIE}(k)$, using 1 oracle call.

Case (a): Given a CIE input tuple $((G,g),(g^x))$ with $x \in \mathbb{Z}^*_{|G|}$, we construct $\mathcal{A}^{\mathcal{O}_{CSE}}$ as follows: Set $h := g^x$, then we have $g = h^t$ for some $t \in \mathbb{Z}^*_{|G|}$. Since $x \in \mathbb{Z}^*_{|G|}$, h is a group generator, and $t = x^{-1}$ exists as we implicitly assumed above. Select $r \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ and pass $((G,h),\ (h^{t+r}))$ to $\mathcal{O}_{CSE}$ where $h^{t+r} = gg^r$. The reason for the randomization with r is that here the inputs to $\mathcal{O}_{CSE}$ are limited to those with secret exponents x from $\mathbb{Z}^*_{|G|}$ whereas the success probability of $\mathcal{O}_{CSE}$ is defined over the input set with $x \in \mathbb{Z}_{|G|}$.

Using the answer of $\mathcal{O}_{CSE}$ we compute

$$h^{t^2} = \frac{\mathcal{O}_{CSE}(h^{t+r})}{g^{2r}h^{r^2}} = \frac{h^{t^2+2rt+r^2}}{(h^t)^{2r}h^{r^2}}.$$

Since $t = x^{-1}$, we exploit the identity $h^{t^2} = (g^x)^{(x^{-1})^2} = (g^x)^{x^{-2}} = g^{xx^{-2}} = g^{x^{-1}}$ to solve CIE input.

Success probability: There is a single call to $\mathcal{O}_{CSE}$. Thus, the resulting success probability is $\alpha_{CIE}(k) = \alpha_{CSE}(k)$.

Efficiency: There is a single oracle call, and there are $O(\log|G|)$ group operations required for computing the inverses and exponentiations.

Case (b): Given a CSE input tuple $((G,g),(g^x))$, we construct $\mathcal{A}^{\mathcal{O}_{CIE}}$ as follows: Set $h := g^x$ then we have $g = h^t$ for some t. Next, pass $((G,h),(h^t))$ to $\mathcal{O}_{CIE}$. Note that $\mathcal{O}_{CIE}$ answers correctly (i.e., $h^{t^{-1}}$) with probability $\alpha_{CIE}(k)$ only to the legal queries, i.e., when h is a generator. The probability for this event is $\frac{\varphi(|G|)}{|G|}$. Note that in this case $t := x^{-1}$ exists, as we implicitly assumed above. The desired solution to the CSE problem is obtained by exploiting the identity $h^{t^{-1}} = (g^x)^{(x^{-1})^{-1}} = g^{x^2}$.

Success probability: There is a single call to $\mathcal{O}_{CIE}$. Thus, the resulting success probability is $\alpha_{CSE}(k) \geq \frac{\varphi(|G|)}{|G|}\alpha_{CIE}(k)$. Depending on the oracle's success probability $\alpha_{CIE}(k)$ we have the following cases:

Perfect oracle ($\alpha_{CIE}(k) \not<_\infty 1$): The resulting success probability cannot be perfect because there is a non-zero error probability when querying the CIE oracle.

Weak oracle ($\alpha_{CIE}(k) \not<_\infty 1/\mathsf{poly}(k)$): The resulting success probability is weak: $\alpha_{CIE}(k)$ is a not negligible function, and $\frac{\varphi(|G|)}{|G|}$ is always non-negligible (see also Lemma 2.5) Thus, the product of these terms is a not-negligible function, implying $\alpha_{CSE}(k) \not<_\infty 1/\mathsf{poly}(k)$.

Invariant oracle ($\alpha_{CIE} \not<_\infty \epsilon_1$): The resulting success probability is (asymptotically) invariant. The proof is similar to that of the invariant case in Lemma 2.6: Since $|G|$ contains no small prime factors, it follows from Lemma 2.5 that $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/\mathsf{poly}(k)$. More precisely, for all $\epsilon' > 0$ there exist a k_0 such that for all $k > k_0$, $\frac{\varphi(|G|)}{|G|} > 1 - \epsilon'$. Since $\alpha_{CIE}(k) \not<_\infty \epsilon_1$, for any k'_0 there exists $k_1 > k'_0$ such that $\alpha_{CIE}(k_1) > \epsilon_1$. Thus, for any $k'_0 > k_0$ there exists $k_1 > k'_0$ such that $\alpha_{CSE}(k_1) > \epsilon_2$ where $\epsilon_2 := (1 - \epsilon')\epsilon_1$. This means $\alpha_{CSE}(k) \not<_\infty \epsilon_2$.

Strong oracle ($1-\alpha_{CIE}(k) \not>_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) strong. The proof is similar to that of the strong case in Lemma 2.6: Since $|G|$ contains no small prime factors, it follows from Lemma 2.5 that $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/\mathsf{poly}(k)$. Further, we have $1-\alpha_{CIE}(k) \not>_\infty 1/\mathsf{poly}(k)$. Applying Lemma 2.7, we obtain $1 - \frac{\varphi(|G|)}{|G|}\alpha_{CIE}(k) \not>_\infty 1/\mathsf{poly}(k)$, and since $1 - \alpha_{CSE}(k) \leq 1 - \frac{\varphi(|G|)}{|G|}\alpha_{CIE}(k)$, it follows $1 - \alpha_{CSE}(k) \not>_\infty 1/\mathsf{poly}(k)$.

■

Combining Theorem 2.4 and Lemma 2.8, we obtain the following theorem on the relation between the medium-granular variants of CIE and CDH assumptions.

Theorem 2.6

$$\ast\text{-CIE(c:}\ast\text{; g:m; f:o)} \xRightarrow{\alpha'=\alpha;\ t'=t+O(\log|G|)} \ast\text{-CDH(c:}\ast\text{; g:m; f:o)};$$

$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CIE(c:}\ast\text{; g:m; f:nsprim)} \xLeftarrow{\alpha' \geq \left(\frac{\varphi(|G|)}{|G|}\right)^3\alpha^3;\ t'=3t+O(\log|G|)} \{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CDH(c:}\ast\text{; g:m; f:nsprim)}$$

□

Remark 2.7. In Theorem 2.6 we consider group orders containing no small prime factors (for the reduction CDH to CIE) to obtain comparable results to the high-granular variant of the reduction. However, the reduction holds also for general group orders[36] although for invariant and strong CIE oracle we need to self-correct the resulting success probability α_{CDH} *after* the reduction (see Corollary 2.1) This is stated in the following lemma.

Lemma 2.9

$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CIE(c:}*\text{; g:m; f:}*\text{)}$$
$$\Longleftarrow$$
$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CDH(c:}*\text{; g:m; f:}*\text{)}$$
□

Remark 2.8. For prime group orders Theorem 2.6 also covers perfect oracles as the special case of elements not in $\mathbb{Z}_p^*$ (i.e., 0) can be tested and handled. ∘

In this section we have analyzed and proved relations between CDH, CSE, CIE in their high- and medium-granular versions. We can summarize the advantages of medium-granular reductions over their high-granular variants as follows:

- The medium-granular reduction of CIE to CDH (Theorem 2.5) does not require the CDH oracle to be self-corrected.
- The medium-granular reduction (Theorem 2.6) is much more efficient than the corresponding high-granular reduction (Theorem 2.5): The reduction CIE-CDH requires a single call to the CDH oracle and $O(\log|G|)$ group operations whereas the high-granular version requires, even without self-correction, $O(\log|G|)$ (very expensive) oracle calls and $O((\log|G|)^2)$ group operations. Further, the reduction achieves a success probability which is higher by a power of $O(\log|G|)$. The success probability of the converse reduction CDH-CIE is comparatively higher for the medium-granular variant.
- The high-granular variant of Theorem 2.6 works for group orders with no small prime factors. It does not cover the range of group orders with at least one large prime factor to those with no small prime factors. However, this gap is covered by the medium-granular version since the reduction works for any group order (see Remarks 2.5 and 2.7).

[36] e.g., it holds also for orders containing at least one large prime factor.

2.6 Decisional DH, SE and IE

2.6.1 Difficulty in the Generic Model

First we state a Lemma which plays an important role for later proofs in the context of generic algorithms:

Lemma 2.10 (Schwartz 1980; Shoup 1997) *Let $p \in \mathbb{P}$ and $e \in \mathbb{N}$. Further, let $P(X_1, X_2, \cdots, X_n)$ be a non-zero polynomial in $\mathbb{Z}_{p^e}[X]$ of total degree $d \geq 0$. Then*

$$\boldsymbol{Prob}[P(x_1, x_2, \cdots, x_n) \equiv 0 :: (x_1, x_2, \cdots, x_n) \in_{\mathcal{R}} \mathbb{Z}_{p^e}^n] \leq d/p.$$

□

Using Lemma 2.10, Wolf (1999) shows the following result: There exists no generic algorithm that can solve DSE in polynomial time if the order of the multiplicative group is not divisible by small primes. This result is summarized in the following theorem:

Theorem 2.7 (Wolf 1999)
true $\Longrightarrow$ ϵ-DSE$^\sigma$(c:$*$; g:h; f:nsprim,o) □

Remark 2.9. More precisely, Wolf (1999) shows, the probability that any generic algorithm $\mathcal{A}^\sigma$ can distinguish correct DSE inputs from incorrect ones is at most $\frac{(T+4)(T+3)}{2p'}$ where p' is the smallest prime factor of $|G|$ and T is an upper bound on the algorithm's runtime.

Remark 2.10. In the sequel, we will consider several decisional problems and prove results on the generic complexity of solving them. For known group orders the generic complexity of these problems is directly proportional to the smallest prime factor of the group order (similar to the result of Theorem 2.7) In other words, these problems can only be hard if the group order $|G|$ is free of small primes.[37]

To make this more clear, consider the following example regarding DDH for group family with $|G| = 2q$ where $q \in \mathbb{P}$ is a prime. We assume that DDH is hard for q. Note that this example can be generalized to any group families with $|G|$ containing small prime factors. Now, assume we are given a correct DDH tuple $I_1 := ((G, g), (g^x, g^y), (g^{xy}))$ and a random DDH tuple $I_0 := ((G, g), (g^x, g^y), (g^{x'y'}))$ with $x', y' \in_{\mathcal{R}} \mathbb{Z}_{|G|}$. Since $|G|$ is even, we can determine the parity $\mathsf{parity}(exp) := exp \bmod 2$ of the exponents $exp \in \{x, y, xy, x'y'\}$ as follows:

$$\mathsf{parity}(exp) = \begin{cases} 0 & : \text{ if } (g^{exp})^q = 1 \\ 1 & : \text{ if } (g^{exp})^q = g^q. \end{cases}$$

[37]Note that if the group order $|G|$ is known, then its small prime factors can easily be computed by using well-known factoring algorithms.

This can be exploited to construct an algorithm (distinguisher) D which solves DDH in G with non-negligible success probability. D gets the tuple I_b as input, where b is a randomly and uniformly chosen bit, and outputs a bit $D(I_b)$ such that

$$D(I_b) := \begin{cases} 1 & : \text{ if } \mathsf{parity}(z) = \mathsf{parity}(x)\mathsf{parity}(y) \\ 0 & : \text{ otherwise} \end{cases}$$

where z is either xy or $x'y'$. D is successful if and only if $D(I_b) = b$. The success probability of D is determined as follows:

$$\begin{aligned} \mathbf{Prob}[D(I_b) = b] &= \mathbf{Prob}[D(I_b) = 1 | b = 1]\mathbf{Prob}[b = 1] \\ &\quad + \mathbf{Prob}[D(I_b) = 0 | b = 0]\mathbf{Prob}[b = 0] \\ &= 1(\frac{1}{2}) + \frac{1}{2}\mathbf{Prob}[D(I_b) = 0 | b = 0] \\ &= \frac{1}{2} + \frac{1}{2}\mathbf{Prob}[D(I_b) = 0 | b = 0]. \end{aligned}$$

Note that $\mathbf{Prob}[D(I_b) = 1 | b = 1] = 1$ always holds. Further, $\mathbf{Prob}[b = 1] = \mathbf{Prob}[b = 0] = 1/2$ holds since b is chosen randomly and uniformly from $\{0, 1\}$. It remains to compute $\mathbf{Prob}[D(I_b) = 0 | b = 0]$. For this, we consider the 4 possible cases for the parities of x and y represented by the disjoint events $E_{i,j}$

$$E_{i,j} := \{(i, j) : x, y \in_{\mathcal{R}} \mathbb{Z}_{|G|} \wedge i = \mathsf{parity}(x) \wedge j = \mathsf{parity}(y)\}$$

for $i, j \in \{0, 1\}$. It follows

$$\begin{aligned} \mathbf{Prob}[D(I_b) = 0 | b = 0] &= \sum_{i,j \in \{0,1\}} \mathbf{Prob}[D(I_b) = 0 | b = 0 \wedge E_{i,j}]\mathbf{Prob}[E_{i,j}] \\ &= \mathbf{Prob}[D(I_b) = 0 | b = 0 \wedge E_{1,1}]\mathbf{Prob}[E_{1,1}] \\ &\quad + \sum_{\substack{i,j \in \{0,1\} \\ (i,j) \neq (1,1)}} \mathbf{Prob}[D(I_b) = 0 | b = 0 \wedge E_{i,j}]\mathbf{Prob}[E_{i,j}]. \end{aligned}$$

Since x and y are chosen uniformly and randomly we have $\mathbf{Prob}[E_{i,j}] = 1/4$ for all $i, j \in \{0, 1\}$. Further, we have

$$\mathbf{Prob}[D(I_b) = 0 | b = 0 \wedge E_{1,1}] = \mathbf{Prob}[\mathsf{parity}(z) = 0] = 3/4$$

and for $i, j \in \{0, 1\}, (i, j) \neq (1, 1)$

$$\mathbf{Prob}[D(I_b) = 0 | b = 0 \wedge E_{i,j}] = \mathbf{Prob}[\mathsf{parity}(z) = 1] = 1/4.$$

Substituting these results in the above equations we obtain:

$$\mathbf{Prob}[D(I_b) = 0 | b = 0] = 1/4(3/4) + 1/4(3/4) = 6/16$$

and

$$\mathbf{Prob}[D(I_b) = b] = 1/2 + 1/2(6/16) = 11/16.$$

According to our definitions of the assumptions in Section 2.3, the adversary's success probability for decisional assumptions is normalized to $(\mathbf{Prob}[D(I_b) = b] - 1/2)2 = 6/16$. Thus, with (non-negligible) success probability 6/16 the distinguisher can recognize the correct DDH tuple.

Remark 2.11. Theorem 2.7 holds also for other variants of the assumption with respect to the success probabilities perfect, weak and strong.

Remark 2.12. It might look surprising that $*$-DSE$^\sigma$(c:$*$; g:h; f:nsprim) always holds, i.e., it's a fact, not an assumption. Of course, the crucial aspect is the rather restricted adversary model (the σ in the assumption statement) which limits adversaries to generic algorithms. However, note that, consequently, to break DSE, one has to exploit deeper knowledge on the actual structure of the used algebraic groups. In particular, for appropriately chosen prime-order subgroups of $\mathbb{Z}_p^*$ and elliptic or hyper-elliptic curves no such exploitable knowledge could yet be found, and all of currently known efficient and relevant algorithms in these groups are generic algorithms, e.g., Pohlig-Hellman (1978) or Pollard-ρ (Pollard 1978). Nevertheless, care has to be applied when proving systems secure in the generic model (Fischlin 2000).

Remark 2.13. As we will see later several (impossibility) results are proven in the generic model. All these proofs use similar techniques to determine bounds on the amount of information a generic adversary can obtain. For the better understanding, we describe this below by giving an example for DDH.

First recall that in the generic model a group element $a \in G$ is represented by its encoding $\sigma(x)$, with $x \in \mathbb{Z}_{|G|}$, using an encoding function $\sigma(\cdot)$ chosen randomly from the set $\Sigma_{G,g}$ of bijective functions $\mathbb{Z}_{|G|} \rightarrow G$. The generic adversary $\mathcal{A}^\sigma$ is given $\sigma(1)$, i.e., the encoding of a generator, and it is given access to oracles for performing addition $\sigma(x+y) \leftarrow \sigma_+(\sigma(x), \sigma(y))$ and inversions $\sigma(-x) \leftarrow \sigma_-(x)$ on group elements (see also Section 2.2)[38]

A decisional problem in this model is formulated as follows: The adversary $\mathcal{A}^\sigma$ is given the encodings of the secret, solution and random parts where the latter two parts are in random order. Then $\mathcal{A}^\sigma$ has to decide on the correct order of these two parts. For DDH it means, $\mathcal{A}^\sigma$ is given $\sigma(1), \sigma(x), \sigma(y)$ and the elements $\{\sigma(xy), \sigma(c)\}$ in random order. Now, $\mathcal{A}^\sigma$ has to decide which of the elements $\{\sigma(xy), \sigma(c)\}$ is the encoding of the solution part and which one corresponds to the encoding of the random part. Assuming $\mathcal{A}^\sigma$ makes T queries to the addition and inversion oracles, we are

[38]As we will see later, for proving the impossibility of some reductions, $\mathcal{A}^\sigma$ is given access to additional oracles.

interested in the amount of information it can obtain. Each time $\mathcal{A}^\sigma$ interacts with these oracles it learns the encoding $\sigma(w_i)$ of a $w_i \in \mathbb{Z}_{|G|}$ where $w_i = P_i(x, y, xy, c)$ is a linear function P_i in x, y, c, and can be determined by using the previous oracle queries. $\mathcal{A}^\sigma$ has the following possibilities to obtain information on the encoded values:

(a)$\mathcal{A}^\sigma$ learns distinct (random) encoding of distinct values. More precisely, for all (i, j) with $P_i \neq P_j$ we have $\sigma(w_i) \neq \sigma(w_j)$.

(b)$\mathcal{A}^\sigma$ learns a linear relation on the values x, y, xy and c. More precisely, there exists (i, j) such that $P_i \neq P_j$ and $\sigma(w_i) = \sigma(w_j)$, meaning that either $P_i(x, y, xy, c) \equiv P_j(x, y, xy, c) \bmod |G|$ holds or $P_i(x, y, c, xy) \equiv P_j(x, y, c, xy) \bmod |G|$. Recall that the order of xy, c is random.

In the case (a), the obtained values are independent random values and do not leak any information to $\mathcal{A}^\sigma$ at all. In contrast, case (b) represents the only way $\mathcal{A}^\sigma$ may obtain information on the values x, y, xy and c. If $\mathcal{A}^\sigma$ can find such a relation, we consider it as successful in finding the correct order of the elements. Hence, we are interested in bounding the probability of $\mathcal{A}^\sigma$'s success. For this purpose, it suffices to bound the probability that a pair (i, j) with $i \neq j$ exists such that $P_i(x, y, xy, c) \equiv P_j(x, y, xy, c) \bmod |G|$ or $P_i(x, y, c, xy) \equiv P_j(x, y, c, xy) \bmod |G|$ when given any T distinct linear polynomials and the random values $x, y, xy, c \in_{\mathcal{R}} \mathbb{Z}_{|G|}$. As we will see later this bound is determined by exploiting the result of Theorem 2.10. ∘

In the following Theorem, we show that also DIE cannot be solved by generic algorithms if the order $|G|$ of the multiplicative group $\mathbb{Z}^*_{|G|}$ is not divisible by small primes.

Theorem 2.8
true $\Longrightarrow$ $*$-DIE$^\sigma$(c:$*$; g:h; f:nsprim) □

Proof. The following lemma associates the minimal generic complexity of solving DIE directly to the smallest prime factor of the order of the underlying group G. Theorem 2.8 immediately follows from this lemma and Remarks 2.14. ■

Lemma 2.11 *Let G be a cyclic group and g a corresponding generator, let p' be the smallest prime factor of $|G|$. Let $\mathcal{A}^\sigma$ be any generic algorithm for groups G with maximum run time T. Then the following always holds:*

$$\begin{array}{l}
(|\ \boldsymbol{Prob}[\mathcal{A}^\sigma(\mathcal{C}, (G, g), w_b, w_{\bar{b}}) = b\ ::\\
\quad b \overset{\mathcal{R}}{\leftarrow} \{0,1\};\ \mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U};\\
\quad PI \leftarrow SPI_{IE}((G, g));\ PI_{\mathcal{R}} \leftarrow SPI_{PI^{\mathcal{P}}}(PI^{SI});\\
\quad w_b \leftarrow (PI^{publ}, PI^{sol});\\
\quad w_{\bar{b}} \leftarrow (PI^{publ}, {PI_{\mathcal{R}}}^{sol})\\
\ ]-1/2\ |\cdot 2)\ \leq \frac{2(T+4)(T+3)}{p'-2}
\end{array}$$

□

Proof. Assume, we are given the encodings $\sigma(1)$, $\sigma(x)$ and $\{\sigma(x^{-1}), \sigma(c)\}$ where $x \in \mathbb{Z}^*_{|G|}$. After T computation steps the algorithm $\mathcal{A}^\sigma$ can compute at most $T+4$ distinct linear combinations P_i of the elements $1, x, x^{-1}$ and c, i.e., it obtains

$$\sigma(P_i(1, x, x^{-1}, c)) = \sigma(a_{i1} + a_{i2}x + a_{i3}x^{-1} + a_{i4}c),$$

where a_{ij} are constant coefficients. Furthermore, it is not a-priori known to $\mathcal{A}^\sigma$ which one of the values in $\{a_{i3}, a_{i4}\}$ is the coefficient for x^{-1} and which one corresponds to c. $\mathcal{A}^\sigma$ may be able to distinguish $\sigma(x^{-1})$ and $\sigma(c)$ by finding relations (collisions) between distinct linear combinations (P_i, P_j) with $i \neq j$. This means it obtains $\sigma(P_i(1, x, x^{-1}, c)) = \sigma(P_j(1, x, x^{-1}, c))$ or $\sigma(P_i(1, x, c, x^{-1})) = \sigma(P_j(1, x, c, x^{-1}))$, implying either $P_i(1, x, x^{-1}, c) \equiv P_j(1, x, x^{-1}, c) \bmod |G|$ or $P_i(1, x, c, x^{-1}) \equiv P_j(1, x, c, x^{-1}) \bmod |G|$. Let E denote this event. We compute an upper bound for the probability that E occurs: There are $\binom{T+4}{2} = \frac{(T+4)(T+3)}{2}$ possible distinct pairs of polynomials (P_i, P_j). For each such a pair (i, j) we can bound the number of solutions to $P_i \equiv P_j \bmod p^e$ for any prime power p^e that exactly divides $|G|$, i.e., $p^{e+1} \not| |G|$. Note that uniformly distributed random values mod $|G|$ are also randomly and uniformly distributed $\bmod p^e$. More precisely, we consider the solutions to the following polynomials

$$F_{i,j}(x, c) := x[P_i(1, x, x^{-1}, c) - P_j(1, x, x^{-1}, c)] \equiv 0 \bmod p^e$$

or

$$G_{i,j}(x, c) := x[P_i(1, x, c, x^{-1}) - P_j(1, x, c, x^{-1})] \equiv 0 \bmod p^e.$$

Here, the polynomials F or G are obtained by multiplying both sides of the congruence $P_i \equiv P_j \bmod p^e$ with x and then reordering the resulting congruence.

Hence, we bound the probability that a random tuple $(x, c) \in_{\mathcal{R}} \mathbb{Z}^*_{p^e} \times \mathbb{Z}^*_{p^e}$ is a zero of the polynomials F or $G \bmod p^e$. Note that $\mathbb{Z}^*_{|G|}$ is the domain of the secret exponents of DIE input tuples.

To do this, we first bound the number of solutions (x, c) to F or $G \bmod p^e$ where (x, c) are randomly selected from $\mathbb{Z}^2_{p^e}$: The total degree of each of the polynomials F and G is two. It follows from Lemma 2.10 that the probability of a random tuple $(x, c) \in \mathbb{Z}^2_{p^e}$ to be a zero of F or $G \bmod p^e$ is at most $2(2/p) = 4/p$. There are p^{2e} tuples (x, c) in $\mathbb{Z}^2_{p^e}$. Thus, there are at most $p^{2e}4/p = 4p^{2e-1}$ zeros for F or $G \bmod p^e$. Further, there are $(\varphi(p^e))^2 = (p^e - p^{e-1})^2$ tuples in $\mathbb{Z}^*_{p^e} \times \mathbb{Z}^*_{p^e}$.

Hence, the probability that such a tuple is a zero of F or $G \bmod p^e$ is upper bounded by $4p^{2e-1}/(p^e - p^{e-1})^2$. It follows

$$\begin{aligned}
\mathbf{Prob}[E] &\leq \frac{(T+4)(T+3)}{2}\frac{4p^{2e-1}}{(p^e-p^{e-1})^2}\\
&= \frac{(T+4)(T+3)}{2}\frac{4p^{2e-1}}{p^{2e}+p^{2e-2}-2p^{2e-1}}\\
&= (T+4)(T+3)\frac{2p}{p^2-2p+1}\\
&\leq (T+4)(T+3)\frac{2p}{p^2-2p} = (T+4)(T+3)\frac{2}{p-2}\\
&\leq (T+4)(T+3)\frac{2}{p'-2}.
\end{aligned}$$

If the complementary event $\bar{E}$ occurs, then $\mathcal{A}^\sigma$ cannot obtain any information about the bit b except pure guessing. Thus, the success probability of $\mathcal{A}^\sigma$ for correctly outputting b is

$$\begin{aligned}
\mathbf{Prob}[\mathcal{A}^\sigma(..) = b] &= \mathbf{Prob}[E] + \frac{1}{2}\mathbf{Prob}[\bar{E}]\\
&= \mathbf{Prob}[E] + \frac{1-\mathbf{Prob}[E]}{2}\\
&= \frac{1}{2} + \frac{\mathbf{Prob}[E]}{2}\\
&\leq \frac{1}{2} + \frac{(T+4)(T+3)}{p'-2}.
\end{aligned}$$

■

Remark 2.14. In the classical formulation of decision problems the adversary gets, depending on the challenge b, either the correct element or a random element as input, i.e., in the case of DIE the adversary gets g^x together with $g^{x^{-1}}$ if $b = 0$ and g^c (with $c \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$) if $b = 1$. The formulation used in Lemma 2.8 considers a slightly different variant of the decisional problem type: We consider here an adversary which receives, in random order, both, the correct and a random element and the adversary has to decide on the order of the elements, i.e., the adversary gets g^x and $(g^{x^{-1}}, g^c)$ for $b = 0$ and $(g^c, g^{x^{-1}})$ for $b = 1$.

This formulation makes the proofs easier to understand. However, note that both variants are equivalent. ○

Next, we consider the Decisional Strong Diffie-Hellman (DSDH) Assumption from (Pfitzmann and Sadeghi 1999) in generic model and show in the following lemma that its minimal generic complexity is associated to the smallest prime factor of the order of the underlying group G. Theorem 2.9 immediately follows from Lemma 2.12 and Remark 2.14.

Theorem 2.9

$\mathit{true} \implies *\text{-DSDH}^{\sigma}(\text{c:}*;\ \text{g:h};\ \text{f:nsprim})$ □

Lemma 2.12 *Let G be a cyclic group and g a corresponding generator, let p' be the smallest prime factor of $|G|$. Let $\mathcal{A}^\sigma$ be any generic algorithm for groups G with maximum run time T. Then the following always holds:*

$$\begin{aligned}
&(|\ \boldsymbol{Prob}[\mathcal{A}^\sigma(\mathcal{C},(G,g),w_b,w_{\bar b}) = b ::\\
&\quad b \xleftarrow{\mathcal{R}} \{0,1\};\ \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U};\\
&\quad PI \leftarrow SPI_{SDH}((G,g));\ PI_{\mathcal{R}} \leftarrow SPI_{PI^{\mathcal{P}}}(PI^{SI});\\
&\quad w_b \leftarrow (PI^{publ}, PI^{sol});\\
&\quad w_{\bar b} \leftarrow (PI^{publ}, {PI_{\mathcal{R}}}^{sol})\\
&\quad]-1/2\ |\cdot 2)\ \le \tfrac{3(T+6)(T+5)}{p'-1}
\end{aligned}$$

□

Proof. Assume, we are given the encodings $\sigma(1)$, $\sigma(x)$, $\sigma(x^{-1})$, $\sigma(y)$ and $\{\sigma(xy), \sigma(c)\}$ where $x \in \mathbb{Z}^*_{|G|}$ and $y \in \mathbb{Z}_{|G|}$. After T computation steps the algorithm $\mathcal{A}^\sigma$ can compute at most $T+6$ distinct linear combinations $P_i(1, x, x^{-1}, y, xy, c)$ of the elements $1, x, x^{-1}, y, xy$ and c, i.e., it obtains

$$\sigma(P_i(1, x, x^{-1}, y, xy, c)) = \sigma(a_{i1} + a_{i2}x + a_{i3}x^{-1} + a_{i4}y + a_{i5}xy + a_{i6}c),$$

where a_{ij} are constant coefficients. Furthermore, it is not a-priori known to $\mathcal{A}^\sigma$ which one of the values in $\{a_{i5}, a_{i6}\}$ is the coefficient for xy and which one corresponds to c. $\mathcal{A}^\sigma$ may be able to distinguish $\sigma(xy)$ and $\sigma(c)$ by finding relations (collisions) between distinct linear combinations (P_i, P_j) with $i \neq j$. This means it obtains either $\sigma(P_i(1, x, x^{-1}, y, xy, c)) = \sigma(P_j(1, x, x^{-1}, y, xy, c))$ or $\sigma(P_i(1, x, x^{-1}, y, c, xy)) = \sigma(P_j(1, x, x^{-1}, x, y, c, xy))$, implying either $P_i(1, x, x^{-1}, y, xy, c) \equiv P_j(1, x, x^{-1}, x, y, xy, c) \bmod |G|$ or $P_i(1, x, x^{-1}, y, c, xy) \equiv P_j(1, x, x^{-1}, x, y, c, xy) \bmod |G|$. Let E denote this event. We compute an upper bound for the probability that E occurs: There are $\binom{T+6}{2} = \frac{(T+6)(T+5)}{2}$ distinct pairs of polynomials (P_i, P_j). For each such a pair (i, j) we can bound the number of possible solutions to $P_i \equiv P_j \bmod p^e$ for any prime power p^e that exactly divides $|G|$, i.e., $p^{e+1} \not| |G|$. Note that uniformly distributed random values mod $|G|$ are also randomly and uniformly distributed mod p^e. More precisely, we consider the solutions to the following polynomials

$$F_{i,j}(x, y, c) := x[P_i(1, x, x^{-1}, y, xy, c) - P_j(1, x, x^{-1}, y, xy, c)] \equiv 0 \bmod p^e$$

or

$$G_{i,j}(x, y, c) := x[P_i(1, x, x^{-1}, y, c, xy) - P_j(1, x, x^{-1}, y, c, xy)] \equiv 0 \bmod p^e.$$

Here the polynomials F or G are obtained by multiplying both sides of the congruence $P_i \equiv P_j \bmod p^e$ with x and then reordering the resulting congruence.

Hence, we bound the probability that a random triple $(x, y, c) \in_{\mathcal{R}} \mathbb{Z}_{p^e}^* \times \mathbb{Z}_{p^e} \times \mathbb{Z}_{p^e}$ is a zero of the polynomials F or $G \bmod p^e$ (Note that the domain of the secret exponent x is $\mathbb{Z}_{|G|}^*$.)

To do this, we first bound the number of possible solutions (x, y, c) to F or $G \bmod p^e$ where (x, y, c) are randomly selected from $\mathbb{Z}_{p^e}^3$. The total degree of each of the polynomials F and G is at most 3. It follows from Lemma 2.10 that the probability of a random $(x, y, c) \in \mathbb{Z}_{p^e}^3$ to be a zero of F or $G \bmod p^e$ is at most $2(3/p) = 6/p$. There are p^{3e} tuples (x, y, c) in $\mathbb{Z}_{p^e}^3$. Hence, there are at most $p^{3e}6/p = 6p^{3e-1}$ zeros for F or $G \bmod p^e$. Further, there are $\varphi(p^e)p^{2e} = (p^e - p^{e-1})p^{2e}$ tuples (x, y, c) in $\mathbb{Z}_{p^e}^* \times \mathbb{Z}_{p^e} \times \mathbb{Z}_{p^e}$.

Hence, the probability that such a tuple is a zero of F or $G \bmod p^e$ is upper bounded by $6p^{3e-1}/(p^e - p^{e-1})p^{2e}$. It follows

$$\begin{aligned}
\mathbf{Prob}[E] &\leq \frac{(T+6)(T+5)}{2} \frac{6p^{3e-1}}{(p^e - p^{e-1})p^{2e}} \\
&= (T+6)(T+5) \frac{3p^{3e-1}}{(p-1)p^{3e-1}} \\
&\leq (T+6)(T+5) \frac{3}{p-1} \\
&\leq (T+6)(T+5) \frac{3}{p'-1}
\end{aligned}$$

If the complementary event $\bar{E}$ occurs, then $\mathcal{A}^\sigma$ cannot obtain any information about the bit b except pure guessing. Thus, the success probability of $\mathcal{A}^\sigma$ for correctly outputting b is

$$\begin{aligned}
\mathbf{Prob}[\mathcal{A}^\sigma(..) = b] &= \mathbf{Prob}[E] + \frac{1}{2}\mathbf{Prob}[\bar{E}] \\
&= \mathbf{Prob}[E] + \frac{1 - \mathbf{Prob}[E]}{2} \\
&= \frac{1}{2} + \frac{\mathbf{Prob}[E]}{2} \\
&\leq \frac{1}{2} + \frac{3(T+6)(T+5)}{2(p'-1)}.
\end{aligned}$$

■

2.6.2 DSE versus DDH

2.6.2.1 High Granular

Wolf (1999) shows the following two results on the relation between DSE and DDH: DSE can easily be reduced to DDH, however, as Theorem 2.11 shows, the converse doesn't hold.

Theorem 2.10 (Wolf 1999)

$$\epsilon\text{-DSE(c:}*\text{; g:h; f:o)} \xRightarrow{\alpha'=\alpha;\ t'=t+O(\log|G|)} \epsilon\text{-DDH(c:}*\text{; g:h; f:o)} \qquad \square$$

Proof. Given a DDH oracle $\mathcal{O}_{DDH}$ which breaks ϵ-DDH(c:$*$; g:h; f:$*$), one can construct an algorithm $\mathcal{A}^{\mathcal{O}_{DDH}}$ for breaking ϵ-DSE(c:$*$; g:h; f:o) as follows: $\mathcal{A}^{\mathcal{O}_{DDH}}$ randomizes its input tuple $((G,g),(g^x),(g^z))$ by choosing $r \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ and constructing the tuple

$$I = ((G,g),(g^X,g^Y),(g^Z))$$

where $g^X{:=}g^x$, $g^Y{:=}g^{x+r}$ and $g^Z{:=}g^z(g^x)^r$. The tuple I has the correct input distribution for $\mathcal{O}_{DDH}$ because (i) $g^Y := g^{x+r}$ is a random group element independent of g^X, and (ii) the last element g^Z is g^{XY} if and only if $g^z = g^{x^2}$, and it is a random group element otherwise.

Success probability: There is a single call to the DDH oracle with correctly distributed inputs. Thus, the resulting success probability is $\alpha_{DSE} = \alpha_{DDH}$, and since α_{DSE} is invariant, it follows that α_{DDH} is also invariant.

Efficiency: There is a single oracle call, and by applying the square and multiply method one requires $O(\log|G|)$ group operations for performing the exponentiations. ■

Remark 2.15. The reduction in Theorem 2.10 also holds for other variants of the assumption with respect to the success probabilities perfect, weak and strong.

Remark 2.16. In the proof of Theorem 2.10 we selected random elements from $\mathbb{Z}_{|G|}$ exploiting that the group order is known. While the group order might not always be publicly known, there is always a publicly known upper bound $B(|G|)$ on the group order as mentioned in Section 2.2 where Parameter 3 is discussed. If we now consider the two probability ensembles

$$X_k^* := \{g^{x^*} :: G \leftarrow SG(1^k) \ \wedge \ g \leftarrow Sg(G) \ \wedge \ x^* \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_{2^k B(|G|)}\}$$

and

$$X_k := \{g^x :: G \leftarrow SG(1^k) \ \wedge \ g \leftarrow Sg(G) \ \wedge \ x \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_{|G|}\},$$

we prove that they are statistically indistinguishable. First, observe that we compute in the exponents implicitly modulo $|G|$. Therefore, it is sufficient to consider the ensembles

$$Y_k^* := \{x^* \pmod{|G|} :: G \leftarrow SG(1^k) \ \wedge \ x^* \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_{2^k B(|G|)}\}$$

and

$$Y_k := \{x :: G \leftarrow SG(1^k) \ \wedge \ x \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_{|G|}\}.$$

Investigating their statistical difference , we can derive following inequalities:

$$\begin{aligned}
\Delta_{(Y^*,Y)}(k) &:= \sum_{y\in\mathbb{Z}_{|G|}} |\mathbf{Prob}[Y_k^* = y] - \mathbf{Prob}[Y_k = y]| \\
&= \sum_{y\in\mathbb{Z}_{|G|}} |\mathbf{Prob}[Y_k^* = y] - \frac{1}{|G|}| \\
&\leq \sum_{y\in\mathbb{Z}_{|G|}} (\mathsf{max}_{y\in\mathbb{Z}_{|G|}}(\mathbf{Prob}[Y_k^* = y]) - \mathsf{min}_{y\in\mathbb{Z}_{|G|}}(\mathbf{Prob}[Y_k^* = y])) \\
&= |G|\ (\mathsf{max}_{y\in\mathbb{Z}_G}(\mathbf{Prob}[Y_k^* = y]) - \mathsf{min}_{y\in\mathbb{Z}_G}(\mathbf{Prob}[Y_k^* = y])) \\
&= |G|\ (\frac{\lceil 2^k B(|G|)/|G|\rceil}{2^k B(|G|)} - \frac{\lfloor 2^k B(|G|)/|G|\rfloor}{2^k B(|G|)}) \\
&= \frac{|G|}{2^k B(|G|)} \\
&\leq \frac{1}{2^k}
\end{aligned}$$

Clearly, from this it follows that Y and Y^* (and indirectly X and X^*) are statistically indistinguishable. Given that the behavior of the oracle machine cannot significantly differ on input distributions which are statistically indistinguishable from the correct ones — otherwise we would have a computational and therefore also statistical distinguisher — it is sufficient to sample random exponents from $\mathbb{Z}_{2^k B(|G|)}$ to make the reduction work also for arbitrary group families.[39]

$\circ$

Remark 2.17. Following Remark 2.3, the result of Theorem 2.10 easily extends to the medium-granular variant. $\circ$

Next theorem states that a DSE oracle, even when perfect, is of no help in breaking DDH assumptions.

Theorem 2.11 (Wolf 1999)
true $\Longrightarrow$ 1-DDH$^\sigma$(c:$*$; g:h; f:nsprim; $\mathcal{O}_{\text{1-DSE(c:}*\text{; g:h; f:nsprim)}}$) $\square$

[39] A similar argument (but without proof) is given by Boneh (1998) for random self-reducing of DDH with unknown order. He proposes to sample from $\mathbb{Z}_{B(|G|)^2}$. However, as in virtually all practical cases $B(|G|)$ is considerably larger than 2^k this results in a much more expensive reduction. Let us consider following (common) example: The computation is done in subgroups of $\mathbb{Z}_p^*$ with prime order q and an obvious upper bound on the group order is p. For concreteness, let us use the group parameters suggested by Lenstra and Verheul (2001) for security parameter $k = 80$, i.e., p and q having approximately 1460 and 142 bits, respectively. While our method requires exponentiation with exponents of 1540 bits, Boneh's method would require exponentiation with exponents of 2920 bits, i.e., a huge difference!

Remark 2.18. More precisely, Wolf shows, the probability that any $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ can distinguish correct DDH inputs from incorrect ones is at most $\frac{(T+5)(T+4)}{2p'}$ where p' is the smallest prime factor of $|G|$ and T is an upper bound on the algorithm's runtime.

Remark 2.19. Theorem 2.11 holds also for other variants of the DDH assumption with respect to success probabilities weak, invariant and strong. ○

2.6.3 DIE versus DDH

2.6.3.1 High Granular

In the following, we prove that similar relations also hold between DDH and DIE. We first prove a reduction from DIE to DDH, and then show in Theorem 2.14 that the converse does not hold in generic model. This means a DIE oracle, even when perfect, is of no help in breaking DDH assumption.

Theorem 2.12

$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-DIE(c:}*\text{; g:h; f:nsprim)}$$
$$\xRightarrow{\alpha' \geq \frac{2\varphi(|G|)-1}{|G|}\alpha;\ t'=t+O(\log|G|)}$$
$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-DDH(c:}*\text{; g:h; f:nsprim)}$$
□

Proof. We prove the following statement: Given a DDH oracle $\mathcal{O}_{DDH}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-DDH(c:$*$; g:h; f:nsprim) with success probability $\alpha_{DDH}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{DDH}}$ which breaks the assumption $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-DIE(c:$*$; g:h; f:nsprim) with success probability $\alpha_{DIE}(k) \geq \frac{2\varphi(|G|)-1}{|G|}\alpha_{DDH}(k)$, using a single call to $\mathcal{O}_{CDH}$ and $O(\log|G|)$ group operations.

Suppose, we are given a DIE input tuple $((G,g),(g^x),(g^z))$ with $z = x^{-1}$ or $z = c$ where $c \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$. We transform the DIE input tuple to a DDH input tuple $((G,g),(g^X,g^Y),(g^Z))$ as follows: Set

$$g^X := g^{x+a_1}, \quad g^Y := g^{a_2 z + a_3}, \quad g^Z := g^{a_4^* x + a_5^* z + a_6^*}.$$

where $a_1, a_2, a_3 \in_{\mathcal{R}} \mathbb{Z}_{|G|}$, $a_4^* = a_3$, $a_5^* = a_2 a_1$ and $a_6^* = a_2 + a_1 a_3$. Here the superscript "$*$" indicates that the corresponding values are constructed.

If $z = x^{-1}$, we get a correct DDH input tuple $((G,g),(g^X,g^Y),(g^Z))$ because (i) X, Y are randomly and uniformly distributed over $\mathbb{Z}_{|G|}$ due to the randomization with $a_1, a_2, a_3 \in_{\mathcal{R}} \mathbb{Z}_{|G|}$, and (ii) the following holds

$$\begin{aligned} XY &= a_2xz + a_3x + a_2a_1z + a_1a_3 \\ &= a_2xx^{-1} + a_3x + a_2a_1z + a_1a_3 \\ &= a_3x + a_2a_1x^{-1} + a_2 + a_1a_3 \\ &= a_4^* + a_5^*x^{-1} + a_6^* \\ &= Z. \end{aligned}$$

The case $z = c$ is more involved. Here, we apply the result of Lemma 2.13 which we prove below: For group orders with no small prime factors, the (constructed) tuples (X, Y, Z) are statistically indistinguishable from tuples (X', Y', Z') chosen randomly from $\mathbb{Z}_{|G|}^3$, and therefore, also indistinguishable for the DDH oracle. Thus, also in this case we get a correct DDH input tuple.

Success probability: There is single call to $\mathcal{O}_{DDH}$ and, therefore, the resulting success probability is

$$\alpha_{DIE}(k) \geq (\frac{2\varphi(|G|)}{|G|} - 1)\alpha_{DDH}(k).$$

The factor $(\frac{2\varphi(|G|)}{|G|} - 1)$ comes from the fact that the input tuples to $\mathcal{O}_{DDH}$ are legal only if the tuples (X, Y, Z) are randomly and uniformly distributed over $\mathbb{Z}_{|G|}$ (see the proof of Lemma 2.13 for details).

In the following, we set $\gamma(k) := \frac{2\varphi(|G|)}{|G|} - 1$. Note that $|G|$ is a function of the security parameter k (see also Lemma 2.5). The type of resulting success probability $\alpha_{DIE}(k)$ depends on the type of $\alpha_{DDH}(k)$. The proofs are similar to those of Lemma 2.6. Nevertheless, for completeness, they are given below.

Perfect oracle ($\alpha_{DDH}(k) \not<_\infty 1$): The resulting success probability cannot be perfect because there is a non-zero error probability when querying the DDH oracle.

Weak oracle ($\alpha_{DDH}(k) \not<_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) weak: Since $|G|$ contains no small prime factors, it follows from Lemma 2.5 that $1 - \gamma(k) = 2(1 - \frac{\varphi(|G|)}{|G|}) <_\infty 1/\mathsf{poly}(k)$. Thus, we can write $\gamma(k) > 1 - 1/\mathsf{poly}(k)$, meaning that $\gamma(k)$ is non-negligible. Further, we have $\alpha_{DDH}(k) \not<_\infty 1/\mathsf{poly}(k)$. It follows $\gamma(k)\alpha_{DDH}(k) \not<_\infty 1/\mathsf{poly}(k)$ (see also Section 2.1.2) Finally, since $\alpha_{DIE}(k) \geq \gamma(k)\alpha_{DDH}(k)$, it follows $\alpha_{DIE}(k) \not<_\infty 1/\mathsf{poly}(k)$.

Invariant oracle ($\alpha_{DDH} \not<_\infty \epsilon_1$): The resulting success probability is (asymptotically) invariant: As shown in the weak case, we can write $\gamma(k) >_\infty 1 - 1/\mathsf{poly}(k)$. More precisely, for any $\epsilon' > 0$ there exists a k_0

such that for all $k > k_0$ we have $\gamma(k) >_\infty 1 - \epsilon'$. From $\alpha_{DDH}(k) \not<_\infty \epsilon_1$, it follows that for any k'_0 there exists a $k_1 > k'_0$ such that $\alpha_{DDH}(k_1) \geq \epsilon_1$. Thus, for any $k'_0 > k_0$ there exists a $k_1 > k'_0$ such that $\alpha_{DIE}(k_1) \geq \epsilon_2$ where $\epsilon_2 := (1 - \epsilon')\epsilon_1$. It follows $\alpha_{DIE}(k) \not<_\infty \epsilon_2$.

Strong oracle ($1 - \alpha_{DDH}(k) \not\geq_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) strong: As shown in the weak case, we can write $\gamma(k) >_\infty 1 - 1/\mathsf{poly}(k)$. Further, we have $1 - \alpha_{DDH}(k) \not\geq_\infty 1/\mathsf{poly}(k)$. It follows from Lemma 2.7 that $1 - \gamma(k)\alpha_{DDH}(k) \not\geq_\infty 1/\mathsf{poly}(k)$. Finally, since $\alpha_{DIE}(k) \geq \gamma(k)\alpha_{DDH}(k)$, it follows $1 - \alpha_{DIE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

■

Lemma 2.13 *Let SG be a group sampler of groups whose orders contain no small prime factors. Further, let V_k and V'_k be probability ensembles defined as*

$$\begin{array}{rcl} V_k & := & \{(X,Y,Z) :: G \leftarrow SG(1^k) \ \wedge \\ & & (x,z) \in_{\mathcal{R}} (\mathbb{Z}^*_{|G|})^2 \wedge \ (a_1,a_2,a_3) \in_{\mathcal{R}} \mathbb{Z}^3_{|G|} \wedge \\ & & a^*_4 = a_3 \ \wedge \ a^*_5 = a_2 a_1 \ \wedge \ a^*_6 = a_2 + a_1 a_3 \ \wedge \\ & & X := x + a_1 \ \wedge \ Y := a_2 z + a_3; \wedge \ Z := a^*_4 x + a^*_5 z + a^*_6\}, \\ V'_k & := & \{(X',Y',Z') :: G \leftarrow SG(1^k) \ \wedge \ (X',Y',Z') \in_{\mathcal{R}} \mathbb{Z}^3_{|G|}\}. \end{array}$$

Then V_k and V'_k are statistically indistinguishable. □

Proof. According to the definition of statistical indistinguishability (see Section 2.1.4) we have to prove that the statistical difference $\Delta_{(V,V')}(k)$ is negligible in security parameter k (here, for group orders $|G|$ with no small prime factors)

Clearly, $\mathbf{Prob}[V'_k = v] = 1/|G|^3$ holds for all $v \in \mathbb{Z}^3_{|G|}$ by definition. Next, we partition $\mathbb{Z}^3_{|G|}$ in two disjoint sets D and its complement $\bar{D} = \mathbb{Z}^3_{|G|} \backslash D$ such that $\mathbf{Prob}[V_k = v] \geq 1/|G|^3$ for all $v \in D$ and $\mathbf{Prob}[V_k = v] < 1/|G|^3$ for all $v \in \bar{D}$. Then we can write

$$\begin{array}{rcl} \Delta_{(V,V')}(k) & := & \displaystyle\sum_{v \in \mathbb{Z}^3_{|G|}} |\mathbf{Prob}[V_k = v] - \mathbf{Prob}[V'_k = v]| \\ & = & \displaystyle\sum_{v \in \mathbb{Z}^3_{|G|}} |\mathbf{Prob}[V_k = v] - 1/|G|^3| \\ & = & \displaystyle\sum_{v \in D} (\mathbf{Prob}[V_k = v] - 1/|G|^3) + \\ & & \qquad \displaystyle\sum_{v \in \bar{D}} (1/|G|^3 - \mathbf{Prob}[V_k = v]). \end{array}$$

It follows

$$\begin{aligned}\Delta_{(V,V')}(k) &\le \sum_{v\in D}\big(\mathbf{Prob}[V_k=v]-1/|G|^3\big)+\\ &\qquad\sum_{v\in\bar D}\big(1/|G|^3-\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)\\ &= \sum_{v\in D}\mathbf{Prob}[V_k=v]-|D|/|G|^3+\\ &\qquad|\bar D|\big(1/|G|^3-\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big).\end{aligned}$$

We exploit the relations $\sum_{v\in D}\mathbf{Prob}[V_k=v]=1-\sum_{v\in\bar D}\mathbf{Prob}[V_k=v]$ and $|D|=|G|^3-|\bar D|$, and substitute them in the above inequality:

$$\begin{aligned}\Delta_{(V,V')}(k) &\le 1-\sum_{v\in\bar D}\mathbf{Prob}[V_k=v]-\frac{|G|^3-|\bar D|}{|G|^3}+\\ &\qquad|\bar D|\big(1/|G|^3-\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)\\ &\le 1-\sum_{v\in\bar D}\big(\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)-1+|\bar D|/|G|^3+\\ &\qquad|\bar D|\big(1/|G|^3-\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)\\ &= -|\bar D|\big(\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)+|\bar D|/|G|^3+\\ &\qquad|\bar D|\big(1/|G|^3-\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)\\ &= 2|\bar D|/|G|^3-2|\bar D|\big(\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)\\ &= 2|\bar D|\big(1/|G|^3-\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)\\ &\le 2|G|^3\big(1/|G|^3-\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\big)\end{aligned}$$

where for the last step we used the fact $|\bar D|\le|G|^3$. Next, we determine a lower bound for $\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])$. For this, we first consider the probability $\mathbf{Prob}[V_k=v]$ together with the event $E_k:=E(V_k):(1-xz)\in\mathbb{Z}^*_{|G|}$. It follows

$$\mathbf{Prob}[V_k=v]=\mathbf{Prob}[V_k=v\wedge E_k]+\mathbf{Prob}[V_k=v\wedge\bar E_k]$$

where $\bar E_k$ is the complement of E_k. Thus, we can write

$$\begin{aligned}\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v]) &\ge \mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v\wedge E_k])\\ &\qquad+\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v\wedge\bar E_k]).\end{aligned}$$

Since $\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v\wedge\bar E_k])\ge 0$, it follows

$$\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v])\ge\mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k=v\wedge E_k]).$$

Further, we have

$$\begin{aligned}\mathbf{Prob}[V_k = v \wedge E_k] &= \mathbf{Prob}[V_k = v|E_k]\mathbf{Prob}[E_k] \\ &\geq \frac{1}{|G|^3}\frac{\varphi(|G|) - (|G| - \varphi(|G|))}{|G|} \\ &\geq \frac{1}{|G|^3}\Big(\frac{2\varphi(|G|)}{|G|} - 1\Big).\end{aligned}$$

To see this, we consider the probability terms separately:

(i) For all v it holds $\mathbf{Prob}[V_k = v|E_k] = 1/|G|^3$. This is because, if $E(V_k)$ (i.e., $1 - xz \in \mathbb{Z}^*_{|G|}$) holds then the tuples (X, Y, Z) are uniformly distributed over $\mathbb{Z}^3_{|G|}$. One can see this as follows: For any $(x, z) \in_{\mathcal{R}} \mathbb{Z}^*_{|G|} \times \mathbb{Z}^*_{|G|}$ with $z \neq x^{-1}$ and for any $(X, Y, Z) \in_{\mathcal{R}} \mathbb{Z}^3_{|G|}$, there exist exactly one tuple $(a_1, a_2, a_3) \in \mathbb{Z}^3_{|G|}$ such that following equations hold

$$X := x + a_1, \quad Y := a_2 z + a_3, \quad Z := a_4^* x + a_5^* z + a_6^*$$

where $a_4^* = a_3$, $a_5^* = a_2 a_1$, $a_6^* = a_2 + a_1 a_3$. We compute a_1, a_3 from the first equations and set them in the third and obtain:

$$Z = XY + a_2 - a_2 xz, \; a_2(1 - xz) = Z - XY.$$

The last equation has a solution for $a_2 = (Z - XY)(1 - xz)^{-1}$ if $\gcd(1 - xz, |G|) = 1$, i.e., $1 - xz \in \mathbb{Z}^*_{|G|}$.[40] Having a_2 computed, we can obtain the other values by computing $a_1 = X - x$ and $a_3 = Y - a_2 z$.

(ii) $\mathbf{Prob}[E_k] \geq \frac{2\varphi(|G|)}{|G|} - 1$: As x, z are elements from $\mathbb{Z}^*_{|G|}$, there are at most $\varphi(|G|)$ possible values for $1 - xz$ which may or may not be relatively prime to $|G|$. In the worst case, at most $|G| - \varphi(|G|)$ of them are not relatively prime to $|G|$. Thus, in the worst case, the number of possible values for $1 - xz$ relatively prime to $|G|$ is still $\varphi(|G|) - (|G| - \varphi(|G|)) = 2\varphi(|G|) - |G|$. Therefore, we can write $\mathbf{Prob}[E_k] \geq \frac{2\varphi(|G|) - |G|}{|G|} = \frac{2\varphi(|G|)}{|G|} - 1$.

From the above explanation follows

$$\mathsf{min}_{v \in \mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k = v \wedge E_k]) = \frac{1}{|G|^3}\Big(\frac{2\varphi(|G|)}{|G|} - 1\Big).$$

[40]The other solutions are all congruence modulo $|G|$. Note that if $d = \gcd((1 - xz), Z - XY)$ the equation has exactly d solutions for a given (x, z). In this case, we have collisions, i.e., there exist d different (a_1, a_2, a_3) tuples and that (X, Y, Z) are in general not uniformly distributed over $\mathbb{Z}^3_{|G|}$.

Now, we return to the statistical difference of the distributions. Substituting the above results, we obtain

$$\begin{aligned}
\Delta_{(V,V')}(k) &\leq 2|G|^3\big(1/|G|^3 - \mathsf{min}_{v\in\mathbb{Z}^3_{|G|}}(\mathbf{Prob}[V_k = v])\big) \\
&\leq 2|G|^3\big(1/|G|^3 - \frac{1}{|G|^3}(\frac{2\varphi(|G|)}{|G|} - 1)\big) \\
&\leq \frac{2|G|^3}{|G|^3}\big(2 - \frac{2\varphi(|G|)}{|G|}\big) \\
&\leq 4\big(1 - \frac{\varphi(|G|)}{|G|}\big)
\end{aligned}$$

For $|G|$ with no small prime factors, it follows from Lemma 2.5 that $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/\mathsf{poly}(k)$ and consequently $\Delta_{(V,V')}(k) <_\infty 1/\mathsf{poly}(k)$. This completes the proof. ■

Remark 2.20. The above reduction does not hold for perfect success probability because of the non-zero error probability introduced to the computation. However, for groups of prime order the reduction also holds for perfect success probability as the only special case $x = 0$ can be explicitly handled, i.e., one can easily test whether $g^0 = 1$ is the input.

Remark 2.21. The reduction in Theorem 2.12 is proven for group orders with no small prime factors. However, it also holds for all other group orders, provided the group order is known. Knowing the group order, one can factor out the small prime factors by well-known factoring algorithms, and then easily solve the decisional problems DIE and DDH (see also Remark 2.10). Thus, we have the following theorem.

Theorem 2.13

$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-DIE(c:∗; g:h; f:∗,o)

$$\Longrightarrow$$

$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-DDH(c:∗; g:h; f:∗,o)

□

○

The following theorem proves (in generic model) that a DIE oracle, even when perfect, is of no help in breaking DDH assumptions.

Theorem 2.14

true $\Longrightarrow$ ∗-DDH$^\sigma$(c:∗; g:h; f:nsprim; $\mathcal{O}_{\text{1-DIE(c:∗; g:h; f:nsprim)}}$) □

Proof. Theorem 2.14 immediately follows from Lemma 2.14 and Remark 2.14. ■

Lemma 2.14 *Let G be a cyclic group and g a corresponding generator, let p' be the smallest prime factor of $|G|$. Let $\mathcal{O}_{DIE}$ be a given oracle which solves DIE tuples in G and let $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ be any generic algorithm for groups G with maximum run time T and oracle access to $\mathcal{O}_{DIE}$. Then the following always holds:*

$(|\, \textbf{\textit{Prob}}[\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}(\mathcal{C}, (G, g), w_b, w_{\bar{b}}) = b ::$
$\quad b \xleftarrow{\mathcal{R}} \{0, 1\};\ \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U};$
$\quad PI \leftarrow SPI_{DH}((G, g));\ PI_{\mathcal{R}} \leftarrow SPI_{PI^{\mathcal{P}}}(PI^{\mathcal{P}});$
$\quad w_b \leftarrow (PI^{publ}, PI^{sol});$
$\quad w_{\bar{b}} \leftarrow (PI^{publ}, {PI_{\mathcal{R}}}^{sol})$
$\quad]-1/2\ | \cdot 2)\ \leq \frac{2(T+5)(T+4)}{p'}$
□

Proof. Assume, we are given the encodings $\sigma(1)$, $\sigma(x)$, $\sigma(y)$ and $\{\sigma(xy), \sigma(c)\}$. After T_1 computation steps, $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ can compute at most $T_1 + 5$ distinct linear combinations $P_i(x, y, xy, c)$ of $1, x, y, xy$ and c, i.e., it obtains

$$\sigma(P_i(1, x, y, xy, c)) = \sigma(a_{i1} + a_{i2}x + a_{i3}y + a_{i4}xy + a_{i5}c)$$

where a_{ij} are constant coefficients. Furthermore, it is not known to $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ which one of the values $\{a_{i4}, a_{i5}\}$ is the coefficient of xy and which one corresponds to c. Further, assume that $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ makes T_2 calls to $\mathcal{O}_{DIE}$.

$\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ may be able to distinguish $\sigma(xy)$ and $\sigma(c)$ by obtaining information from either of the following events:

E_a: $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ finds relations (collision) between two distinct linear combination (P_i, P_j) with $i \neq j$. This means, it obtains either $\sigma(P_i(1, x, y, xy, c)) = \sigma(P_j(1, x, y, xy, c))$ or $\sigma(P_i(1, x, y, c, xy)) = \sigma(P_j(1, x, y, c, xy))$.

E_b: $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ gets at least one positive answer from $\mathcal{O}_{DIE}$, i.e., it obtains either $\sigma(P_i(1, x, y, xy, c)) = \sigma((P_j(1, x, y, xy, c))^{-1})$ or $\sigma(P_i(1, x, y, c, xy)) = \sigma((P_j(1, x, y, c, xy))^{-1})$.

We compute an upper bound for the probability that either of the events E_a and E_b occurs.

Case E_a: In this case we have $P_i(1, x, y, xy, c) \equiv P_j(1, x, y, xy, c) \bmod |G|$ or $P_i(1, x, y, c, xy) \equiv P_j(1, x, y, c, xy) \bmod |G|$. There are $\binom{T_1+5}{2} = \frac{(T_1+5)(T_1+4)}{2}$ distinct pairs of polynomials (P_i, P_j). For each such a pair (i, j) we bound the number of solutions to $P_i \equiv P_j \bmod p^e$ for any prime power p^e that exactly divides $|G|$, i.e., $p^{e+1} \not| |G|$. Note that uniformly distributed random values $\bmod |G|$ are also randomly and uniformly distributed $\bmod\, p^e$. More precisely, we consider the solutions to the following polynomials

$$F_{i,j}(x,y,c) := P_i(1,x,y,xy,c) - P_j(x,y,xy,c) \equiv 0 \bmod p^e$$

or

$$G_{i,j}(x,y,c) := P_i(1,x,y,c,xy) - P_j(x,y,c,xy) \equiv 0 \bmod p^e.$$

Each of the polynomials F and G has the total degree 2. It follows from Lemma 2.10 that the probability for a random $(x,y,c) \in \mathbb{Z}_{p^e}^3$ to be a zero of $F \bmod p^e$ or $G \bmod p^e$ is at most $2(2/p) = 4/p$. Thus, we have

$$\begin{aligned}\mathbf{Prob}[E_a] &\leq \frac{(T_1+5)(T_1+4)}{2}\frac{4}{p} \\ &\leq \frac{2(T_1+5)(T_1+4)}{p'}.\end{aligned}$$

Case E_b: In this case we have $P_i \equiv P_j{}^{-1} \bmod |G|$. However, it is not possible to derive this relation between the polynomials P_i and P_j but only between their evaluations at the points (x,y,c), i.e., when $P_i(1,x,y,c,xy) \equiv P_j(x,y,c,xy)^{-1} \bmod |G|$ (with $P_j(x,y,c,xy) \not\equiv 0 \bmod |G|$).

Similar to our approach for E_a, for each such a pair (i,j), we can bound the number of solutions to $P_i(x,y,xy,c) - P_j(x,y,xy,c)^{-1} \equiv 0 \bmod p^e$ or $P_i(x,y,c,xy) - P_j(x,y,c,xy)^{-1} \equiv 0 \bmod p^e$ for any prime power p^e that exactly divides $|G|$. More precisely, we consider the solutions to the following polynomials

$$H_{i,j}(x,y,c) := P_i(1,x,y,xy,c)P_j(x,y,xy,c) - 1 \equiv 0 \bmod p^e$$

or

$$I_{i,j}(x,y,c) := P_i(1,x,y,c,xy)P_j(x,y,c,xy) - 1 \equiv 0 \bmod p^e.$$

Here, the polynomial $H(x,y,c)$ is obtained by multiplying both sides of the equation $P_i(1,x,y,xy,c) - P_j(1,x,y,xy,c)^{-1} \equiv 0 \bmod p^e$ with $P_j(1,x,y,xy,c)$. Similarly, we obtain $I(x,y,c)$.

Hence, we bound the probability that a random triple $(x,y,c) \in \mathbb{Z}_{p^e}^3$ is a zero of the polynomials I or H:[41] The total degree of each of the polynomials H and I is at most 4. It follows from Lemma 2.10 that the probability

[41]Note that DIE oracle is guaranteed to answer correctly only to the legal inputs, i.e., when the secret exponents are elements from $\mathbb{Z}_{|G|}^*$. Thus, the answer of DIE oracle is correct if $P_i(x,y,xy,c)$ and $P_j(x,y,xy,c)$ are elements of $\mathbb{Z}_{|G|}^*$. As the DIE oracle can be evil on illegal inputs, the adversary can obtain fewer information, if any, than in the case where the inputs are legal. Hence, to be on the safe side, we give the generic adversary the advantage that all values $P_i(x,y,xy,c)$ and $P_j(x,y,xy,c)$ are legal inputs to the DIE oracle.

of a randomly chosen $(x, y, c) \in_{\mathcal{R}} \mathbb{Z}^3_{p^e}$ to be a zero of H or I is at most $2(4/p) = 8/p$. It follows

$$\mathbf{Prob}[E_b] \leq T_2 \frac{8}{p} \leq T_2 \frac{8}{p'}.$$

In total we have

$$\begin{aligned}
\mathbf{Prob}[E] &\leq \mathbf{Prob}[E_a] + \mathbf{Prob}[E_b] \\
&\leq \frac{2(T_1+5)(T_1+4)}{p'} + \frac{8T_2}{p'} \\
&\leq \frac{2(T+5)(T+4)}{p'}
\end{aligned}$$

where $T_1 + T_2 \leq T$. If the complementary event $\bar{E}$ occurs, then $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ cannot obtain any information about the bit b except by pure guessing. Thus, the success probability of $\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}$ for correctly outputting b is

$$\begin{aligned}
\mathbf{Prob}[\mathcal{A}^{\sigma,\mathcal{O}_{DIE}}(..) = b] &= \mathbf{Prob}[E] + \frac{1}{2}\mathbf{Prob}[\bar{E}] \\
&= \mathbf{Prob}[E] + \frac{1 - \mathbf{Prob}[E]}{2} \\
&= \frac{1}{2} + \frac{\mathbf{Prob}[E]}{2} \\
&\leq \frac{1}{2} + \frac{(T+5)(T+4)}{p'}.
\end{aligned}$$

■

2.6.4 DSE versus DIE

2.6.4.1 High Granular

In the next theorem, we prove that an oracle breaking 1-DSE(c:∗; g:h; f:∗) is of no help in breaking ∗-DIE$^\sigma$(c:∗; g:h; f:∗).

Theorem 2.15
true $\Longrightarrow$ ∗-DIE$^\sigma$(c:∗; g:h; f:nsprim; $\mathcal{O}_{\text{1-DSE(c:∗; g:h; f:nsprim)}}$) □

Proof. Similar to the proofs of Theorem 2.8 and 2.14, we define a Lemma which associates the minimal generic complexity of solving DIE directly to the smallest prime factor of the order of the underlying group G. Theorem 2.15 immediately follows from Lemma 2.15 and Remark 2.14. ■

Lemma 2.15 *Let G be a cyclic group and g a corresponding generator, let p' be the smallest prime factor of $|G|$. Let $\mathcal{O}_{DSE}$ be a given oracle solving DSE tuples in G and let $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ be any generic algorithm for groups G with maximum run time T and oracle access to $\mathcal{O}_{DSE}$. Then the following always holds:*

$$\begin{aligned}
&(\mid \boldsymbol{Prob}[\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}(\mathcal{C},(G,g),w_b,w_{\bar{b}}) = b :: \\
&\quad b \xleftarrow{\mathcal{R}} \{0,1\};\ \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}; \\
&\quad PI \leftarrow SPI_{IE}((G,g));\ PI_{\mathcal{R}} \leftarrow SPI_{PI^{\mathcal{P}}}(PI^{\mathcal{P}}); \\
&\quad w_b \leftarrow (PI^{publ}, PI^{sol}); \\
&\quad w_{\bar{b}} \leftarrow (PI^{publ}, {PI_{\mathcal{R}}}^{sol}) \\
&\ ]-1/2 \mid \cdot 2)\ \le \frac{2(T+4)(T+3)}{p'-2}
\end{aligned}$$
□

Proof. Assume, we are given the encodings $\sigma(1)$, $\sigma(x)$ and $\{\sigma(x^{-1}), \sigma(c)\}$ where $x \in \mathbb{Z}^*_{|G|}$. After T_1 computation steps the algorithm $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ can compute at most $T_1 + 4$ distinct linear combinations P_i of the elements $1, x, x^{-1}$, and c, i.e., it obtains

$$\sigma(P_i(1,x,x^{-1},c)) = \sigma(a_{i1} + a_{i2}x + a_{i3}x^{-1} + a_{i4}c),$$

where a_{ij} are constant coefficients. Furthermore, it is not a-priori known to $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ which of the values in $\{a_{i3}, a_{i4}\}$ is the coefficient for x^{-1} and which one corresponds to c. Further, assume that $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ makes T_2 calls to $\mathcal{O}_{DSE}$.

$\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ may be able to distinguish $\sigma(x^{-1})$ and $\sigma(c)$ by obtaining information from either of the following events:

E_a: $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ finds a relation (collision) between two distinct linear equations (P_i, P_j) with $i \neq j$. This means it obtains $\sigma(P_i(1,x,x^{-1},c)) = \sigma(P_j(1,x,x^{-1},c))$ or $\sigma(P_i(1,x,c,x^{-1})) = \sigma(P_j(1,x,c,x^{-1}))$.

E_b: $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ gets at least one positive answer from $\mathcal{O}_{DSE}$ with $i \neq j$, i.e., it obtains $\sigma(P_i(1,x,x^{-1},c)) = \sigma((P_j(1,x,x^{-1},c))^2)$ or $\sigma(P_i(1,x,c,x^{-1})) = \sigma((P_j(1,x,c,x^{-1}))^2)$.

We compute an upper bound for the probability that either of these events occurs.

Case E_a: In this case we have $P_i(1,x,x^{-1},c) \equiv P_j(1,x,x^{-1},c) \bmod |G|$ or $P_i(1,x,c,x^{-1}) \equiv P_j(1,x,c,x^{-1}) \bmod |G|$. There are $\binom{T_1+4}{2} = \frac{(T_1+4)(T_1+3)}{2}$ distinct polynomial pairs (P_i, P_j). For each such a pair (i,j), we can bound the number of possible solutions to $P_i \equiv P_j \bmod p^e$ for any prime power p^e that exactly divides $|G|$, i.e., $p^{e+1} \not| |G|$. Note that uniformly distributed random values mod$|G|$ are also randomly and uniformly distributed mod p^e. More precisely, we consider the solutions to the following polynomials

$$F_{i,j}(x,c) := x[P_i(1,x,x^{-1},c) - P_j(1,x,x^{-1},c)] \equiv 0 \bmod p^e$$

or

$$G_{i,j}(x,c) := x[P_i(1,x,c,x^{-1}) - P_j(1,x,c,x^{-1})] \equiv 0 \bmod p^e.$$

Here, F and G are obtained by multiplying both sides of the congruence $P_i \equiv P_j \bmod p^e$ with x and then reordering the resulting congruence.

Hence, we bound the probability that a random tuple $(x,c) \in \mathbb{Z}^*_{p^e} \times \mathbb{Z}^*_{p^e}$ is a zero of the polynomials F or $G \bmod p^e$ (Note that $\mathbb{Z}^*_{|G|}$ is the domain of the secret exponents of DIE inputs.)

To do this, we first bound the number of solutions to F or $G \bmod p^e$ where (x,c) are randomly selected from $\mathbb{Z}^2_{p^e}$: The total degree of each of the polynomials F and G is two. It follows from Lemma 2.10 that the probability for a random tuple $(x,c) \in_{\mathcal{R}} \mathbb{Z}^2_{|G|}$ to be a zero of F or $G \bmod p^e$ is at most $2(2/p) = 4/p$. There are p^{2e} tuples (x,c) in $\mathbb{Z}^2_{p^e}$. Thus, there are at most $p^{2e}4/p = 4p^{2e-1}$ zeros for either F or $G \bmod p^e$. Further, there are $\big(\varphi(p^e)\big)^2 = (p^e - p^{e-1})^2$ tuples (x,c) in $\mathbb{Z}^*_{p^e} \times \mathbb{Z}^*_{p^e}$.

Hence, the probability that such a tuple is a zero of F or $G \bmod p^e$ is upper bounded by $4p^{2e-1}/(p^e - p^{e-1})^2$. It follows

$$\begin{aligned}
\mathbf{Prob}[E_a] &\leq \frac{(T_1+4)(T_1+3)}{2}\frac{4p^{2e-1}}{(p^e-p^{e-1})^2} \\
&= \frac{(T_1+4)(T_1+3)}{2}\frac{4p^{2e-1}}{p^{2e}+p^{2e-2}-2p^{2e-1}} \\
&= (T_1+4)(T_1+3)\frac{2p}{p^2-2p+1} \\
&\leq (T_1+4)(T_1+3)\frac{2}{p-2} \\
&\leq (T_1+4)(T_1+3)\frac{2}{p'-2}.
\end{aligned}$$

Case E_b: In this case we have $P_i \equiv {P_j}^2 \bmod |G|$. However, it is not possible to derive this relation between the polynomials P_i and P_j but only between their evaluations at the points (x,c), i.e., when $P_i(1,x,x^{-1},c) \equiv P_j(1,x,x^{-1},c)^2 \bmod |G|$. Similar to the case E_a, for each pair (i,j), $i \neq j$, we can bound the number of possible solutions to $P_i(1,x,x^{-1},c) \equiv (P_j(1,x,x^{-1},c))^2 \bmod p^e$ or $P_i(1,x,c,x^{-1}) \equiv (P_j(1,x,c,x^{-1}))^2 \bmod p^e$ for any prime power p^e that exactly divides $|G|$. More precisely, we consider the solutions to the following polynomials

$$H_{i,j}(x,c) := x^2[P_i(1,x,x^{-1},c) - (P_j(1,x,x^{-1},c))^2] \equiv 0 \bmod p^e$$

or

$$I_{i,j}(x,c) := x^2[P_i(1,x,c,x^{-1}) - (P_j(1,x,c,x^{-1}))^2] \equiv 0 \bmod p^e.$$

Here, we obtain the polynomial $H(x,c)$ by multiplying both sides of the congruence $P_i(1,x,x^{-1},c) \equiv P_j(1,x,x^{-1},c) \bmod p^e$ with x^2 and then reordering the resulting congruence. Similarly, we obtain $I(x,c)$.

Hence, we bound the probability that a random tuple $(x,c) \in_{\mathcal{R}} \mathbb{Z}_{p^e}^* \times \mathbb{Z}_{p^e}^*$ is a zero of the polynomials H or $I \bmod p^e$ (as for E_a): The total degree of each of the polynomials H and I is at most 4. It follows from Lemma 2.10 that the probability for a random tuple $(x,c) \in_{\mathcal{R}} \mathbb{Z}_{|G|}^2$ to be a zero of H or $I \bmod p^e$ is at most $2(4/p) = 8/p$. Thus, there are at most $p^{2e}8/p = 8p^{2e-1}$ zeros for either H or $I \bmod p^e$. Further, there are $\big(\varphi(p^e)\big)^2 = (p^e - p^{e-1})^2$ tuples (x,c) in $\mathbb{Z}_{p^e}^* \times \mathbb{Z}_{p^e}^*$.

Hence, the probability that $(x,c) \in \mathbb{Z}_{p^e}^* \times \mathbb{Z}_{p^e}^*$ is a zero of H or $I \bmod p^e$ is upper bounded by $8p^{2e-1}/(p^e - p^{e-1})^2$. It follows

$$\begin{aligned}
\mathbf{Prob}[E_b] &\le \frac{8T_2p^{2e-1}}{(p^e-p^{e-1})^2} = \frac{8T_2p^{2e-1}}{p^{2e}+p^{2e-2}-2p^{2e-1}} \\
&= \frac{8T_2p}{p^2-2p+1} \\
&\le \frac{8T_2p}{p^2-2p} = \frac{8T_2}{p-2} \\
&\le \frac{8T_2}{p'-2}.
\end{aligned}$$

In total we have

$$\begin{aligned}
\mathbf{Prob}[E] &\le \mathbf{Prob}[E_a] + \mathbf{Prob}[E_b] \\
&= \frac{2(T_1+4)(T_1+3)}{p'-2} + \frac{8T_2}{p'-2} \\
&\le \frac{2(T+4)(T+3)}{p'-2}
\end{aligned}$$

with $T_1 + T_2 \le T$. If the complementary event $\bar{E}$ occurs, then $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ cannot obtain any information about the bit b except by pure guessing. Thus, the success probability of $\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}$ for correctly outputting b is

$$\begin{aligned}
\mathbf{Prob}[\mathcal{A}^{\sigma,\mathcal{O}_{DSE}}(..) = b] &= \mathbf{Prob}[E] + \frac{1}{2}\mathbf{Prob}[\bar{E}] \\
&= \mathbf{Prob}[E] + \frac{1-\mathbf{Prob}[E]}{2} \\
&= \frac{1}{2} + \frac{\mathbf{Prob}[E]}{2} \\
&\le \frac{1}{2} + \frac{(T+4)(T+3)}{p'-2}.
\end{aligned}$$

■

2.6.4.2 Medium Granular

In sharp contrast to the above mentioned high-granular case, we prove in the following theorem that these assumptions are equivalent for their medium-granular version (other parameters remain unchanged).

Theorem 2.16

$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-DSE}(\text{c:}*;\text{g:m};\text{f:nsprim}) \xRightarrow{\alpha'\geq\left(\frac{\varphi(|G|)}{|G|}\right)^2\alpha;\ t'=t} \{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-DIE}(\text{c:}*;\text{g:m};\text{f:nsprim})$$

$$*\text{-DSE}(\text{c:}*;\text{g:m};\text{f:nsprim}) \xLeftarrow{\alpha'\geq\alpha-\left(1-\left(\frac{\varphi(|G|)}{|G|}\right)^2\right);\ t'=t} *\text{-DIE}(\text{c:}*;\text{g:m};\text{f:nsprim})$$

□

Proof. We prove the following statements:

(a) Given a DIE oracle $\mathcal{O}_{DIE}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-DIE(c:$*$; g:m; f:nsprim) with success probability $\alpha_{DIE}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{DIE}}$ which breaks $\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-DSE(c:$*$; g:m; f:nsprim) with success probability $\alpha_{DSE}(k) \geq \left(\frac{\varphi(|G|)}{|G|}\right)^2\alpha_{DIE}(k)$, using a single oracle call.

(b) Given an oracle $\mathcal{O}_{DSE}$ which breaks $*$-DSE(c:$*$; g:m; f:nsprim) with success probability $\alpha_{DSE}(k)$, there exists an algorithm $\mathcal{A}^{\mathcal{O}_{DSE}}$ which breaks $*$-DIE(c:$*$; g:m; f:nsprim) with success probability $\alpha_{DIE}(k) \geq \alpha_{DSE}(k) - \left(1-\left(\frac{\varphi(|G|)}{|G|}\right)^2\right)$, using a single oracle call.

Case (a): Assume, we are given a DSE input tuple $((G,g),(g^x),(g^z))$ where z is either x^2 or a random element $c \in_{\mathcal{R}} \mathbb{Z}_{|G|}$. Set $h := g^x$, and pass $((G,h),(h^t),(h^{tz}))$ to $\mathcal{O}_{DIE}$. Here, we used the relations $g = h^t$ and $g^z = h^{tz}$ where we implicitly assumed $t = x^{-1}$. This holds only if $x \in \mathbb{Z}^*_{|G|}$ which occurs with probability $\frac{\varphi(|G|)}{|G|}$ (Note that DIE oracle is not guaranteed to answer correctly on illegal inputs, i.e., inputs with secret exponents from $\mathbb{Z}_{|G|} \setminus \mathbb{Z}^*_{|G|}$.)

If $z = x^2$ then the tuple $((G,h),(h^t),(h^{xt}))$ is with probability $\frac{\varphi(|G|)}{|G|}$ a legal DIE input tuple, since we have $x = t^{-1}$, $z = t^{-2}$, i.e., the input tuple has the form $((G,h),(h^t),(h^{t^{-1}}))$.

If $z \neq x^2$ then the tuple $((G,h),(h^t),(h^{tz}))$ is a legal input tuple for DIE oracle with probability $\left(\frac{\varphi(|G|)}{|G|}\right)^2$. This holds because (i) h is a generator with probability $\frac{\varphi(|G|)}{|G|}$ (and thus h^t is a legal public part of the DIE input tuple), and (ii) h^{zt} is a legal random part of a DIE input tuple only if $z \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$ which is true with probability $\frac{\varphi(|G|)}{|G|}$. Since these events are independent, the probability for both to occur is $\big(\frac{\varphi(|G|)}{|G|}\big)^2$.

Success probability: We have $\alpha_{DSE}(k) \geq \alpha_{DIE}(k)\big(\frac{\varphi(|G|)}{|G|}\big)$ for the correct case (i.e., $z = x^2$) and $\alpha_{DSE}(k) \geq \alpha_{DIE}(k)\big(\frac{\varphi(|G|)}{|G|}\big)^2$ for the random case (i.e., $z \neq x^2$). Hence, for the resulting success probability the following holds

$$\alpha_{DSE}(k) \geq \big(\frac{\varphi(|G|)}{|G|}\big)^2 \alpha_{DIE}(k).$$

In the following we set $\gamma(k) := \big(\frac{\varphi(|G|)}{|G|}\big)^2$ (Note that $|G|$ is a function of the security parameter k, see also Lemma 2.5). Depending on the success probability of DIE oracle we have the following cases:

Perfect oracle ($\alpha_{DIE}(k) \not<_\infty 1$): The resulting success probability cannot be perfect because there is a non-zero error probability when querying the DIE oracle.

Weak oracle ($\alpha_{DIE}(k) \not<_\infty 1/\mathsf{poly}(k)$): The resulting success probability is also weak: This holds since $\frac{\varphi(|G|)}{|G|}$ and consequently $\gamma(k)$ is always non-negligible, and its multiplication with a not negligible function results in a not negligible function. Thus, we can write $\gamma(k)\alpha_{DIE}(k) \not<_\infty 1/\mathsf{poly}(k)$. Since $\alpha_{DSE}(k) \geq \gamma(k)\alpha_{DIE}(k)$, it follows $\alpha_{DSE}(k) \not<_\infty 1/\mathsf{poly}(k)$.

Invariant oracle ($\alpha_{DIE} \not<_\infty \epsilon_1$): The resulting success probability is (asymptotically) invariant: Since $|G|$ contains no small prime factors, it follows from Lemma 2.5 that $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/\mathsf{poly}(k)$. More precisely, for any $\epsilon' > 0$ there exists a k_0 such that for all $k > k_0$, $\frac{\varphi(|G|)}{|G|} > 1 - \epsilon'$. It follows that for all $k > k_0$, $\gamma(k) > (1-\epsilon')^2 = 1 - \epsilon''$ where $\epsilon'' := 2\epsilon' - \epsilon'^2$. Since $\alpha_{DIE}(k) \not<_\infty \epsilon_1$, for each k_0' there exists a $k_1 > k_0'$ such that $\alpha_{DIE}(k_1) \geq \epsilon_1$. Hence, for each $k_0' > k_0$ there exists $k_1 > k_0'$ such that $\alpha_{DSE}(k_1) \geq \epsilon_2$ where $\epsilon_2 := (1-\epsilon'')\epsilon_1$. This means $\alpha_{DSE}(k) \not<_\infty \epsilon_2$.

Strong oracle ($1 - \alpha_{DIE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) strong: Since $|G|$ contains no small prime factors, it follows from Lemma 2.5 that $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/\mathsf{poly}(k)$. Then we can write $\frac{\varphi(|G|)}{|G|} >_\infty 1 - 1/\mathsf{poly}(k)$ and $\gamma(k) >_\infty (1 - 1/\mathsf{poly}(k))^2 = 1 - 1/\mathsf{poly}(k)$. Hence, we have $1 - \gamma(k) <_\infty 1/\mathsf{poly}(k)$. From Lemma 2.7 follows $1 - \gamma(k)\alpha_{DIE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$. Finally, since $1 - \alpha_{DSE}(k) \leq 1 - \gamma(k)\alpha_{DIE}(k)$ we have $1 - \alpha_{DSE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

Case (b): Assume, we are given a DIE input tuple $((G,g),(g^x),(g^z))$ where $x, z \in \mathbb{Z}^*_{|G|}$, and z is either x^{-1} or a random element $c \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$. Set $h := g^z$

and pass $((G,h),(h^t),(h^{tx}))$ to $\mathcal{O}_{DSE}$ where $h^t = g$ and $h^{tx} = g^x$ for some $t \in \mathbb{Z}^*_{|G|}$.

If $z = x^{-1}$ then $t = x$ and the tuple $((G,h),(h^t),(h^{xt}))$ has the form $((G,h),(h^t),(h^{t^2}))$ which is a correct DSE input tuple. This is because h is a generator, and h^t is a group element with $t \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$. Thus, this instance can be solved by the given DSE oracle. However, the probability for a correct answer is not necessarily $\alpha_{DSE}(k)$ since the inputs to the DSE oracle are limited to those with secret exponents from $\mathbb{Z}^*_{|G|}$ whereas its success probability is defined over $\mathbb{Z}_{|G|}$. Let $\alpha'_{DSE}(k)$ and $\alpha''_{DSE}(k)$ denote the oracles' success probabilities under the condition that the random secret exponents x are chosen from $\mathbb{Z}^*_{|G|}$ and from $\mathbb{Z}_{|G|} \setminus \mathbb{Z}^*_{|G|}$ respectively. It follows

$$\begin{aligned}\alpha_{DSE}(k) &= \alpha''_{DSE}(k)\mathbf{Prob}[x \in_{\mathcal{R}} \mathbb{Z}_{|G|} \setminus \mathbb{Z}^*_{|G|}] + \alpha'_{DSE}(k)\mathbf{Prob}[x \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}] \\ &= \alpha''_{DSE}(k)\big(1 - \frac{\varphi(|G|)}{|G|}\big) + \alpha'_{DSE}(k)\frac{\varphi(|G|)}{|G|}.\end{aligned}$$

By reordering we obtain

$$\begin{aligned}\alpha'_{DSE}(k) &= \frac{\alpha_{DSE}(k) - \big(1 - \frac{\varphi(|G|)}{|G|}\big)\alpha''_{DSE}(k)}{\frac{\varphi(|G|)}{|G|}} \\ &\geq \alpha_{DSE}(k) - \big(1 - \frac{\varphi(|G|)}{|G|}\big)\alpha''_{DSE}(k) \\ &\geq \alpha_{DSE}(k) - \big(1 - \frac{\varphi(|G|)}{|G|}\big)\end{aligned}$$

where in the last inequality we set $\alpha''_{DSE}(k) = 1$ to lower bound $\alpha'_{DSE}(k)$. Thus, the oracle answers correctly on the restricted inputs with probability at least $\alpha'_{DSE}(k) \geq \alpha_{DSE}(k) - (1 - \frac{\varphi(|G|)}{|G|})$.

If $z \neq x^{-1}$ then $t \neq x$ and the tuple $((G,h),(h^t),(h^{xt}))$ is a correct (random) DSE input tuple. This is because (i) h is a generator, and (ii) h^t and h^{xt} are group elements (with $x, t \in_{\mathcal{R}} \mathbb{Z}^*_{|G|}$) representing legal public and random parts of the DSE input tuple. However, the inputs to the DSE oracle are now limited to those with secret exponents (x,t) from $\mathbb{Z}^*_{|G|} \times \mathbb{Z}^*_{|G|}$. Thus, similar to the previous (correct) case, we can determine the probability that the oracle answers correctly on these inputs. This probability is $\alpha'_{DSE}(k) \geq \alpha_{DSE}(k) - \big(1 - (\frac{\varphi(|G|)}{|G|})^2\big)$.

Success probability: We have $\alpha'_{DSE}(k) \geq \alpha_{DSE}(k) - \big(1 - \frac{\varphi(|G|)}{|G|}\big)$ for the correct case (i.e., $z = x^{-1}$) and $\alpha'_{DSE}(k) \geq \alpha_{DSE}(k) - \big(1 - (\frac{\varphi(|G|)}{|G|})^2\big)$ for the random case (i.e., $z \neq x^{-1}$). Hence, for the resulting success probability the following holds

$$\alpha_{DIE}(k) \geq \alpha_{DSE}(k) - \big(1 - (\frac{\varphi(|G|)}{|G|})^2\big).$$

In the following we set $\gamma(k) := \big(\frac{\varphi(|G|)}{|G|}\big)^2$ (Note that $|G|$ is a function of the security parameter k, see also Lemma 2.5). Depending on the success probability of DSE oracle we have the following cases:

Perfect oracle ($\alpha_{DSE}(k) \not<_\infty 1$): The resulting success probability is also perfect because DIE instances represent legal inputs to the DSE oracle, and they all are solved by the perfect oracle.

Weak oracle ($\alpha_{DSE}(k) \not<_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) weak: As shown in the case (a), for $|G|$ with no small prime factors we have $1-\gamma(k) <_\infty 1/\mathsf{poly}(k)$. The subtraction of a negligible function from a not negligible function results in a not negligible function, i.e., $\alpha_{DSE}(k) - (1-\gamma(k)) \not<_\infty 1/\mathsf{poly}(k)$. This implies $\alpha_{DIE}(k) \not<_\infty 1/\mathsf{poly}(k)$.

Invariant oracle ($\alpha_{DSE} \not<_\infty \epsilon_1$): The resulting success probability is (asymptotically) invariant: As shown in the case (a), for $|G|$ with no small prime factors we have $1-\gamma(k) <_\infty 1/\mathsf{poly}(k)$. More precisely, for any ϵ' there exists a k_0 such that for all $k > k_0$, $1-\gamma(k) < \epsilon'$ holds. Since $\alpha_{DSE}(k) \not<_\infty \epsilon_1$, for any k_0' there exists a $k_1 > k_0'$ such that $\alpha_{DSE}(k_1) \geq \epsilon_1$. Thus, for any $k_0' > k_0$ there exists a $k_1 > k_0'$ such that $\alpha_{DSE}(k_1) - (1-\gamma(k_1)) \geq \epsilon_2$ where $\epsilon_2 := \epsilon_1 - \epsilon'$. Hence, we can write $\alpha_{DSE}(k) - (1-\gamma(k)) \not<_\infty \epsilon_2$. Finally, since $\alpha_{DIE} \geq \alpha_{DSE}(k) - (1-\gamma(k))$, it follows that $\alpha_{DIE} \not<_\infty \epsilon_2$.

Strong oracle ($1-\alpha_{DSE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$): The resulting success probability is (asymptotically) strong: From $\alpha_{DIE}(k) \geq \alpha_{DSE}(k) - (1-\gamma(k))$ follows $1-\alpha_{DIE}(k) \leq 1-\alpha_{DSE}(k) + (1-\gamma(k))$. As shown in the case (a), for $|G|$ with no small prime factors we have $1-\gamma(k) <_\infty 1/\mathsf{poly}(k)$. Further, we have $1-\alpha_{DSE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$. Thus, the right side of the above inequality is a not non-negligible function as it is the sum of a not non-negligible and a negligible functions. Hence, we can write $1-\alpha_{DSE}(k) + (1-\gamma(k)) \not\geq_\infty 1/\mathsf{poly}(k)$, implying $1-\alpha_{DIE}(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

■

Remark 2.22. The reductions in Theorem 2.16 are proven for group orders with no small prime factors. However, they also hold for all other group orders provided the group order is known (see also Remark 2.21) Thus, the following holds.

Theorem 2.17

$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}$-DSE(c:∗; g:m; f:∗,o)

$$\Longrightarrow$$

$$\ast\text{-DSE(c:}\ast\text{; g:m; f:}\ast\text{,o)} \quad \overset{\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-DIE(c:}\ast\text{; g:m; f:}\ast\text{,o)}}{\Longleftarrow} \quad \ast\text{-DIE(c:}\ast\text{; g:m; f:}\ast\text{,o)}$$

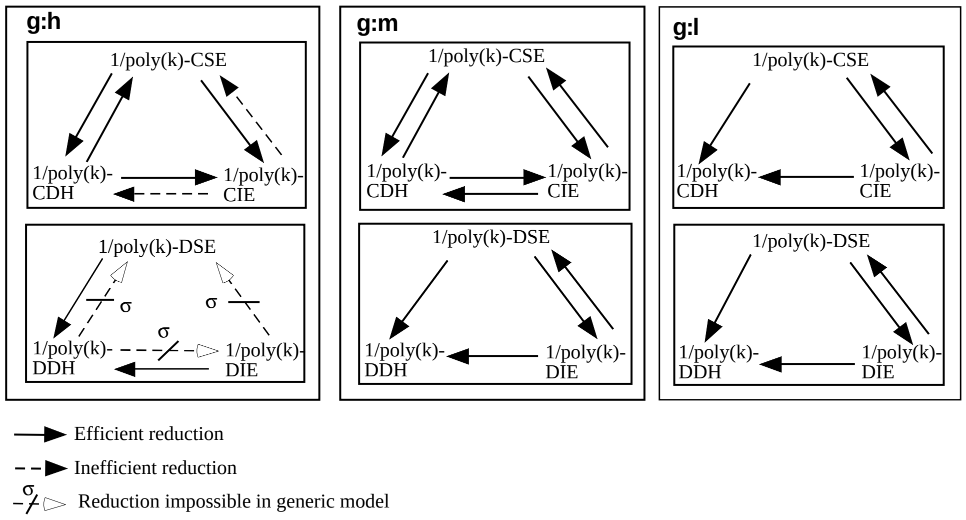
□

Remark 2.23. The reduction (DIE to DSE) in Theorem 2.16 (2.17 respectively) does not hold for perfect success probability due to the introduced error probability. However, for groups of prime order the reduction also holds for perfect oracles as the only special case $x = 0$ can be explicitly handled, i.e., one can easily test whether $g^0 = 1$ is the input. ○

2.7 Summary and Conclusion

In this chapter, we identify the parameters relevant to cryptographic assumptions. Based on this, we present a framework and notation for defining assumptions related to Discrete Logarithms. Using this framework these assumptions can be precisely and systematically classified. Wider adoption of such a terminology would ease the study and comparison of results in the literature, e.g., the danger of ambiguity and mistakes in lengthily stated textual assumptions and theorems would be minimized. Furthermore, clearly stating and considering these parameters opens an avenue to generalize results regarding the relation of different assumptions and to get a better understanding of them. A parameter in defining assumptions previously ignored in the literature is granularity. We show (as summarized in Figure 2.3) that varying this parameter leads to surprising results: We prove that some DL-related assumptions are equivalent in one case (medium granular) and provably not equivalent, at least not in a generic sense, in another case (high granular). Furthermore, we show that some reductions for medium granularity are much more efficient than their high-granular version leading to considerably improved concrete security, in particular as medium granularity results in weaker assumptions than high-granular ones. However, note that medium- or low-granular assumptions apply in cryptographic settings only when the choice of system parameters is guaranteed to be truly random. Interesting open questions remain to be answered: While for both CDL and CDH it can be shown that their high- and medium-granular assumptions are equivalent, this is not yet known for DDH (also briefly mentioned as an open problem by Shoup (1999)). Only few relations can be shown for low-granular assumptions as no random self-reducibility is yet known. However, achieving such "full" random self-reducibility seems very difficult in general (if not impossible) in number-theoretic settings (Boneh 2000) contrary to, e.g., lattice settings used by Ajtai and Dwork (1997). Finally, high granularity is almost intrinsic in the generic model and it is not clear how to extend the generic model to medium or low granularity. Our surprising results also

Figure 2.3 Summary of our results



throw some shadow of doubt onto the use of the generic model as a tool to show impossibility results. It remains to be further explored whether these results are due to the limitations of the generic model or are really intrinsic differences between assumptions with medium and high granularity.

Chapter 3

Conventions and Basic Building Blocks

This chapter presents the conventions and the cryptographic primitives that serve as foundation for the fingerprinting schemes handled in this thesis. In particular, we motivate and explain in more details those cryptographic primitives and aspects which play a major role in later considerations.

3.1 General Notation

Set and Groups: The set of **natural numbers** is $\mathbb{N}$ and $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. $\mathbb{P}$ and $\mathbb{P}_k$ denote the set of **primes** and the set of primes of bit-length k. Recall that for a positive integer n, $\mathbb{Z}_n$ denotes the set of **integers modulo** n, which is the set of distinct equivalence classes under the relation of congruence modulo n in $\mathbb{Z}$. The multiplicative group of $\mathbb{Z}_n$ is denoted with $\mathbb{Z}_n^*$. For the integers a and b, $a|b$ denotes a divides b.

For a set A, we have $A^n := A \times A \cdots \times A$ (n times). An **alphabet** Σ is a finite set of symbols *sym*. Σ^l denotes the set of strings/words consisting of l symbols where each symbol is an element of Σ. We denote the length of a word x with *length*(x). The subgroup relation is denoted by $<$, i.e., for groups H and G we write $H < G$ if H is a real subgroup of G.

$\mathbb{F}_m$ denotes a **finite field** of order m where m is a prime power. The case $m = 2^e$ is of special interest since computations can be implemented more efficiently. The elements of $\mathbb{F}_{2^e}$ can be represented as integers from 0 to $2^e - 1$. Thus, they can be encoded by binary words of length e. The addition and subtraction of elements $a, b \in \mathbb{F}_{2^e}$ are computed by XOR operation $a \pm b := a \oplus b$, i.e., bit wise addition modulo 2. Note that the arithmetic of elements in such field is identical to polynomial arithmetic modulo a primitive polynomial $q(x)$ of degree e over $\mathbb{F}_2$, i.e., one can set $\mathbb{F}_{2^e} := \mathbb{F}_{2^e}[x]/q(x)$ (see Lidl and Niederreiter (1997) or Koblitz (1987)). An element $z \in G$ is represented as a polynomial over $\mathbb{F}_{2^e}$. The multiplication and division are more complex. Multiplication can be performed by first converting the binary numbers to their polynomial equivalents over the field $\mathbb{F}_2$, multiplying the polynomials and then converting the result back to binary. This can be implemented efficiently by using logarithmic mappings, i.e., each element is represented by its discrete logarithm. Multiplication of elements means adding their logarithms and division means subtracting them.

Vectors: A tuple is usually denoted by a vector. For instance for a set G, the elements of G^n are represented as vectors $\vec{x} := (x_1, x_2, \cdots, x_n)$ with $x_i \in G$. We denote the number of the components/symbols of a vector $\vec{x}$ by $\overrightarrow{length}(\vec{x})$. Given two vectors $\vec{x} := (x_1, \cdots, x_n)$ and $\vec{y} := (y_1, \cdots, y_n)$, we define the component wise multiplication of these vectors by the operation $\bullet$:

$$\vec{x} \bullet \vec{y} := (x_1 y_1, \cdots, y_n y_n)$$

where $x_i y_i$ is the ordinary multiplication of the components x_i and y_i.

Let $\oplus$ be the XOR operation (bit wise addition modulo 2). Then we define

$$\vec{x} \oplus \vec{y} := (\mathsf{bin}(x_1) \oplus \mathsf{bin}(y_1), \cdots, \mathsf{bin}(y_n) \oplus \mathsf{bin}(y_n))$$

where $\mathsf{bin}()$ is a binary encoding (see below).

Encodings: Let G be a finite cyclic group. We are mainly concerned with the following encodings:

Integer encoding: $\mathsf{int} : G \mapsto \mathbb{N}_0$ is an efficiently computable and invertible integer encoding function.

Binary encoding: $\mathsf{bin} : \mathbb{N}_0 \mapsto \{0,1\}^l$ is an efficiently computable and invertible binary encoding function. We denote the binary representation of $x \in \mathbb{N}_0$ with

$$[\ x_j\ ]_{0 \le j \le l-1} := (x_{l-1}, x_{l-2}, \cdots, x_1, x_0) := \mathsf{bin}(x)$$

where the symbols x_i are from $\{0,1\}$. The binary length of an element x is denoted by $length_2(x)$ (for dyadic representation, we obviously have $length_2(x) = \lceil \log_2(x) \rceil$.) For $\vec{x} = (x_1, x_2, \cdots, x_n)$ with $length_2(x_i) = l$ we define

$$\begin{aligned} [\ x_{i,j}\ ]_{1 \le i \le n, 1 \le j \le l} &:= \begin{bmatrix} x_{1,1}, & x_{1,2}, & \cdots, & x_{1,l} \\ x_{2,1}, & x_{2,2}, & \cdots, & x_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1}, & x_{n,2}, & \cdots, & x_{n,l} \end{bmatrix} \\ &:= \overrightarrow{\mathsf{bin}}(\vec{x}). \end{aligned}$$

The dyadic representation of an integer $x \in \mathbb{N}_0$ is written as $x = \sum_{j=0}^{l-1} x_j 2^j$ with $x_j \in \{0,1\}$.

Unary encoding: $\mathsf{un} : \mathbb{N}_0 \mapsto \{0,1\}^{l'}$ is an efficiently computable (in a security parameter k) and invertible unary encoding function. We denote the unary representation of $x \in \mathbb{N}_0$ as follows:

$$(x_0^{\mathsf{un}}, \cdots, x_i^{\mathsf{un}}, \cdots, x_{l'}^{\mathsf{un}}) := \mathsf{un}(x)$$

where

$$x_i^{\mathsf{un}} := 1_{i=x} := \begin{cases} 1 & : \text{ if } i = x \\ 0 & : \text{ else} \end{cases}$$

for $0 \leq i \leq l'$.

Remark 3.1. As mentioned above the binary and unary representations of group elements must be computable. This requires the parameters l and l' to be polynomial in the security parameter k: For the binary representation $\log(|G|) \leq l \leq k^d$, $d > 0$ has to hold, and thus, $|G| \leq 2^{k^d}$. For the unary representation $|G| \leq l' \leq k^d$ has to hold, and thus, $|G| \in \mathsf{poly}(k)$. ○

Protocols and algorithms: The protocols we are concerned with consist of several parties exchanging messages where each party $\mathcal{P}_i$ is modeled by an interactive probabilistic algorithm. A common model for the computational complexity of interactive algorithms is based on **Interactive Turing Machines** (ITM) (see e.g., Goldwasser, Micali, and Rackoff (1989)). An ITM is a deterministic multi-tape Turing machine consisting of the following components: (i) a local read-and-write worktape, (ii) a local read-only random tape filled with uniformly distributed random bits before the start of the computation, and (iii) a read-only receiving tape and a write-only sending tape for communication with other machines (see Goldreich (2001b) for more details).

We denote a protocol $\mathsf{ProtName}()$ between the interactive algorithms $\mathcal{P}_1, \cdots, \mathcal{P}_n$ as follows:

$$\begin{aligned}&(\mathcal{P}_1 : \mathit{output}_{\mathcal{P}_1};\ \cdots;\ \mathcal{P}_n : \mathit{output}_{\mathcal{P}_n})\\ &\quad\leftarrow\ \mathsf{ProtName}(\mathcal{P}_1 : \mathit{input}_{\mathcal{P}_1};\ \cdots;\ \mathcal{P}_n : \mathit{input}_{\mathcal{P}_n};\ * : \mathit{input}).\end{aligned}$$

Each party $\mathcal{P}_i$ inputs its input values(s) $\mathit{input}_{\mathcal{P}_i}$ and may obtain output value(s) $\mathit{output}_{\mathcal{P}_i}$ after the protocol $\mathsf{ProtName}()$ has ended. In our notation, different parties are separated by semicolon. The inputs (outputs) of a party $\mathcal{P}_i$ are separated with comma. **Common input**, i.e., values input by all parties, is denoted by the place holder "*". If a party is not required to input any value to the protocol, or it does not obtain any output after protocol completion, then we denote this by a "–" written after the party's name. If a protocol fails, yet, this leaves the system in a safe state, we will (silently) assume that an abort indicator is passed to the protocols at higher layers, and if this has no security impact, those protocol(s) also abort.[1]

Example 3.1 *Consider a protocol* $\mathsf{sum}()$ *between three parties* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *and* $\mathcal{P}_3$*. Let* c *denote the common input. Further, let* $input_{\mathcal{P}_1} := a$*,* $input_{\mathcal{P}_2} := (b, \sqrt{c})$ *and* $input_{\mathcal{P}_3} := d$*. Assume, only* $\mathcal{P}_2$ *can compute the square root of* c*. The output to* $\mathcal{P}_1$*,* $\mathcal{P}_2$ *and* $\mathcal{P}_3$ *should be* $y_1 := b + d + \sqrt{c}$*,* $y_2 := a + d$ *and* $y_3 := a + b + \sqrt{c}$*, respectively. We denote this scenario with*

$$(\mathcal{P}_1 : y_1;\ \mathcal{P}_2 : y_2;\ \mathcal{P}_3 : y_3) \quad \leftarrow \quad \mathsf{sum}(\mathcal{P}_1 : a;\ \mathcal{P}_2 : b, \sqrt{c};\ \mathcal{P}_3 : d;\ * : c).$$

Note that the input/output behavior mentioned above applies only to correct parties, i.e., those who behave according to the protocol.

We use the notation for multi-party protocols also for non-interactive algorithms executed by a party $\mathcal{P}$, i.e., we write:

$$(\mathcal{P} : output_{\mathcal{P}}) \leftarrow \mathsf{AlgoName}(\mathcal{P} : input_{\mathcal{P}})$$

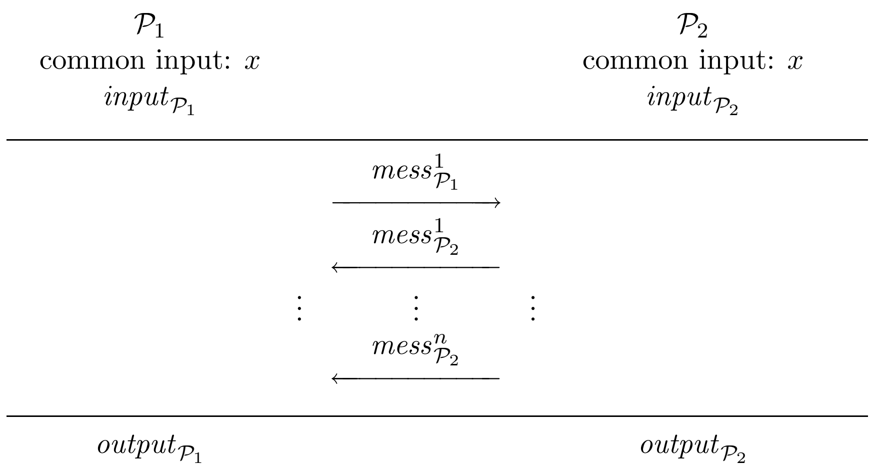
and, when it is clear from the context, we omit the name of the executing party $\mathcal{P}$ in the notation and write only

$$output \leftarrow \mathsf{AlgoName}(input).$$

We sometimes present the interaction flow of interactive protocols, as shown in Figure 3.1, between two (interactive) algorithms $\mathcal{P}_1$ and $\mathcal{P}_2$. In general, the interaction (or conversation) between the involved parties consists of several steps where in each step only one message is sent from or received by a party. Informally, each party (here algorithm $\mathcal{P}_i$, $i = 1, 2$), in its current state, takes the message it receives, may perform some computations and, outputs a message $mess_{\mathcal{P}_i}$ to be sent. Finally it enters into its next state.

[1] One may explicitly consider an indicator signal *ind* to a party only in case a caller is required to perform special recovery, e.g., in a protocol where calling on a third party such as a court is required to resolve the situation.

Figure 3.1 Example of an interactive 2-party protocol



3.2 Number Theoretic Preliminaries

3.2.1 Factoring Numbers

The security of many cryptographic schemes relies on the intractability of **factoring** integers, i.e., given an integer $n \in \mathbb{N}$ find pairs (p_i, e_i) with $p_i \in \mathbb{P}$ distinct primes and $e_i \in \mathbb{N}$ such that $n = \prod_{i=1}^{m} p_i^{e_i}$ (see also Lenstra (2000), Lenstra and Verheul (2001))

There exists a number of integer factorization algorithms from which the most relevant ones are principally similar to those for computing discrete logarithms. The most efficient known factoring algorithm is the **number field sieve** (Lenstra and Lenstra 1993). Its expected running time is of the order $L_n[1/3, c_0]$ where $L_x[t, \gamma] = e^{\gamma + o(1))(\log x)^t (\log(\log x))^{1-t}}$ and $c_0 = (64/9)^{1/3}$ (see also Lenstra 2001).

In many cryptographic applications we are concerned with integers of the form $n = pq$ where $p, q \in \mathbb{P}$ and $p \neq q$.[2] The primes p, q should be selected in such a way that the factorization of n is hard. The main restrictions are that p and q should have approximately the same bit length $f(k)$ where f is a polynomial function in the security parameter k, and that they should be sufficiently large (see Section 2.2 and Menezes, van Oorschot, and Vanstone 1997 and Kaliski 1998 for a discussion on other restrictions)

[2]The most prominent example for applying integers of this form is the **RSA encryption scheme** by (Rivest, Shamir, and Adleman 1978).

3.2.2 Quadratic Residues

A **quadratic residue** modulo a given number $n \in \mathbb{N}$, $n > 1$ is defined as follows:

Definition 3.1 (Quadratic Residues) Let $n \in \mathbb{N}$, $n > 1$ and $y \in \mathbb{Z}_n^*$ Then y is called a quadratic residue modulo n, if there exists $x \in \mathbb{Z}_n^*$ such that $x^2 \equiv y \bmod n$. The set of quadratic residues modulo n is denoted as follows:

$$\mathsf{QR}_n := \{y \in \mathbb{Z}_n^* | \exists x \in \mathbb{Z}_n^* : x^2 \equiv y \bmod n\}.$$

◇

The values x are called **square roots** of y and are denoted by $x \equiv y^{1/2} \bmod n$. An integer $a \in \mathbb{Z}_n^*$ is called a **quadratic non-residue** modulo n if a is not a quadratic residue modulo n. We denote the set of quadratic non-residues modulo n with $\mathsf{QNR}_n := \mathbb{Z}_n^* \setminus \mathsf{QR}_n$.

Some useful rules for squares and roots are as follows: Let $n \in \mathbb{N}$, $n > 1$ then the following holds:

1. $\mathsf{QR}_n < \mathbb{Z}_n^*$.
2. $x \equiv y^{1/2} \bmod n \Rightarrow -x \equiv y^{1/2} \bmod n$.
3. $y_1 \in \mathsf{QR}_n$, $y_2 \in \mathsf{QR}_n \Rightarrow y_1 y_2 \in \mathsf{QR}_n$.
4. $y_1 \in \mathsf{QR}_n$, $y_2 \in \mathsf{QNR}_n \Rightarrow y_1 y_2 \in \mathsf{QNR}_n$.
5. $y_1 \in \mathsf{QNR}_n$, $y_2 \in \mathsf{QNR}_n \Rightarrow y_1 y_2 \in \mathsf{QR}_n$.

A useful notation for recognizing quadratic residues from non-residues modulo a prime number is called **Legendre symbol** and defined as follows:

Definition 3.2 (Legendre Symbol) Let $p \in \mathbb{P}$ be an odd prime and $y \in \mathbb{Z}$. The Legendre symbol $\left(\frac{y}{p}\right)$ is defined as

$$\left(\frac{y}{p}\right) := \begin{cases} 0 & : \text{ if } p | y \\ 1 & : \text{ if } y \in \mathsf{QR}_p \\ -1 & : \text{ if } y \in \mathsf{QNR}_p \end{cases}$$

◇

The generalization of Legendre symbol to arbitrary odd integers $n \in \mathbb{N}$ is called **Jacobi symbol** defined as follows:

Definition 3.3 (Jacobi Symbol) Let $y \in \mathbb{Z}$ and $n \in \mathbb{N}$, $n \geq 3$ be odd with prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$. The Jacobi symbol $\mathsf{J}_n(y)$ is defined as

$$\mathsf{J}_n(y) := \left(\frac{y}{p_1}\right)^{e_1} \left(\frac{y}{p_2}\right)^{e_2} \cdots \left(\frac{y}{p_m}\right)^{e_m}.$$

◇

The Jacobi symbol can assume either the value 1 or -1, but unlike the Legendre symbol it is not a definite indication of whether a number y is a quadratic residue modulo n. This means that there exist numbers $y \in \mathsf{QNR}_n$ with $\mathsf{J}_n(y) = 1$ (but not the opposite). We denote the set of integers (modulo n) with Jacobi symbol 1 and -1 as follows:

$$\begin{aligned}\mathbb{Z}_n^{(+1)} &:= \{y \in \mathbb{Z}_n^* | \mathsf{J}_n(y) = 1\} \\ \mathbb{Z}_n^{(-1)} &:= \{y \in \mathbb{Z}_n^* | \mathsf{J}_n(y) = -1\}.\end{aligned}$$

Further, the following holds:

1. $\mathsf{QR}_n < \mathbb{Z}_n^{(+1)} < \mathbb{Z}_n^*$.
2. $\mathsf{J}_n(xy) = \mathsf{J}_n(x)\mathsf{J}_n(y)$.

As mentioned before, many cryptographic applications use group families which are subgroups of $\mathbb{Z}_n^*$ where the index n has the form $n = pq$ with $p, q \in \mathbb{P}$ (see also Section 3.2.1). Thus, we restrict our consideration to these group families. In case the factors of n are *known*, certain computations can be performed efficiency whereas the same computations are considered to be infeasible otherwise.

Group Families with Known Group Index Factorization

Let $n = pq$, $p, q \in \mathbb{P}$, $p \neq q$ and p, q odd. Then the following holds:

1. $\mathsf{J}_n(y) = 1 \Leftrightarrow (\mathsf{J}_p(y) = 1 \wedge \mathsf{J}_q(y) = 1) \vee (\mathsf{J}_p(y) = -1 \wedge \mathsf{J}_q(y) = -1)$.
2. $y \in \mathsf{QR}_n \Leftrightarrow \mathsf{J}_p(y) = 1 \wedge \mathsf{J}_q(y) = 1$.
3. Every $y \in \mathsf{QR}_n$ has exactly 4 square roots which can be efficiently computed.[3]

Group Families with Special Properties (Blum Integers)

Some computations will be more efficient if the factors of n have the following property: $p \equiv q \equiv 3 \bmod 4$. Integers with this property are also called **Blum integers**. For Blum integers the following results hold:

1. $-1 \in \mathsf{QNR}_n \wedge \mathsf{J}_n(-1) = 1$.
2. All non-squares with Jacobi symbol 1 are exactly the negatives of the squares:
$$\mathbb{Z}_n^{(+1)} \setminus \mathsf{QR}_n = -\mathsf{QR}_n := \{-x | x \in \mathsf{QR}_n\}.$$

[3]Compute $x_p := y^{1/2} :\equiv y^{\frac{p+1}{4}}$ and $x_q := y^{1/2} :\equiv y^{\frac{q+1}{4}}$ and apply CRA (**Chinese Reminder Algorithm**) to the 4 combinations of $\pm x_p$ and $\pm x_q$ (see also Menezes, van Oorschot, and Vanstone 1997) Thus, there are exact 4 different square roots.

Thus, we can write $|\mathbb{Z}_n^{(+1)} \setminus \mathsf{QR}_n| = |\mathsf{QR}_n|$. This is important in the context of commitments schemes based on quadratic residues in later sections.

Group Families with Unknown Group Index Factorization

One can efficiently compute the Jacobi symbol of an integer or randomly and uniformly select elements from QR_n without knowing the factors of n.[4] However, some tasks such as extracting (square) roots and distinguishing quadratic residues from non-residues modulo an integer n are assumed to be hard if n is a hard-to-factor integer (see Section 3.2.1). Note that the computational complexity of some cryptographic primitives (e.g., certain type of commitment schemes) relies on the difficulty of solving these tasks. It can be shown that computing square roots is hard under the factoring assumption, but this is not known for testing quadratic residuosity and, thus, an assumption is formulated for this case called **Quadratic Residuosity Assumption** (QRA). Informally, this assumption says that for $n = pq$ (product of two large primes) no polynomial algorithm is known which can decide, with probability significantly better than pure guessing, whether an element $a \in_{\mathcal{R}} \mathbb{Z}_n^{(+1)}$ is a quadratic residue or not. For a formal definition of the assumption we generalize our definitions of assumptions from Section 2.3 to capture the following items:

- The underlying algebraic structure $\mathbb{Z}_n^{(+1)}$ is a subgroup of $\mathbb{Z}_n^*$. For composite n this group is not a cyclic group in general, and thus, the structure instance contains no group generator. Hence, we allow structure instances to contain more general groups.

- One can interpret the success probability of an adversary for solving a computational problem as the sum of the following terms: One term represents the probability that the adversary finds the solution by pure guessing, and the other represents its capability of doing more, i.e., the significance of its success beyond pure guessing. Let $|sol|$ be the cardinality of the solution domain, and B be the bound on the tolerated success probabilities for adversary (weak, invariant, etc), as defined in Section 2.2. The guessing term is then $\frac{1}{|sol|}$. We define the additional term as $\frac{B}{(1-1/|sol|)}$ where we normalized B. Now, we express the bound for the success probabilities by $B' := \frac{1}{|sol|} + \frac{B}{(1-1/|sol|)}$.

  As an example, consider computational DL-based assumptions with weak success probability, i.e., $B := \frac{1}{k^d}$, as defined in Section 2.3. For

[4]The Jacobi symbol of $a \in \mathbb{Z}_n^*$ can be computed in $O((\log n)^2)$ bit operations (Menezes, van Oorschot, and Vanstone 1997). Choosing quadratic residues is simple since we can choose a random element $a \in_{\mathcal{R}} \mathbb{Z}_n^*$, and then compute $y := a^2 \bmod n$. The quadratic residues y, we obtain in this way, have uniform distribution over QR_n since every element in QR_n has exactly 4 roots.

the upper bound B' we can write $B' = \frac{1}{|sol|} + \frac{1}{k^d}\left(\frac{1}{(1-1/|sol|)}\right) \in 1/\mathsf{poly}(k)$, since $1/|sol| \in 1/\mathsf{poly}(k)$, so nothing changes with respect to previous results.

Considering the above issues for the quadratic residuosity assumption it follows:

- The structure instance contains only a group but no generator. Therefore, the granularity can only be high or low. For the high-granular probability space the group family $\mathbb{Z}_n^{(+1)}$ is fixed whereas it is not for the low granular case since the probability space is defined over the choice of primes p and q.

- The quadratic residuosity problem family, denoted by index QR, relates a (secret) value w with a (public) value $y \in G$ where w is either a root of y, i.e., $w = x :\equiv y^{1/2}$ or it is the empty string $w = e$ indicating that no root for y exists. For our purposes, we set $G = \mathbb{Z}_n^{(+1)}$, $PI_{\mathrm{QR}}{}^{SI} := (\mathbb{Z}_n^{(+1)})$, $PI_{\mathrm{QR}}{}^{priv} := (w)$, $PI_{\mathrm{QR}}{}^{publ} := (y)$ and $PI_{\mathrm{QR}}{}^{sol} := \{0,1\}$. The public part is a value $y \in \mathbb{Z}_n^{(+1)}$, and the secret part is the square root $x :\equiv y^{1/2} \bmod n$ if $y \in \mathsf{QR}_n$, and e otherwise. The solution domain is the set $sol := \{0,1\}$ where 0 means quadratic residue and 1 quadratic non-residue. Thus, we have $|sol| = 2$.

For our purposes, it suffices to consider the low-granular weak version of QRA. We also immediately define QRA for group families $\mathbb{Z}_n^{(+1)}$ where the group index n is a Blum integer.[5]

Definition 3.4 (Quadratic Residuosity Assumption)

$$1/\mathsf{poly}(k)\text{-CQR(c:u; g:l; f:}\mathbb{Z}_n^{(+1)}),$$

or in terms of probability

$\forall \mathcal{A} \in \mathcal{UPTM};$
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$

$(|\,\mathbf{Prob}[\mathcal{A}(\mathcal{C}, SI, PI_{\mathrm{QR}}{}^{publ}) \in PI_{\mathrm{QR}}{}^{sol} ::$
$\quad G \in [SG_{\mathbb{Z}_n^{(+1)}}(1^k)];$
$\quad SI \leftarrow G;$
$\quad PI_{\mathrm{QR}} \leftarrow SPI_{\mathrm{QR}}(SI);$
$\quad \mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U}$
$\quad]-1/2\,|\cdot 2) \ < 1/k^{d_1}.$
$\diamond$

[5]Note that due to *Dirichlet's prime number theorem* there are infinitely many integers with this property.

Remark 3.2. As elements are sampled from $\mathbb{Z}_n^{(+1)}$, it always hold that $\mathsf{J}_n(a) = 1$. From $|\mathsf{QR}_n| = |\mathbb{Z}_n^* \setminus \mathsf{QR}_n|$ follows that with probability 1/2 a randomly selected element is a quadratic residue or not.

Remark 3.3. We consider the low granular version of QRA because in later protocols the party who generates n is only secure if it generates n correctly.

Remark 3.4. For completeness, one has to define assumptions of the type above in the generic model. For this, one has to consider the generic model for such group families first (see also Damgård and Koprowski 2002). However, this is not our concern in this work, and will be the subject of the future research. ∘

3.3 Random Oracle

The **random oracle** methodology introduced by Bellare and Rogaway (1993) is a model for designing and assessing cryptographic systems (protocols). Its approach is as follows: First, an *ideal* computational model is specified where every party is given access to random functions, called **random oracle**. The next step is to design the desired system (protocol) within this model and then prove its security. Finally, all random oracles are replaced by efficient cryptographic functions. Thus, one hopes that if the efficient cryptographic functions do not have security flaws then the protocol is still secure in the real world (see also Bellare (1998)) The main advantage of this approach is that it improves on methods and systems which are completely ad hoc. The main drawback is that there is no real formal connection between the security proven in the ideal model and the actual security of the protocol in the *real* world (after replacing random oracles with "real" cryptographic functions.) In other words, informal arguments suggest that there is a close match between the security of the protocol in the ideal and real world, if the protocol and the replacements of the random oracles are designed carefully. Many provably secure cryptographic systems are based on inefficient (and sometimes unrealistic) primitives whereas there are many efficient systems with very poor or no security analysis. In this context, the main concern of the random oracle model is to provide the possibility of having efficient designs and "security". Although the security-guarantees achieved in the random oracle model are not at the same level as the standard provable security approach offers, it is considerably better than ad-hoc protocol design and security analysis. A typical instantiation of the random oracle is a **cryptographic hash function** (see Preneel (1998)). Hash functions such as Secure Hash Algorithm (SHA-1) in Laboratory (1995), MD5 in Rivest (1992) or RIPEMD-160 in Dobbertin, Bosselaers, and Preneel (1996) are believed to be "good" enough as replacement for random oracle. The claim of random oracle model – that the security proofs for protocols in the ideal model remain valid to a significant degree after converting it into the real world – has been defeated by negative results (Canetti, Goldreich, and Halevi 1998). Informally, they show examples of cryptographic systems which can be proven secure in the random oracle model, but, all of their instantiations result in insecure cryptographic systems. However, one should note that these results are created by artificially constructed scenarios which are not relevant for many real world implementations. These negative results do not discredit the random oracle scope but rather show its limitations.

3.4 Public-Key Cryptography

3.4.1 Introduction

Cryptosystems can be divided into two classes: **symmetric cryptosystems** and **asymmetric cryptosystems**. Informally, the difference between symmetric and asymmetric systems is as follows: In a symmetric system, both parties share the same secret information, usually the same secret key. In an asymmetric system there are different keys, usually a secret key and a public-key (hence the term "public-key cryptography") where each key is used for a different task. The idea of public-key cryptography was first proposed by Diffie and Hellman in Diffie and Hellman (1976).[6] Asymmetric cryptosystems offer several benefits over symmetric ones. For instance, in a symmetric cryptosystem, a different key is needed for each possible set of parties communicating with each other, and additionally, these keys have to be distributed confidentially. In asymmetric cryptosystems, in contrast, distribution of keys over authentic channels suffices. Further, asymmetric cryptosystems allow us to achieve important security goals such as **non-repudiation** that is required for **accountability**. This is crucial when we require cryptosystems to provide means to provably identify cheating parties, and to make them accountable (for more comparisons between these classes see Menezes, van Oorschot, and Vanstone (1997)).

3.4.2 Encryption Schemes

In this section we consider asymmetric encryption schemes and their corresponding components. Let k denote the security parameter and M_k the message space (parameterized by the security parameter). Later, we review briefly concrete encryption schemes which we will use in the fingerprinting protocols. The components of an asymmetric encryption schemes are as follows:

- **Key generation**: The key generation algorithm is described by the probabilistic algorithm $\mathsf{GenKeyEnc}()$ as follows: $(sk, pk) \leftarrow$

[6]The British government announced that public-key cryptography was originally invented by British cryptographers at Government Communications Headquarters (see Singh (1999)).

$\mathsf{GenKeyEnc}(k)$ where k is the security parameter, and the strings sk and pk are the **secret key** and the **public key** of the scheme.

- **Encryption**: Let $m \in M_k$ be the message to be encrypted called the **plaintext**. The encryption function is described by a probabilistic algorithm $\mathsf{Encrypt}()$ as follows: $enc \leftarrow \mathsf{Encrypt}(pk; m)$ where pk is the public key. The output enc is called the **ciphertext**.
- **Decryption**: The decryption function is described by the deterministic algorithm $\mathsf{Decrypt}()$ as follows: $x \leftarrow \mathsf{Decrypt}(sk; enc)$ where sk is the secret key of the scheme. The output x is either a plaintext m or *failed*. The decryption must fulfill the following correctness condition: $\forall k \in \mathbb{N}, \forall (sk, pk) \in [\mathsf{GenKeyEnc}(k)], \forall m \in M_k$ then

$$enc \in [\mathsf{Encrypt}(pk; m)] \Rightarrow \mathsf{Decrypt}(sk; enc) = m.$$

3.4.2.1 Security Requirements

The notion of security for an asymmetric encryption scheme should capture several properties. In general, we may distinguish between the security of the scheme against **passive adversaries** and against **active adversaries**. Passive adversaries only observe the communication channel whereas active adversaries actively attack a target system, e.g., through interaction with the system or manipulation of communication channels. An encryption scheme secure against active attacks is also secure against passive attacks. For our purposes, we require only asymmetric encryption schemes secure against passive adversaries (semantically secure schemes).

Semantic Security

For an encryption scheme secure against passive attacks, one may desire the following properties: Firstly, the secret (decryption key) should not be computable from the public (encryption) key. Secondly, messages should not be recovered by only observing their encryptions. However, these requirements are still not sufficient because our primary goal is to hide the content such that the ciphertext and the public parameters reveal even no partial information. One of the first attempts to formally define the security against passive adversaries was **polynomial indistinguishability** by Goldwasser and Micali (1984). Loosely speaking, the following game is considered: The adversary chooses two messages m_0^*, m_1^*. The encryption oracle (honest user) chooses one of the messages at random, encrypts it, and gives the result to the adversary. Next the adversary has to guess which message is contained in the encryption. The definition requires that no polynomial-time adversary is able to predict the right message with a probability significantly better than pure guessing. More formally, we have the following definition:

Definition 3.5 An asymmetric encryption system, denoted by the tuple ($\mathsf{GenKeyEnc}()$, $\mathsf{Encrypt}()$, $\mathsf{Decrypt}()$, M_k), is called secure in the sense of (polynomial) indistinguishability if the following holds:

$\forall \mathcal{A}_1, \mathcal{A}_2 \in \mathcal{UPTM}$;
$\forall d_1 > 0$; $\exists k_0$; $\forall k > k_0$;
$\mathbf{Prob}[\mathcal{A}_2(state_{\mathcal{A}_1}, enc_b) = b ::$
$\quad (sk, pk) \leftarrow \mathsf{GenKeyEnc}(k);$
$\quad rand \xleftarrow{\mathcal{R}} \mathcal{U};$
$\quad (state_{\mathcal{A}_1}, m_0^*, m_1^*) \leftarrow \mathcal{A}_1(k, rand);$
$\quad b \xleftarrow{\mathcal{R}} \{0,1\};$
$\quad enc_b \leftarrow \mathsf{Encrypt}(pk; m_b^*)$
$] < 1/2 + 1/k^{d_1}$.
$\diamond$

In the definition above, the adversary consists of the two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ where the "knowledge" of $\mathcal{A}_1$ is stored in the state $state_{\mathcal{A}_1}$, and can be used by $\mathcal{A}_2$. Since the adversary can simply guess b correctly with probability 1/2, the definition only requires that the adversary's success probability should not be significantly better than 1/2.

Note that any asymmetric encryption scheme with deterministic encryption function cannot pass the above definition, because given m_0^*, m_1^*, it is easy to find out which message is encrypted to which ciphertext $enc \in \{\mathsf{Encrypt}(pk, m_0^*), \mathsf{Encrypt}(pk, m_1^*)\}$.

An alternative definition called **semantic security** requires that the adversary should not be able to guess any non-trivial function $f(m)$ of the message m based on the ciphertext and public information only. Informally, this means that whatever a passive polynomial adversary is able to compute given the ciphertext and the public information, it is also able to compute without this information. Goldwasser and Micali (1984) show that *semantic security* is equivalent to the *polynomial indistinguishability*.

3.4.2.2 ElGamal Encryption Scheme

The **ElGamal** encryption scheme is a public key encryption scheme proposed by ElGamal (1985). The details of the scheme are as follows:

Key generation: Let G be a finite cyclic group of order q with a group generator g. An example for a concrete G is the multiplicative subgroup $\mathbb{Z}_{p/q}^*$ of $\mathbb{Z}_p^*$ of order q with $p, q \in_{\mathcal{R}} \mathbb{P}$ and $q|(p-1)$ (see also Section 2.1.7 for generation of these parameters). The public/secret key generated by $\mathsf{GenKeyELG}(k)$ are $sk := x \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ and $pk := (G, g, h)$ where $h := g^x$.

Encryption: For a message $m \in G$ the ciphertext is

$$(d_1, d_2) \leftarrow \mathsf{EncryptELG}(pk; m)$$

where $d_1 := g^a$, $d_2 := pk^a m$ and $a \in_{\mathcal{R}} \mathbb{Z}_{|G|}$. Note that the encryption is probabilistic, as a is chosen randomly.

Decryption: For the ciphertext $enc = (d_1, d_2)$ the plaintext is:

$$\frac{d_1}{d_1^{sk}} := m \leftarrow \mathsf{DecryptELG}(sk; d_1, d_2).$$

It is easy to see that

$$\frac{d_2}{d_1^{sk}} = \frac{pk^a m}{(g^a)^x} = \frac{g^{xa} m}{g^{xa}} = m.$$

Remark 3.5. Alternatively, the encryption could be computed as $(d_1 := pk^a, d_2 := g^a m)$, which can be decrypted in the following way:

$$\frac{d_2}{d_1^{sk^{-1}}} = \frac{g^a m}{(pk^a)^{x^{-1}}} = \frac{g^a m}{g^{xax^{-1}}} = m.$$

○

Security

Both forms of the ElGamal scheme are equally secure (Tsiounis and Yung 1998); they are semantically secure under the *Diffie-Hellman Decision* assumption (see also Section 2.2).

3.4.2.3 Okamoto Uchiyama Encryption Scheme

The **Okamoto-Uchiyama** (OU) encryption scheme is proposed by Okamoto and Uchiyama (1998). It is based on efficiently solving discrete logarithm in a specific finite subgroup of $\mathbb{Z}^*_{p^2}$ called p**-Sylow** subgroups where $p \in_{\mathcal{R}} \mathbb{P}$. The details of the scheme are as follows: Let p be an odd prime and

$$\Gamma = \{x \in \mathbb{Z}^*_{p^2} : x \equiv 1 \bmod p\}$$

where $\mathbb{Z}^*_{p^2}$ is a cyclic group of order $p(p-1)$.[7] Γ is the p-Sylow subgroup of $\mathbb{Z}^*_{p^2}$. Now, consider the function $L(x) = \frac{x-1}{p}$. It is well-defined on Γ, and can be interpreted as a logarithmic function with the property $L(ab) = L(a) + L(b) \bmod p$. Further, the following holds:

Corollary 3.1 (Okamoto and Uchiyama 1998) *Let $x \in \Gamma$ such that $L(x) \not\equiv 0 \bmod p$, and $y \equiv x^m \bmod p^2$ for $m \in \mathbb{Z}_p$. Then*

$$m = \frac{L(y)}{L(x)} = \frac{y-1}{x-1} \bmod p.$$

□

[7] The elements of Γ are those from $\mathbb{Z}^*_{p^2}$ with order p. Let g be a generator of $\mathbb{Z}^*_{p^2}$. One can write $\Gamma = \{1, g^{p-1}, g^{2(p-1)}, \cdots g^{(p-1)(p-1)}\}$. It holds for every element $a \in \Gamma$ that $a^p \equiv 1 \bmod p^2$ and $a \equiv 1 \bmod p$.

We denote this scheme in short with OU-scheme. The components of OU-scheme are as follows:

Key generation: The public/secret keys generated by $\mathsf{GenKeyOU}(k)$ are $sk := (p, q)$ and $pk := (n, g, h, k)$ with the following properties:

- $n = p^2q$ where $p, q \in_{\mathcal{R}} \mathbb{P}$, $|p| = |q| = k$, $\gcd(p, q-1) = 1$ and $\gcd(q, p-1) = 1$,
- $g \in_{\mathcal{R}} \mathbb{Z}_n^*$ such that the order of $g_p := g^{p-1} \bmod p^2$ is p, and
- $h = g^n \bmod n$.

Encryption: For a message $m \in \mathbb{Z}_{2^{k-1}}$ the ciphertext $c \leftarrow \mathsf{EncryptOU}(pk; m)$ is computed as follows:

$$c := g^m h^r \bmod n$$

where $r \in_{\mathcal{R}} \mathbb{Z}_n$.

Decryption: A ciphertext c is decrypted by $\mathsf{DecryptOU}(sk; c)$ as follows:

$$\frac{L(c_p)}{L(g_p)} := m \leftarrow \mathsf{DecryptOU}(sk; c)$$

where $c_p := c^{p-1} \bmod p^2$. The decryption follows from the result of Corollary 3.1.

Remark 3.6. A similar encryption scheme based on higher-degree residue is proposed by Benaloh (1994) (see also Naccache and Stern (1998)). However, the decryption mechanism of this scheme is very inefficient. ○

Security

The security properties of the OU-scheme against passive adversary are as follows (Okamoto and Uchiyama 1998): The security of OU encryption function is equivalent to factoring numbers of the form $n = p^2q$ if the adversary succeeds in obtaining the whole plaintext.[8] This means that if an adversary algorithm can compute the whole message by using the ciphertext and the public information, then one can use it to construct an algorithm to factor n. The scheme is semantically secure under the p-subgroup assumption saying that h^r and $gh^{r'}$ with $r, r' \in_{\mathcal{R}} \mathbb{Z}_n$ are computationally indistinguishable.

[8]It is not known whether that factoring numbers of this form is as intractable as factoring numbers of the form $n = pq$ (Okamoto and Uchiyama 1998).

3.4.3 Digital Signatures

Historically, the main concern of cryptography was designing and analyzing encryption systems to provide confidentiality when communicating over insecure channels. Today, cryptography is concerned with designing and analyzing a variety of primitives. A primitive fundamental to authentication, authorization and non-repudiation is the **digital signature**. Informally, a digital signature associates a message with an entity. It is a value which depends on a secret, only known to this entity, and on the message being signed. The goal of a signature is to unforgeably link the identity of the entity to the signature such that any other entity can verify this link without knowing the secret values used in the signing. For instance, if a dispute arises on whether a certain party has signed a document, any honest party can resolve this dispute. For a good overview on digital signatures see Menezes, van Oorschot, and Vanstone (1997), and for a comprehensive treatment of digital signatures see Pfitzmann (1996a).

3.4.3.1 Model

The involved parties are a signer $\mathcal{S}$ and a receiver $\mathcal{R}$. Let k denote the security parameter and M_k the message space (parameterized by the security parameter). The components of an asymmetric signature scheme are as follows:

- **Key generation:** The key generation algorithm is a probabilistic algorithm denoted by $(sk_{\mathcal{S}}, pk_{\mathcal{S}}) \leftarrow \mathsf{GenKeySig}(k)$ where the strings $sk_{\mathcal{S}}$ and $pk_{\mathcal{S}}$ are called the secret **signing key** and the public **verification key** of the scheme.

- **Signing:** Let $m \in M_k$ be the message to be signed. The signing function is a probabilistic algorithm denoted by $\sigma \leftarrow \mathsf{Sign}(sk_{\mathcal{S}}; m)$ where $sk_{\mathcal{S}}$ is the signing key. The output σ is called the **signature**.

- **Verification:** The verification function is a deterministic algorithm denoted by $ind \leftarrow \mathsf{VerSign}(pk_{\mathcal{S}}; m, \sigma)$ where $pk_{\mathcal{S}}$ is the verification key, and $ind \in \{true, false\}$ denotes the result of verification. If this result is *true*, we say that the signature is valid.

  The verification must fulfill the following correctness condition: $\forall k \in \mathbb{N}, \forall (sk_{\mathcal{S}}, pk_{\mathcal{S}}) \in [\mathsf{GenKeySig}(k)], \forall m \in M_k$ then

$$\sigma \in [\mathsf{Sign}(pk_{\mathcal{S}}; m)] \Rightarrow true \leftarrow \mathsf{VerSign}(pk_{\mathcal{S}}; m, \sigma).$$

3.4.3.2 Security Requirements

The main requirement is **unforgeability**, i.e., it must be infeasible (impossible) to generate a valid digital signature on a forged message. Informally, the main types of forgery are:

- **Selective forgery:** The adversary succeeds in generating a valid signature for some messages chosen a priori.
- **Existential forgery:** The adversary succeeds in forging a signature for at least one message, not necessarily of his choice (i.e., the message whose signature is obtained is determined during the attack.)

Informally, the main types of attacks on digital signatures are:

- **Known-signature attack:** The adversary knows message-signature pairs where the messages are not chosen by him.
- **Chosen-message attack:** The adversary obtains valid signatures on messages which are chosen by him before attempting to forge a signature.
- **Adaptive chosen-message attack:** The adversary is allowed to interact with the signer using the signer as an oracle; the adversary may request signatures for messages which depend on the signer's public key and the previously obtained signatures or messages. At the end, the adversary outputs a new message and a valid signature on it.

The strongest security notion for a digital signature scheme is security against existential forgery under adaptive chosen-message attack.

There are a variety of proposals for signature schemes offering different level of security. Well-known signature schemes are related to RSA by (Rivest, Shamir, and Adleman 1978) or ElGamal by ElGamal (1985), Schnorr (1991) and Schnorr (1992). The security of these schemes is based on heuristic arguments. Some less known schemes with provable security (and less efficiency) are Goldwasser, Micali, and Rivest (1988), Cramer and Damgård (1996) and Cramer and Shoup (2000).

In the next section, we briefly review the signature scheme from Schnorr (1991) which we will use at different places in this thesis.

3.4.3.3 Schnorr Signature Scheme

The **Schnorr signature** scheme Schnorr (1991) is based on the discrete logarithm problem and has the following components:

- **Key generation:** The underlying algebraic structure is a cyclic group G of order $|G| := q$ in which the computation of discrete logarithms is assumed to be infeasible.[9] For a concrete structure, one can take G from the group family $\mathbb{Z}^*_{p/q}$ (see Sections 2.1.7 and 2.2). Let $g \in G$ be a group generator. Then we have $sk_{\mathcal{S}} := x \in_{\mathcal{R}} \mathbb{Z}_q$, and $pk_{\mathcal{S}} := h \equiv g^x \bmod p$. We will use the pair (x, h) in the following.

[9]Note that G must also satisfy other requirements, e.g., one should know efficient algorithms for the required arithmetic operations in G (see Section 2.1.1).

- **Signing:** A signature on a message $m \in M_k$ is a pair $(c, r) \leftarrow \mathsf{Sign}(x; m)$ where $(c, r) \in \mathbb{Z}_q \times \mathbb{Z}_q$ can be computed as follows: Choose $w \in_{\mathcal{R}} \mathbb{Z}_q$, and compute $a :\equiv g^w$ and $c := \mathit{hash}(m||a)$ where *hash* is a cryptographic hash function. Finally compute $r := w - cx \bmod q$.

- **Verification:** On receiving (c, r), the receiver's verification is as follows: Compute $c' := \mathit{hash}(m||a')$ where $a' :\equiv g^r h^c$, and accept if and only if $c' = c$.

Deriving the Signature from Identification Protocol

Next, we look at this signature from another point of view. Originally, this signature was derived from Schnorr's identification protocol. In this protocol, the owner of the secret value x proves to a verifying party that she knows a value x such that $h = g^x$ without revealing any "useful" information about x. In other words, this protocol is a (minimum disclosure) proof of knowledge of discrete logarithm of h (see Section 3.7.1). Fiat and Shamir (1987) introduced a general approach to transform this type of identification protocols[10] into a signature scheme. The idea is to create a virtual verifier in the proof protocol by letting the prover (here the signer) compute the challenge c (see Section 3.7.1) herself as $c := \mathit{hash}(m||a)$ where m is the message to be signed, and *hash* is a cryptographic hash function. Now, the receiver accepts if and only if $g^r \equiv ah^{-c}$ for $c = \mathit{hash}(m||a)$.

Security

Pointcheval and Stern (2000) prove in the random oracle model (Section 3.3) that Fiat-Shamir technique provides security against existential forgery through an adaptive chosen message attack, under the discrete logarithm assumption.

3.4.4 Blind Signatures

The concept of **blind signatures** was introduced by Chaum (1983). The goal is to protect the privacy of users in cryptographic applications such as anonymous electronic payment systems (see Section 4.7.2). Other than a normal signature, a blind signature is issued by an interactive protocol $\mathsf{BlindSign}()$ between the signer $\mathcal{S}$ and the receiver $\mathcal{R}$. At protocol completion, $\mathcal{R}$ obtains a signature σ' of $\mathcal{S}$ on the message to be signed while $\mathcal{S}$ knows neither the message nor the signature σ' on it. For blind signatures there is a further security requirement called **blindness**. Informally, it means: Given a set of transcripts of the blind signature protocol runs, and the set of message-signature pairs generated by these protocol-runs, an

[10]More specifically, a three-pass honest-verifier zero-knowledge identification protocol

adversary cannot associate the transcripts to the message-signature pairs with a probability significantly better than pure guessing.

Next, we review a concrete blind signature scheme from Chaum and Pedersen (1993). We will apply an extension of this signature, proposed by Brands (1994), as a building block in our fingerprinting protocols.

3.4.4.1 Chaum-Pederson Blind Signature Scheme

The blind signature scheme proposed by Chaum and Pedersen (1993) is an extension of Schnorr's signature scheme we reviewed in Section 3.4.3.3. Thus, the parameter and key generation remains the same as for Schnorr with signer's key pair (x, h). The protocol for this signature is shown in Figure 3.2. Note that the hash function *hash* is assumed to behave as a random oracle (see Section 3.3). We denote the signature protocol with

$$(\mathcal{R} : m', \sigma';\ \mathcal{S} : -) \leftarrow \mathsf{BlindSignCP}(\mathcal{R} : -;\ \mathcal{S} : sk_{\mathcal{S}};\ * : m)$$

where the message m is the common input, m' is an **unblinded** version of m (a transformation of m), and $\sigma' := (z', a', b', c', r')$ is the unblinded signature on m'. The verification is denoted by $ind \leftarrow \mathsf{VerBlindSignCP}(h; m', \sigma')$ where $ind \in \{true, false\}$ indicates whether the signature passes the verification. Here, σ' is accepted as a signature on m' if and only if the following relations hold:

$$g^{r'} \equiv a'h^{c'} \bmod p, \text{ and } m'^{r'} \equiv b'z'^{c'}.$$

.

Figure 3.2 The blind signature of Chaum-Pederson (CP)

$$\begin{array}{rcl}
\mathcal{R} & & \mathcal{S} \\
 & & z \leftarrow m^x \\
 & & w \in_R \mathbb{Z}_q \\
 & & a \leftarrow g^w \bmod p \\
s \in_R \mathbb{Z}_q^*, t,\ u,\ v \in_R \mathbb{Z}_q & \xleftarrow{\quad z, a, b \quad} & b \leftarrow m^w \bmod p \\
m' \leftarrow m^s g^t,\ z' \leftarrow z^s h^t & & \\
a' \leftarrow a^u g^v,\ b' \leftarrow b^{su} m'^v a^{tu} & & \\
c' \leftarrow hash(m', z', a', b') \bmod q & & \\
c \leftarrow c' u^{-1} \bmod q & \xrightarrow{\quad c \quad} & \\
g^r \overset{?}{\equiv} ah^c \bmod p,\ m^r \overset{?}{\equiv} bz^c \bmod p & \xleftarrow{\quad r \quad} & r \leftarrow cx + w \bmod q \\
r' \leftarrow ru + v \bmod q & & \\
\sigma' \leftarrow (z', a', b', c', r') & &
\end{array}$$

Next, we explain the main ideas behind this protocol: The value $z = m^x$ can be interpreted as a signature since it is computed using the secret key of the signer. The rest of the message exchanges in the protocol serves as a proof of this fact to the receiver. The value w is a random variant of x and b a random variant of z. The verifications are done with respect to the generator g (as for Schnorr), and with respect to m. Informally, passing these verifications means that the signer knows a value x such that $h \equiv g^x$ and $z \equiv m^x$. The challenge c in the corresponding proof protocol is replaced by a hash value $hash(m', z', a', b')$ based on the approach from Fiat and Shamir (1987) as mentioned in Section 3.4.3.3. Note that the hash function is applied to all values obtained in previous steps. The actual blind signature is the tuple $\sigma := (z, a, b, c, r)$ that the receiver verifies by checking whether $g^r \equiv ah^c \bmod p$ and $m^r \equiv bz^c \bmod p$ hold. This signature is then *unblinded* by the receiver to $\sigma' := (z', a', b', c', r')$.

Security

We briefly and informally consider the main security aspects.

One-more forgery: For some integer l, polynomial in the security parameter k, an attacker cannot obtain $l+1$ valid signatures after $q \leq l$ executions of the blind signing protocol with the signer (Pointcheval and Stern 2000).

It is assumed but not proven that CP blind signature is secure against one-more forgery attack. The basic idea for this assumption is that no matter how an adversary modify the values in the blind signing protocol, it cannot compute the hash values c' except when it chooses these values in the order specified in the protocol.

Blindness: Any pair (m', σ') of message and valid signature the receiver obtains (view of the receiver) can correspond to any pair (m, σ) which the signer obtains (view of the signer) in the blind signing protocol (see Chaum and Pedersen (1993) for a proof sketch).

3.5 Commitment Schemes

A **commitment** scheme enables one to fix a message such that it cannot be changed anymore after committing while the committed value is kept secret from anyone else (Brassard, Chaum, and Crépeau (1988), Damgård (1998)). To understand the intuition behind a commitment scheme (protocol), consider the following example: Suppose, Bill secretly writes a message on a piece of paper, puts it in a safe which he locks with a key. Bill then gives the safe to George. Bill cannot change the message anymore because George has the safe, and thus, Bill is committed to the message. Moreover, George does not learn the message unless Bill opens the safe by giving him the key.[11]

3.5.1 Model

A commitment scheme for a message space M and commitment space C involves two parties, a **committer** $\mathcal{C}$ and a **receiver** $\mathcal{R}$. The main parts of a commitment scheme are the following two subprotocols: $\mathsf{ProtCom}()$ to commit to a value $m \in M$ and $\mathsf{ProtOpen}()$ to reveal the committed value, i.e., to open a commitment. We call the committed value the **content** of the commitment.

Parameter Generation: For computing and opening concrete commitments schemes, we usually require specific parameters. On input of the security parameters par_{sec}, a probabilistic polynomial generating algorithm $\mathsf{GenParCom}()$ returns the required parameters, denoted by par_{com}. These parameters are the common inputs to the commitment protocol $\mathsf{ProtCom}()$. In Section 3.5.1.1, we will discuss in more details, how and by whom these parameters can (should) be generated, and distributed.

Commit: The commit protocol is denoted by

$$(\mathcal{C} : state_{\mathcal{C}}^{C_m};\ \mathcal{R} : state_{\mathcal{R}}^{C_m}) \leftarrow \mathsf{ProtCom}(\mathcal{C} : m;\ \mathcal{R} : -; * : par_{\mathsf{com}}).$$

During this protocol the committer $\mathcal{C}$ inputs the message $m \in M$ to be committed. The output to the receiver $\mathcal{R}$ is the commitment value C_m to m and the state $state_{\mathcal{R}}^{C_m}$ for this commitment. The output to $\mathcal{C}$ is the state $state_{\mathcal{C}}^{C_m}$ (Note that in the notation we do not explicitly mention the indicator $ind \in \{failed, ok\}$ output to each party indicating whether the protocol has been correctly completed or not.)

For the concrete commitment schemes that we will use later, the committer applies a prescribed (randomized) algorithm $\mathsf{Com}()$ to the message m and par_{com}. The algorithm is denoted by:

$$(C_m, key^{C_m}) \leftarrow \mathsf{Com}(m, par_{\mathsf{com}}).$$

The output is the commitment value C_m and the opening key key^{C_m} which is a part of $state_{\mathcal{C}}^{C_m}$. Typically we have $state_{\mathcal{C}}^{C_m} := (m, key^{C_m})$.

Open: To open a commitment C_m the protocol $\mathsf{ProtOpen}()$ is executed between the (same) committer $\mathcal{C}$ and the receiver $\mathcal{R}$. The open protocol is denoted by

$$(\mathcal{C} : -;\ \mathcal{R} : m', ind) \leftarrow \mathsf{ProtOpen}(\mathcal{C} : state_{\mathcal{C}}^{C_m};\ \mathcal{R} : state_{\mathcal{R}}^{C_m}).$$

[11] We assume that there is only one key to this lock, and that the safe is tamper-resistant such that it cannot be easily broken by George.

The committer $\mathcal{C}$ inputs the state $state_{\mathcal{C}}^{C_m}$ for the commitment C_m and obtains no output. The receiver $\mathcal{R}$ also inputs the corresponding state $state_{\mathcal{R}}^{C_m}$, and obtains the output (m', ind) where $ind \in \{reject, accept\}$. More concretely, the output to $\mathcal{R}$ is either $(m', ind = accept)$ with $m' \in M$ indicating that $\mathcal{R}$ accepts the message m' as the content of the commitment C_m, or the output is $(m, ind = reject)$ for any m meaning that $\mathcal{R}$ did not accept.

For the concrete commitment schemes which we will use later, the committer $\mathcal{C}$ sends $state_{\mathcal{C}}^{C_m}$, and the receiver $\mathcal{R}$ uses a deterministic algorithm $\mathsf{VerOpen}()$ denoted as follows:

$$ind \leftarrow \mathsf{VerOpen}(state_{\mathcal{C}}^{C_m}, state_{\mathcal{R}}^{C_m})$$

where $state_{\mathcal{R}}^{C_m}$ is the commitment state the verifier has stored, and $ind \in \{true, false\}$. Typically, we have $state_{\mathcal{R}}^{C_m} := (C_m, par_{\mathsf{com}})$. The open protocol $\mathsf{ProtOpen}()$ outputs *accept* if and only if $\mathsf{VerOpen}()$ outputs *true*.

3.5.1.1 Parameter Generation Mechanisms

As mentioned in Section 3.5.1, the generation algorithm $\mathsf{GenParCom}()$ generates the parameters par_{com} required for computing and opening (concrete) commitments. The correct choice of these parameters is crucial for the security (hiding, binding) of the involved parties. To ensure correctness, we consider the following approaches:

- *Trusted parameter generation*: A trusted third party generates par_{com} (i.e., $\mathsf{GenParCom}()$ is trusted)

- *Parameter generation without trust*: The committer $\mathcal{C}$ and the receiver $\mathcal{R}$ run $\mathsf{GenParCom}()$ jointly and securely. Thereby, each party has to take care that its own security requirement is guaranteed, i.e., hiding property for the committer and binding for the receiver.

Some concrete commitment schemes, we use later, allow the parameter generating party to open any commitment (computed with the same parameters) by using a trapdoor information. We denote this information with key_{com}.

In the context of fingerprinting schemes, different commitment schemes are used as building blocks within more complicated protocols. Depending on the security requirements, we may apply either of the above approaches to generate the required parameters. Thus, we can assume that the required parameters are always correctly generated, no matter which of the approaches is used.

3.5.2 Security Requirements

The security requirements are the **binding** (committing) and **hiding** (secrecy) properties. Informally, the first one requires that a dishonest committer cannot open a commitment in *two* different ways, and the second one requires that the commitment protocol does not reveal *any* information about the content of the commitment to the receiver.

To define the hiding property formally, one can use the notion of indistinguishability (under chosen message attack) for encryption schemes defined in Section 3.4.2.1. For the binding property, one considers the success probability of an adversary algorithm which finds two messages m and m' with $m \neq m'$ such that $C := C_m = C_{m'}$. Each of these properties can be fulfilled either information-theoretically or computationally. However, it is impossible for both properties to hold information-theoretically (Brassard, Chaum, and Crépeau 1988).

Remark 3.7. For the commitment schemes as defined before (Section 3.5.1), we may define the success probability of an adversary against the hiding property as follows:

```
$P_H :=$
$\mathbf{Prob}[\mathcal{A}_2(state_{\mathcal{A}_1}, C_{m_b^*}) = b ::$
    $par_{\mathsf{com}} \leftarrow \mathsf{GenParCom}^*(k);$
    $rand \xleftarrow{\mathcal{R}} \mathcal{U};$
    $(m_0^*, m_1^*, state_{\mathcal{A}_1}) \leftarrow \mathcal{A}_1(k, par_{\mathsf{com}}, rand);$
    $b \xleftarrow{\mathcal{R}} \{0,1\};$
    $(C_{m_b^*}, key^{C_m}) \leftarrow \mathsf{Com}(m_b^*, par_{\mathsf{com}});$
  $]$
```

where $\mathsf{GenParCom}^*()$ denotes the generating algorithm of the adversary. In the definition above, the adversary consists of the two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ where the “knowledge” of $\mathcal{A}_1$ is stored in the state $state_{\mathcal{A}_1}$, and can be used by $\mathcal{A}_2$. If $P_H \leq 1/2$ for all $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathsf{GenParCom}^*(k)$, and all k, then we say that the commitment scheme is **information-theoretically hiding**. If for all $\mathcal{A}_1, \mathcal{A}_2, \mathsf{GenParCom}^*() \in \mathcal{UPTM}$, $P_H <_\infty 1/\mathsf{poly}(k)$ holds, then we say that the commitment is **computationally hiding**.

We may define the success probability of an adversary against the binding property as follows:

```
$P_B :=$
$\mathbf{Prob}[true \leftarrow \mathsf{VerOpen}(m, key^{C_m}, C, par_{\mathsf{com}}) \wedge true \leftarrow$
$\mathsf{VerOpen}(m', key^{C_{m'}}, C, par_{\mathsf{com}}) \wedge m \neq m' ::$
    $par_{\mathsf{com}} \leftarrow \mathsf{GenParCom}^*(k);$
    $rand \xleftarrow{\mathcal{R}} \mathcal{U}$
    $(m, m', key^{C_m}, key^{C_{m'}}, C) \leftarrow \mathcal{A}_1(k, par_{\mathsf{com}}, rand);$
  $]$ .
```

Similar to the hiding property the binding property can be information-theoretically or computationally. ○

3.5.3 Conventions for Commitments

For commitments we use the following notations: $C_x := com(x)$ denotes the commitment value to a message $x \in M$ and should not be confused with the algorithm $\mathsf{Com}()$ which outputs the value $com(x)$. For concrete commitment schemes, we explicitly include all parameters (e.g., par_{com}) in the notation whenever this is not clear from the context.

The bit-commitment to $x \in \mathbb{N}_0$ with $length_2(x) = l$ is defined as follows:

$$\begin{aligned} com(\mathsf{bin}(x)) &:= com([\, x_j \,]_{0 \le j \le l-1}) \\ &:= \big(com(x_{l-1}), com(x_{l-2}), \cdots, com(x_0)\big). \end{aligned}$$

For the inverse of bit commitments we write

$$\begin{aligned} com(\mathsf{bin}(x))^{-1} &:= com([\, x_j \,]_{0 \le j \le l-1})^{-1} \\ &:= \big(com(x_{l-1})^{-1}, com(x_{l-2})^{-1}, \cdots, com(x_0)^{-1}\big) \end{aligned}$$

Let $x, y \in \mathbb{N}_0$ with $length_2(x) = length_2(y) = l$. The product and division of the corresponding bit-commitments are defined as follows:

$$\begin{aligned} & com(\mathsf{bin}(x))com(\mathsf{bin}(y)) \\ &\quad := com([\, x_j \,]_{0 \le j \le l-1})com([\, y_j \,]_{0 \le j \le l-1}) \\ &\quad := \big(com(x_{l-1})com(y_{l-1}), com(x_{l-2})com(y_{l-2}), \cdots, com(x_0)com(y_0)\big) \\ & com(\mathsf{bin}(x))/com(\mathsf{bin}(y)) \\ &\quad := com([\, x_j \,]_{0 \le j \le l-1})\big(com([\, y_j \,]_{0 \le j \le l-1})\big)^{-1} \\ &\quad := \big(com(x_{l-1})com(y_{l-1})^{-1}, com(x_{l-2})com(y_{l-2})^{-1}, \cdots, com(x_0)com(y_0)^{-1}\big) \end{aligned}$$

The commitment to a vector $\vec{x} := (x_1, x_2, \cdots, x_n) \in M^n$ is defined as the commitment to all its components, i.e.,

$$com(\vec{x}) := \big(com(x_1), com(x_2), \cdots, com(x_n)\big).$$

The bit-commitment to $\vec{x} = (x_1, x_2, \cdots x_n)$ with $length_2(x_i) = l$ is defined as

$$\begin{aligned} com(\vec{\mathsf{bin}}(\vec{x})) &:= \big[com(\mathsf{bin}(x_1)), com(\mathsf{bin}(x_2)), \cdots, com(\mathsf{bin}(x_n))\big] \\ &:= [\, com(x_{i,j}) \,]_{1 \le i \le n, 1 \le j \le l} \\ &:= \begin{bmatrix} com(x_{1,1}) & com(x_{1,2}) & \cdots & com(x_{1,l}) \\ com(x_{2,1}) & com(x_{2,2}) & \cdots & com(x_{2,l}) \\ \vdots & \vdots & \ddots & \vdots \\ com(x_{n,1}) & com(x_{n,2}) & \cdots & com(x_{n,l}) \end{bmatrix} \end{aligned}$$

For the inverse of commitments to vectors we write

$$\begin{aligned} com(\vec{\mathsf{bin}}(\vec{x}))^{-1} &:= \big[com(\mathsf{bin}(x_1))^{-1}, com(\mathsf{bin}(x_2))^{-1}, \cdots, com(\mathsf{bin}(x_n))^{-1}\big] \\ &:= [\, com(x_{i,j})^{-1} \,]_{1 \le i \le n, 1 \le j \le l} \end{aligned}$$

When it is clear from the context, we omit the bounds of indices i, j, and write $\left[\ com(x_{i,j})\ \right]$.

Let $\vec{x} = (x_1, x_2, \cdots, x_n)$ and $\vec{y} = (y_1, y_2, \cdots, y_n)$ with $length_2(x_i) = length_2(y_i) = l$. For the product of the bit-commitments we write

$$\begin{aligned}
&com(\vec{\mathsf{bin}}(\vec{x}))com(\vec{\mathsf{bin}}(\vec{y}))\\
&:= \left[\ com(x_{i,j})\ \right]_{1\le i\le n, 1\le j\le l} \star \left[\ com(y_{i,j})\ \right]_{1\le i\le n, 1\le j\le l}\\
&:= \left[\ com(x_{i,j})com(y_{i,j})\ \right]_{1\le i\le n, 1\le j\le l}\\
&:= \begin{bmatrix} com(x_{1,1})com(y_{1,1}) & com(x_{1,2})com(y_{1,2}) & \cdots & com(x_{1,l})com(y_{1,l}) \\ com(x_{2,1})com(y_{2,1}) & com(x_{2,2})com(y_{2,2}) & \cdots & com(x_{2,l})com(y_{2,l}) \\ \vdots & \vdots & \ddots & \vdots \\ com(x_{n,1})com(y_{n,1}) & com(x_{n,2})com(y_{n,2}) & \cdots & com(x_{n,l})com(y_{n,l}) \end{bmatrix}
\end{aligned}$$

Similarly, we define the division

$$\begin{aligned}
&com(\vec{\mathsf{bin}}(\vec{x}))/com(\vec{\mathsf{bin}}(\vec{y}))\\
&:= \left[\ com(x_{i,j})\ \right]_{1\le i\le n, 1\le j\le l} \star\left(\left[\ com(y_{i,j})\ \right]_{1\le i\le n, 1\le j\le l} \right)^{-1}\\
&:= \left[\ com(x_{i,j})com(y_{i,j})^{-1}\ \right]_{1\le i\le n, 1\le j\le l}\\
&:= \begin{bmatrix} com(x_{1,1})com(y_{1,1})^{-1} & com(x_{1,2})com(y_{1,2})^{-1} & \cdots & com(x_{1,l})com(y_{1,l})^{-1} \\ com(x_{2,1})com(y_{2,1})^{-1} & com(x_{2,2})com(y_{2,2})^{-1} & \cdots & com(x_{2,l})com(y_{2,l})^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ com(x_{n,1})com(y_{n,1})^{-1} & com(x_{n,2})com(y_{n,2})^{-1} & \cdots & com(x_{n,l})com(y_{n,l})^{-1} \end{bmatrix}
\end{aligned}$$

3.5.4 Quadratic Residue Commitment

The **quadratic residue commitment** scheme (**QR-commitment**) enables the committer $\mathcal{C}$ to commit to a single bit. The QR-commitment scheme is a special case of the probabilistic encryption scheme proposed in Goldwasser and Micali (1984) (see also Brassard, Chaum, and Crépeau (1988)).

Parameter generation: The parameter generating algorithm is denoted by $par^{\mathsf{QR}}_{\mathsf{com}} \leftarrow \mathsf{GenParComQR}(k)$ where k is the security parameter, $par^{\mathsf{QR}}_{\mathsf{com}}{:=}n$ denotes the required commitment parameter that is a *Blum integer*. The message space is $\mathbb{Z}_2 = \{0, 1\}$ and the commitment space $\mathbb{Z}_n^*$.

Commit: The QR commitment protocol is denoted by $\mathsf{ProtComQR}(\mathcal{C} : m;\ \mathcal{R} : -;\ * : par^{\mathsf{QR}}_{\mathsf{com}})$. The individual steps of this protocol are as follows:

1. To compute the QR-commitment to a message $m \in \{0, 1\}$, the committer $\mathcal{C}$ runs the algorithm $\mathsf{ComQR}(m, par^{\mathsf{QR}}_{\mathsf{com}})$ specified as follows: It chooses $x \in_{\mathcal{R}} \mathbb{Z}_n^*$, and computes the commitment to the message $m \in \{0, 1\}$ as

$$C_m := com^{\mathsf{QR}}(m, x, par^{\mathsf{QR}}_{\mathsf{com}}) := (-1)^m x^2 \bmod n.$$

When it is clear from the context, we omit $par^{\mathsf{QR}}_{\mathsf{com}}$ in the notation, and write only $C_m := com^{\mathsf{QR}}(m, x)$.

A QR-commitment to $m = 0$ represents a quadratic residue and a QR-commitment to $m = 1$ represents a quadratic non-residue. Further, we have $state^{C_m}_{\mathcal{C}} := (m, x)$, $key^{C_m} := x$ and $state^{C_m}_{\mathcal{R}} := (C_m, n)$.

2. The committer $\mathcal{C}$ sends C_m to $\mathcal{R}$.

Open: The open protocol for QR-commitment is denoted by $\mathsf{ProtOpenQR}(\mathcal{C} : state^{C_m}_{\mathcal{C}};\ \mathcal{R} : state^{C_m}_{\mathcal{R}})$. The individual protocol steps are as follows:

1. The committer $\mathcal{C}$ sends $state^{C_m}_{\mathcal{C}} := (m, x)$ to the receiver $\mathcal{R}$.
2. The receiver $\mathcal{R}$ runs the verification algorithm $\mathsf{VerOpenQR}(state^{C_m}_{\mathcal{C}}, state^{C_m}_{\mathcal{R}})$ specified as follows: It checks whether the following holds:

$$m \in \{0, 1\}; \quad C_m, x \in \mathbb{Z}_n^*; \quad C_m = (-1)^m x^2 \bmod n.$$

If all these verifications are satisfied then $\mathsf{VerOpenQR}()$ outputs *true*, and consequently $\mathsf{ProtOpenQR}()$ outputs *accept*, otherwise *reject*.

Due to the construction of the QR-commitment scheme, any party who knows the factorization of n (i.e., p, q) is able to open any QR-commitment C computed with this modulo n, i.e., decrypt the committed message (see also Section 3.2.2). We denote this trapdoor information with $key^{\mathsf{QR}}_{\mathsf{com}}{:=}(p, q)$, and the corresponding opening algorithm with $\mathsf{OpenComQR}(key^{\mathsf{QR}}_{\mathsf{com}}; C)$.

3.5.4.1 Requirements on Parameter Generation

To guarantee the security requirements, the generated parameters must have the correct form. This can be achieved either by a trusted parameter generation, or by a verifiable and secure procedure (see also Section 3.5.1.1).

For QR-commitment scheme, as introduced above, trusted parameter generation simply means that a trusted third party generates $par^{\mathsf{QR}}_{\mathsf{com}} := n$. However, as we use QR-commitment scheme as subprotocol within other protocols, in some cases, we need to apply the secure verifiable generation approach (For instance, when we require that only one of the involved parties should be able to open the QR-commitments by knowing the factorization of n.)

One way to do this is to let the committer generate $par^{\mathsf{QR}}_{\mathsf{com}}$. This means, however, that the committer knows the factorization of n, i.e., the trapdoor information $key^{\mathsf{QR}}_{\mathsf{com}}{:=}(p, q)$. As mentioned before, knowing $key^{\mathsf{QR}}_{\mathsf{com}}$ allows the committer to open any QR-commitment computed with this n. However, there must be a way for the receiver to verify that the parameter(s) are correctly generated.[12] To ensure this, we may use either of the following approaches:

- In the opening phase, the committer $\mathcal{C}$ sends $state_{\mathcal{C}}^{C_m} := (m, x, key_{\mathsf{com}}^{\mathsf{QR}})$ to the receiver $\mathcal{R}$, and $\mathcal{R}$ verifies that n has the correct form, i.e., it is a Blum integer.

- In the parameter generation phase, the committer also proves (in *zero-knowledge*) that $-1 \in \mathsf{QNR}_n$ (see Section 3.6.4 on zero-knowledge proof systems), e.g., by applying the proof techniques from Goldwasser and Micali (1984).

  Another way to show the correctness of n is as follows: First, apply the efficient proof protocols of van de Graaf and Peralta (1988) to show that n has the form $p^r q^s$ where r and s are odd integers, $p, q \in \mathbb{P}$, $p \neq q$ and $p \equiv q \equiv 3 \bmod 4$. Then use a protocol from Boyar, Friedl, and Lund (1991) to show that a given number n is square free, i.e., there is no prime p with $p|n$ and $p^2|n$. Thus, if both properties are shown for n then $n = pq$ with $p \equiv q \equiv 3 \bmod 4$ hold. Note that the requirement "n is a Blum integer" is sufficient for the verifier, and it is in committer's own interest (hiding property) to construct n as specified.

Remark 3.8. In the context of fingerprinting schemes we will apply the second approach since the same $par_{\mathsf{com}}^{\mathsf{QR}}$ is used in all subprotocols throughout the whole protocol, and the receiver should not be able to open the commitment by itself, i.e., it should not know the factorization of n. ○

3.5.4.2 Properties

Next, we consider the main properties of the QR-commitment scheme.

- *Security:* The quadratic residuosity commitment scheme $(\mathsf{GenParComQR}(), \mathsf{ProtComQR}(), \mathsf{ProtOpenQR}())$, as described above, is information-theoretically binding, and it is hiding under the *Quadratic Residuosity Assumption* (QRA) defined in Definition 3.4 (see also Goldwasser and Micali (1984) and Brassard, Chaum, and Crépeau (1988)).

- *Homomorphic property:* The QR-commitment is homomorphic with respect to XOR (addition mod 2) operation. More concretely, given the commitments $C_{m_1} := com^{\mathsf{QR}}(m_1, x_1, par_{\mathsf{com}}^{\mathsf{QR}})$ and $C_{m_2} :=$

[12] A cheating committer can choose the prime numbers p, q such that it can break the binding property: Assume $p \equiv q \equiv 1 \mod 4$ then -1 is a square modulo $n = pq$. Since $\mathcal{C}$ knows the factorization, it can compute square roots u of -1. For a commitment to 0 it sends $com^{\mathsf{QR}}(0) := x^2 \bmod n$ but it can open it as 1 in opening phase by sending $ux \bmod n$ instead of x.

$com^{\mathsf{QR}}(m_2, x_2, par^{\mathsf{QR}}_{\mathsf{com}})$ with $m_1, m_2 \in \mathbb{Z}_2$ and $C_{m_1}, C_{m_2} \in \mathbb{Z}_n^*$ the following holds:

$$\begin{aligned} C_{m_1}C_{m_2} &= com^{\mathsf{QR}}(m_1, x_1, par^{\mathsf{QR}}_{\mathsf{com}})com^{\mathsf{QR}}(m_2, x_2, par^{\mathsf{QR}}_{\mathsf{com}}) \\ &= com^{\mathsf{QR}}(m_1 \oplus m_2, x_1x_2, par^{\mathsf{QR}}_{\mathsf{com}}). \end{aligned}$$

Note that if the correctness of $n := par^{\mathsf{QR}}_{\mathsf{com}}$ is shown without revealing its factors (see discussion above) then the product commitment $C_* = C_{m_1}C_{m_2}$ can be opened to $m_1 \oplus m_2$ (i.e., $C_* = (-1)^{m_1 \oplus m_2}(x_1x_2)^2 \bmod n$) without revealing additional information about the contents of each of the commitments C_{m_1} and C_{m_2}.

- *Equality of contents:* As a consequence of homomorphic property, we can test whether two given QR-commitments have the same content without knowing each individual content: Given the commitments $C_{m_1} := com^{\mathsf{QR}}(m_1, x_1, par^{\mathsf{QR}}_{\mathsf{com}})$ and $C_{m_2} := com^{\mathsf{QR}}(m_2, x_2, par^{\mathsf{QR}}_{\mathsf{com}})$ compute and open the product $C_* = C_{m_1}C_{m_2}$. Then the following holds:

$$C_* = com^{\mathsf{QR}}(0, x_1x_2, par^{\mathsf{QR}}_{\mathsf{com}}) \Longleftrightarrow m_1 = m_2.$$

- *Inverse of QR-commitment:* The inverse of a given QR-commitment to a value m is again a commitment to the same value m. We formulate this property in the following lemma.

Lemma 3.1 *Let* $C_m := com^{\mathsf{QR}}(m, x, par^{\mathsf{QR}}_{\mathsf{com}}) := (-1)^m x^2 \bmod n$. *Then the following holds*

$$(C_m)^{-1} = com^{\mathsf{QR}}(m, y, par^{\mathsf{QR}}_{\mathsf{com}}) = (-1)^m y^2 \bmod n$$

where $y \in \mathbb{Z}_n^*$. □

Proof. Consider the inverse of the given commitment:

$$\begin{aligned} C'_m := C_m^{-1} &= (-1)^{-m}(x^2)^{-1} \bmod n \\ &= (-1)^{-m}(x^{-1})^2 \bmod n \\ &= (-1)^{-m}(y)^2 \bmod n \end{aligned}$$

where $y := x^{-1} \in \mathbb{Z}_n^*$. If $m = 0$ then $C'_m = y^2 \bmod n$ is quadratic residue, and represents a QR-commitment to 0. If $m = 1$ then we $(-1)^{-1} = -1 \bmod n$, and $C'_m = -y^2 \bmod n$ is a quadratic non-residue representing a QR-commitment to 1. ■

- *Blinding:* Given a QR-commitment C_m to a message m, one can generate a random and independent commitment C'_m to the same message m by multiplying C_m with a random commitment C_0 to $m' = 0$. We formulate this property in the following lemma.

Lemma 3.2 *Let $C_m := (-1)^m x^2 \bmod n$ be a given QR-commitment to the message $m \in M$, and $C_0 := x'^2 \bmod n$ a QR-commitment to zero where $x' \in_{\mathcal{R}} \mathbb{Z}_n^*$. Then $C'_m := C_0 C_m$ is a random and independent QR-commitment to m.* □

Proof. Consider the product of the commitments:

$$\begin{aligned} C'_m &:= C_0 C_m = x'^2 (-1)^m x^2 \bmod n \\ &= (-1)^m (x'x)^2 \bmod n = (-1)^m y^2 \bmod n \end{aligned}$$

where $y :\equiv x'x \bmod n$. The value y is randomly and uniformly distributed over $\mathbb{Z}_n^*$ since $x \in \mathbb{Z}_n^*$ is fixed and x' is randomly and uniformly selected value from $\mathbb{Z}_n^*$. Note that multiplication with a fixed value $x \in \mathbb{Z}_n^*$ is a bijection on $\mathbb{Z}_n^*$. ■

3.5.5 Discrete Logarithm Commitment

The **discrete logarithm commitment** scheme (**DL-commitment**) enables the committer $\mathcal{C}$ to commit to messages $m \in M := \{0,1\}^l$ of length l (see e.g., Boyar, Kurtz, and Krentel (1990), Pedersen (1992))

Parameter generation: The parameter generating algorithm is denoted by $par^{\mathsf{DL}}_{\mathsf{com}} \leftarrow \mathsf{GenParComDL}(k, k')$ where k, k' are security parameters, $par^{\mathsf{DL}}_{\mathsf{com}} := (p, q, g, h)$ denotes the required parameters with $p, q \in_{\mathcal{R}} \mathbb{P}$, $length_2(q) < l$, and $q|(p-1)$.[13] The parameters $g, h \neq 1$ denote two generators of the (unique) multiplicative subgroup $\mathbb{Z}^*_{p/q}$ of $\mathbb{Z}_p^*$ of order q (see also Section 2.1.7). The commitment space is $\mathbb{Z}_p^*$.

Commit: The DL commitment protocol is denoted by $\mathsf{ProtComDL}(\mathcal{C} : m;\ \mathcal{R} : -;\ * : par^{\mathsf{DL}}_{\mathsf{com}})$. The individual steps of this protocol are as follows:

1. To compute the QR-commitment to a message in M, the committer $\mathcal{C}$ first represents the message as an integer m in $\mathbb{Z}_q$ by using an appropriate encoding function (see Section 3.1).
2. The committer $\mathcal{C}$ runs the algorithm $\mathsf{ComDL}(m, par^{\mathsf{DL}}_{\mathsf{com}})$ specified as follows: It randomly and uniformly selects $r \in_{\mathcal{R}} \mathbb{Z}_q$, and computes the commitment to the message m as

$$C_m := com^{\mathsf{DL}}(m, r, par^{\mathsf{DL}}_{\mathsf{com}}) := g^m h^r \bmod p.$$

   When it is clear from the context, we omit $par^{\mathsf{QR}}_{\mathsf{com}}$ in the notation and write only $com^{\mathsf{DL}}(m, r)$. The states of the committer and the receiver for the commitment C_m are $state^{C_m}_{\mathcal{C}} := (m, r)$ and $state^{C_m}_{\mathcal{R}} := (C_m, par^{\mathsf{DL}}_{\mathsf{com}})$ where $key^{C_m} := r$

[13] More concretely, $length_2(q) = f(k')$, $length_2(p) = g(k)$ where f, g are polynomials.

3. The committer $\mathcal{C}$ sends the commitment C_m to $\mathcal{R}$.

Open: The open protocol is denoted by $\mathsf{ProtOpenDL}(\mathcal{C} : \mathit{state}_{\mathcal{C}}^{C_m}; \mathcal{R} : \mathit{state}_{\mathcal{R}}^{C_m})$. The individual protocol steps are as follows:

1. The committer $\mathcal{C}$ sends $\mathit{state}_{\mathcal{C}}^{C_m} := (m, r)$ to $\mathcal{R}$.
2. The receiver $\mathcal{R}$ runs the verification algorithm $\mathsf{VerOpenDL}(\mathit{state}_{\mathcal{C}}^{C_m}, \mathit{state}_{\mathcal{R}}^{C_m})$ specified as follows: It verifies that the following holds:

$$m, r \in \mathbb{Z}_q; \quad C_m \in \mathbb{Z}_p^*; \quad C_m := g^m h^r \bmod p.$$

If all these verifications are satisfied then $\mathsf{VerOpenDL}()$ outputs *true*, and consequently $\mathsf{ProtOpenDL}()$ outputs $\mathit{ind} = \mathit{accept}$, otherwise *reject*.

3.5.5.1 Requirements on Parameter Generation

To guarantee the security requirements, the parameters must have the correct form. This can be achieved either by trusted parameter generation, or by a verifiable and secure procedure (see also Section 3.5.1.1).

For DL-commitment scheme, as introduced above, trusted parameter generation simply means that a trusted third party generates $\mathit{par}_{\mathsf{com}}^{\mathsf{DL}} := (p, q, g, h)$, e.g., at global system setup as global parameters. However, if this is not desired, we may approach as follows: We let the receiver $\mathcal{R}$ generate $\mathit{par}_{\mathsf{com}}^{\mathsf{DL}} := (p, q, g, h)$, and let the committer $\mathcal{C}$ verify them in the committing phase, i.e., $\mathcal{C}$ additionally checks whether p, q are primes, $g, h \in \mathbb{Z}_{p/q}^*$ and $h \not\equiv 1$. Note that the committer $\mathcal{C}$ should not choose these parameters, since a cheating committer may choose g and h such that it knows the relative discrete logarithm between them, being able to break the binding property of the DL-commitment scheme.

3.5.5.2 Properties

Next, we consider the main properties of the DL-commitment scheme.

- *Security:* The DL-commitment scheme denoted by the tuple $(\mathsf{GenParComDL}(), \mathsf{ProtComDL}(), \mathsf{ProtOpenDL}())$, as described above, is information-theoretically hiding, and it is binding under the *Discrete Logarithm* assumption (Pedersen 1992).

- *Homomorphic property:* Let $r, r_1, r_2, e \in \mathbb{Z}_q$, and let $C_m := \mathit{com}^{\mathsf{DL}}(m, r, \mathit{par}_{\mathsf{com}}^{\mathsf{DL}})$, $C_{m_1} := \mathit{com}^{\mathsf{DL}}(m_1, r_1, \mathit{par}_{\mathsf{com}}^{\mathsf{DL}})$, $C_{m_2} := \mathit{com}^{\mathsf{DL}}(m_2, r_2, \mathit{par}_{\mathsf{com}}^{\mathsf{DL}})$ be DL-commitments to the values $m, m_1, m_2 \in \mathbb{Z}_q$. The following relations hold:

$$\begin{aligned} C_{m_2} C_{m_2} &= \mathit{com}^{\mathsf{DL}}(m_1 + m_2, r_1 + r_2, \mathit{par}_{\mathsf{com}}^{\mathsf{DL}}) \\ (C_m)^e &= \mathit{com}^{\mathsf{DL}}(em, er, \mathit{par}_{\mathsf{com}}^{\mathsf{DL}}). \end{aligned}$$

Using these properties, one can commit to $m \in \mathbb{Z}_q$ using commitments to its bits as follows: Let $m = \sum_{j=0}^{l-1} m_j 2^j$ with $m_j \in \{0,1\}$ (the dyadic representation of m), and $com^{\mathsf{DL}}(m_j, r_j, par^{\mathsf{DL}}_{\mathsf{com}})$ be commitments to the bits m_j where $r_j \in_{\mathcal{R}} \mathbb{Z}_q$. Then we can write

$$\prod_{j=0}^{l-1} \left(com^{\mathsf{DL}}(m_j, r_j, par^{\mathsf{DL}}_{\mathsf{com}})\right)^{2^j} = g^m h^r = com^{\mathsf{DL}}(m, r, par^{\mathsf{DL}}_{\mathsf{com}}) \bmod p$$

with $r = \sum_{j=0}^{l-1} r_j 2^j \bmod q$.

3.5.6 Okamoto-Uchiyama Commitment

Any (semantically) secure asymmetric encryption scheme (Section 3.4) can be a candidate for a commitment scheme. We briefly explain this: In commit phase the committer $\mathcal{C}$ generates a key pair $key := (sk, pk) \leftarrow \mathsf{GenKeyEnc}(k)$ where k is a security parameter. $\mathcal{C}$ encrypts the message $m \in M$, i.e., $enc \leftarrow \mathsf{Encrypt}(pk; m)$, and sends the commitment $C_m = (enc, pk)$ to the receiver $\mathcal{R}$. In the opening phase, $\mathcal{C}$ sends $(m, state^{key}_{\mathcal{C}})$ to $\mathcal{R}$ where $state^{key}_{\mathcal{C}}$ contains the random choices made by $\mathsf{GenKeyEnc}()$ to generate key. $\mathcal{R}$ verifies as follows: It regenerates the key pair (sk', pk') using $state^{key}_{\mathcal{C}}$, and runs the encryption algorithm on m, i.e., $enc' \leftarrow \mathsf{Encrypt}(pk'; m)$. $\mathcal{R}$ accepts if and only if $m \in M$, $enc' = enc$ and $pk' = pk$.

The commitment scheme we obtain from a (semantically) secure asymmetric encryption scheme is information-theoretically binding and computationally hiding.

The encryption scheme to be used as a commitment scheme, is the Okamoto-Uchiyama (OU) encryption scheme explained in Section 3.4.2.3.

Parameter generation: The required parameters are $par^{\mathsf{OU}}_{\mathsf{com}} := (n, g, h, k)$ as defined in Section 3.4.2.3.

Commit: The OU commitment protocol is denoted by $\mathsf{ProtComOU}(\mathcal{C} : m;\ \mathcal{R} : -;\ * : par^{\mathsf{OU}}_{\mathsf{com}})$. The individual steps of this protocol are as follows:

1. To compute the OU-commitment to a message in M, the committer $\mathcal{C}$ first represents the message as an integer m in $\mathbb{Z}_{2^{k-1}}$ using an appropriate encoding function (see Section 3.1).
2. The committer $\mathcal{C}$ runs the algorithm $\mathsf{ComOU}(m, par^{\mathsf{OU}}_{\mathsf{com}})$ that corresponds to the encryption part of OU encryption scheme. This algorithm selects $r \in_R \mathbb{Z}_n$ and computes the commitment to m as

$$C_m := com^{\mathsf{OU}}(m, r, par^{\mathsf{OU}}_{\mathsf{com}}) = g^m h^r \bmod n.$$

The states of the committer and the receiver are $state^{C_m}_{\mathcal{C}} := (m, r)$ and $state^{C_m}_{\mathcal{R}} := (C_m, par^{\mathsf{OU}}_{\mathsf{com}})$.

3. The committer $\mathcal{C}$ sends the commitment C_m to $\mathcal{R}$.

Open: The open protocol is denoted by $\mathsf{ProtOpenOU}(\mathcal{C} : state_{\mathcal{C}}^{C_m}; \mathcal{R} : state_{\mathcal{R}}^{C_m})$. The individual steps of this protocol are as follows:

1. The committer $\mathcal{C}$ sends (m, r) to $\mathcal{R}$.
2. The receiver $\mathcal{R}$ runs the verification algorithm $\mathsf{VerOpenOU}(state_{\mathcal{C}}^{C_m}, state_{\mathcal{R}}^{C_m})$ specified as follows: It verifies that the following holds:

$$0 < m < 2^{k-1},\ C_m \in \mathbb{Z}_n,\ r \in \mathbb{Z}_n,\ C_m = g^m h^r \bmod n.$$

   If all these verifications are satisfied then $\mathsf{VerOpenOU}()$ outputs *true*, and consequently $\mathsf{ProtOpenOU}()$ outputs *accept*, otherwise *reject*.

3.5.6.1 Requirements on Parameter Generation

As for the previous commitment schemes, we may generate the required parameters with or without trusted generation.

For OU commitment scheme, we have a similar situation as for the QR commitment scheme (see the corresponding part in Section 3.5.4): We let the committer generate $par_{\mathsf{com}}^{\mathsf{OU}}$. This means that the committer knows the factorization of n, denoted by $key_{\mathsf{com}}^{\mathsf{OU}} := (p, q)$, that allows the committer to open any OU-commitment computed with n.

To ensure their correctness, we may use either of the following approaches:

- In the opening phase the committer $\mathcal{C}$ discloses the random coins of the key generating algorithm $\mathsf{GenKeyOU}(k)$, and the receiver $\mathcal{R}$ verifies correctness, as mentioned at the beginning of this section.

- The committer $\mathcal{C}$ proves in *zero-knowledge* (see Section 3.6.4) to the receiver $\mathcal{R}$ that the parameters $n = p^2 q$ and g have the correct form. One way to do this is to apply the techniques introduced by Camenisch and Michels (1999b): They present efficient zero-knowledge proofs for the following tasks: (i) Proving that a committed number is a prime, and (ii) proving that certain modular relations hold between committed numbers. More precisely, the latter means that given commitments $com(a), com(b), com(c)$ and $com(d)$, there are explicit and efficient (statistically) zero-knowledge protocols for proving that $c \equiv a{*}b \bmod d$ where the operation $* \in \{+, \times, exp\}$ can be the sum, multiplication and exponentiation in $\mathbb{Z}$.

  Hence, the committer can commit to p, p^2 and q, and combine these protocols to prove that p, q are primes with $n = p^2 q$ and $g^{p(p-1)} \equiv 1 \bmod p^2$.[14]

[14] It must also be proven that the content of the commitment hiding p^2 is the square of the content of the commitment hiding p.

Remark 3.9. In the context of fingerprinting schemes, we will apply the second approach since the same parameters $par_{\mathsf{com}}^{\mathsf{OU}}$ are used (in subprotocols) throughout the whole protocol, and the receiver should not be able to open the commitment, i.e., it should not know the factorization of n. Thus, we can assume that the output of $\mathsf{GenParOU}()$ is always trusted. ∘

3.5.6.2 Properties

Next, we consider the main properties of OU-commitment:

- *Security:* The OU-commitment scheme denoted by the tuple $(\mathsf{ProtComOU}(), \mathsf{GenParOU}(), \mathsf{ProtOpenOU}())$, and described above, is information-theoretically binding, and it is hiding under the p-Subgroup Assumption (see also Section 3.4.2.3).

- *Homomorphic property:* It is similar to that of DL-commitments (see Section 3.5.5) except that for the content m of a commitment must hold $m < p$. Let $e \in \mathbb{Z}$ and the commitments $C_x := com^{\mathsf{OU}}(x, r_x, par_{\mathsf{com}}^{\mathsf{OU}})$ and $C_y := com^{\mathsf{OU}}(y, r_y, par_{\mathsf{com}}^{\mathsf{OU}})$ be given where $x + y < p$ and $xe < p$ hold. Then the following relations hold:

$$\begin{aligned} C_x C_y &= com^{\mathsf{DL}}(x + y, r_x + r_y, par_{\mathsf{com}}^{\mathsf{OU}}) \\ (C_x)^e &= com^{\mathsf{DL}}(ex, er_x, par_{\mathsf{com}}^{\mathsf{OU}}). \end{aligned}$$

3.6 Interactive Proofs and Zero-Knowledge

In this section, we consider interactive proof systems and their main goals, namely, **proof of language membership** and **proof of knowledge**. Proof systems are widely used in cryptographic systems. We will consider the zero-knowledge aspect which is an important security requirement on the proof systems when deployed in cryptographic systems. There is a large body of literature written on zero-knowledge proof systems. Some major contributions are from Goldwasser, Micali, and Rackoff (1985), Goldreich, Micali, and Wigderson (1991), Bellare and Goldreich (1993) and Goldreich and Krawczyk (1996). Some useful manuscripts are Pfitzmann (1999) and Bellare and Goldwasser (2001). A comprehensive theoretical work on this topic can be found in Goldreich (2001b).

3.6.1 Introduction

Suppose someone, say George, claims to be capable of telling the origin of different wines from a given (agreed upon) palette of wines! However, he is not willing to disclose his method of recognizing the wines to us. To verify George's claim, we may run the following protocol with him: We choose randomly and secretly a wine sort from the given palette and *challenge*

George by asking him the origin of this wine.[15] Now, George takes the wine, and may disappear for some *reasonable* (or agreed upon) time. As soon as he is finished, he gives us his *response* by telling us the origin of the chosen wine. We will *accept* only if the answer is *correct*, otherwise we will *reject*. Now, we may have some doubt on George's "wine culture" and may run this protocol for several times or we may randomly select a subset of (e.g., the half of) the wines in the palette and verify whether he can tell their origin. In either case, we may accept if and only if all responses to our challenges are correct, or may follow another strategy for accepting or rejecting the proof (e.g., we may tolerate certain number of failures.) This simple informal example represents the soul of many **interactive proof systems** used in cryptographic applications. An interactive proof system is a 2-party protocol between two interactive algorithms, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where $\mathcal{P}$ proves a statement to $\mathcal{V}$. This statement is represented by the **common input** x. Each party might have private inputs called **auxiliary input** $aux_X, X \in \{\mathcal{P}, \mathcal{V}\}$. Further, each party has access to a source of randomness, i.e., a random tape $rand_X$. The computational task of $\mathcal{P}$ is to generate a proof, and that of $\mathcal{V}$ is to verify its validity.

The fundamental properties of an interactive proof system are **completeness** and **soundness**.

- **Completeness:** A correct prover $\mathcal{P}$ can prove all correct statements (assertions) to a correct verifier $\mathcal{V}$. For our purposes, both parties should be efficient (i.e., their running time is polynomial in the length of the common input which should be a function of security parameter(s), see also Section 2.1.3)

- **Soundness:** A cheating prover $\mathcal{P}^*$ cannot prove a wrong statement to an honest verifier, i.e., a verification procedure cannot be faked such that $\mathcal{V}$ accepts false statements.[16]

The main types of proof systems can be classified as follows: The proof of **language membership** and the proof of **knowledge**. In the former class a language L and a string x is given where x is defined over an alphabet Σ, and is known to both parties. The objective of the proof system is for $\mathcal{P}$ to prove to $\mathcal{V}$ that x lies in L. In the latter class a binary relation R and a string x are given. The objective of the proof system for $\mathcal{P}$ is to prove to $\mathcal{V}$ that it "knows" a string w called **witness** such that $(w, x) \in R$. We will take a closer look at these topics in the next sections.

As mentioned before, the interaction between the interactive algorithms $\mathcal{P}$ and $\mathcal{V}$ consists of several steps, starting with an initialization. The initial state of a party $X \in \{\mathcal{P}, \mathcal{V}\}$ is assumed to be the tuple

[15] The wine sorts are filled in glasses with exactly the same material and shape.

[16] The proof is not always absolute but rather probabilistic, i.e., there may be a tolerated success probability for a cheating prover.

$state_X^0 := (x, rand_X, input_X)$ where x is the common input(s), $rand_X$ is the content of X's random tape, and $input_X$ is its input.[17] In each step i, only one message is sent from or received by a party. Each party X takes the message $mess_X^i$ it receives and its current state $state_X^i$, does some computations, and outputs a message $mess_X^{i+1}$ (to be sent) and the next state $state_X^{i+1}$. At the end, $\mathcal{V}$ outputs an indicator $ind \in \{accept, reject\}$ indicating whether it accepts or rejects. Most interactive proof protocols we are concerned with in this work are of the following **challenge and response** form: Given a common input, the protocol consists of three moves. The prover $\mathcal{P}$ starts by sending a message to the verifier $\mathcal{V}$. Then $\mathcal{V}$ sends a challenge to $\mathcal{P}$, and in the last move $\mathcal{P}$ responds depending on the challenge. Finally, $\mathcal{V}$ verifies this response, and accepts or rejects.

3.6.2 Proof of Language Membership

The main components of a proof system are as follows:

Parameters: The parameters are a language L, and the set of all security parameters par_{sec} (Note that security parameters concern different aspects, e.g., the probability representing the degree of the confidence in a proof system, or the boundary for the success probability in computational problems.)

Generating algorithm: A probabilistic polynomial-time algorithm $\mathsf{Gen}^L()$ for the prover $\mathcal{P}$. On input par_{sec}, it outputs pairs $(x, aux) \in L$.

Protocol: A two-party protocol described by

$$(\mathcal{P} : -;\ \mathcal{V} : ind) \leftarrow \mathsf{Prove}(\mathcal{P} : aux;\ \mathcal{V} : -;\ * : x, par_{sec})$$

The random variable corresponding to verifier's output ind in this protocol is denoted by $prove_X^{\mathcal{V}}()$ where X may be an honest prover $\mathcal{P}$ or a cheating one $\mathcal{P}^*$. The **accepting probability** is defined as

$$\begin{aligned} &P_{accept}(aux, x, par_{sec}) \\ &\quad := \ \mathbf{Prob}[ind = true :: ind \leftarrow prove_{\mathcal{P}}^{\mathcal{V}}(aux, x, par_{sec})] \end{aligned}$$

where the probability function $\mathbf{Prob}[\cdot]$ is induced by the random tapes $rand_{\mathcal{P}}$ and $rand_{\mathcal{V}}$.

An interactive proof system for language membership is defined as follows:

Definition 3.6 Let L be a language, par_{sec} be the set of security parameters, and $\gamma_s \in par_{sec}$. Further, let $\mathsf{Gen}^L()$ be a generating algorithm, and $\mathcal{P}$

[17]Note that the random tape is the only source of randomness of the corresponding algorithm (see also Section 2.1.3)

and $\mathcal{V}$ be interactive algorithms. We call a proof system an interactive proof system for **language membership** providing **information-theoretical soundness** over L, if the following properties hold:

1. *Correct generation*: For all valid parameters par_{sec}
$$(x, aux) \leftarrow \mathsf{Gen}^L(par_{sec}) \wedge x \in L.$$
2. *Completeness*: For all valid parameters par_{sec}, and all $(x, aux) \in [\mathsf{Gen}^L(par_{sec})]$, an honest prover can always convince an honest verifier, i.e.,
$$\begin{aligned} &P_{accept}(aux, x, par_{sec}) \\ &\quad := \mathbf{Prob}[ind = true :: ind \leftarrow prove^{\mathcal{V}}_{\mathcal{P}}(aux, x, par_{sec})] = 1. \end{aligned}$$
3. *Soundness*: For all interactive algorithms $\mathcal{P}^*$, for all valid parameters par_{sec}, for all $x \notin L$ and for all $aux \in \{0,1\}^*$
$$\begin{aligned} &P_{accept}(aux, x, par_{sec}) \\ &\quad = \mathbf{Prob}[ind = true :: ind \leftarrow prove^{\mathcal{V}}_{\mathcal{P}^*}(aux, x, par_{sec})] \leq 2^{-\gamma_s}. \end{aligned}$$

◇

We denote an interactive proof system with $(\mathcal{P}, \mathcal{V})$.

Remark 3.10. In a basic proof system, also called **atomic proof**, there may be a relatively large success probability for a cheating prover to convince the verifier. To achieve a higher degree of confidence in the soundness, the proof is iterated (see also Goldreich (2001b) for more details) Let p be the success probability of a cheating prover $\mathcal{P}^*$ in the atomic proof, and ϵ_s be the tolerated success probability for $\mathcal{P}^*$. Then the number of iterations is $\gamma_s = \log_p(\epsilon_s)$. As an example, consider an atomic proof system where $\mathcal{P}^*$ has the success probability $p = 1/2$ in each iteration. By iterating the atomic proof for, say $\gamma_s = 40$ times, we achieve the confidence of $1 - 1/2^{40}$. ○

3.6.3 Proof of Knowledge

As mentioned in Section 3.6.1, interactive proof systems are used in many cryptographic protocols where a party proves to another party that she "knows" something (e.g. a secret). In the example of the previous section, this secret was the method of recognizing wines. Another example is knowing the square root of a quadratic residue modulo a hard-to-factor integer, or the discrete logarithm of an integer with respect to a certain basis.

Now, what does it mean when we say an algorithm $\mathcal{P}^*$ "knows" something? Roughly speaking, this means that we can use $\mathcal{P}^*$ as a black box and can "efficiently" modify it such that we can successfully compute (extract) this "something" from it. More precisely, there exists a polynomial-time algorithm called **knowledge extractor** $\mathcal{E}$ which, given a common input x and oracle access to $\mathcal{P}^*$ (as black-box), can compute a witness w such that (w, x) belong to a well-defined relation R, i.e., $(w, x) \in R$. The extractor can interact with $\mathcal{P}^*$ and keep track of $\mathcal{P}^*$'s state. The knowledge extractor is allowed to initialize the state of $\mathcal{P}^*$, and to reset $\mathcal{P}^*$ to a previous state including its random tape $rand_{\mathcal{P}^*}$. This can be seen as a "privilege" given to the extractor since the verifier $\mathcal{V}$ in the real protocol cannot reset the prover – yet, any real cheater having a successful prover algorithm could do it. The success probability of $\mathcal{E}$ is related to that of $\mathcal{P}^*$.

Note that proof of language membership or knowledge is a property of the prover $\mathcal{P}^*$. Thus, when showing this property for a proof system, we consider cheating provers and honest verifiers.

The components of a proof system for proof of knowledge are defined similar to that of proof of language membership described in Section 3.6.2. The following definition for proof of knowledge relies on the definition given in Bellare and Goldreich (1993).

Definition 3.7 Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation, $L_R := \{x | \exists w : (w, x) \in R\}$, par_{sec} be the set of security parameters, $\mathsf{Gen}^{L_R}()$ be a generating algorithm, and $\mathcal{P}$ and $\mathcal{V}$ be interactive algorithms. We call a proof system an interactive **proof of knowledge** if the following properties hold:

1. *Correct generation*: For all valid parameters par_{sec} it holds
$$R \ni (x, aux) \leftarrow \mathsf{Gen}^{L_R}(par_{sec}).$$
2. *Completeness*: For all valid parameters par_{sec} and $(aux, x) \in R$ the accepting probability is
$$\begin{aligned} &P_{accept}(aux, x, par_{sec}) \\ &\quad := \mathbf{Prob}[ind = true :: ind \leftarrow prove^{\mathcal{V}}_{\mathcal{P}}(aux, x, par_{sec})] = 1. \end{aligned}$$
3. *Weak soundness*: There exist a polynomial poly and a probabilistic algorithm $\mathcal{E}$ (the knowledge extractor) with oracle access such that for all probabilistic polynomial-time interactive algorithms $\mathcal{P}^*$, for all strings aux and for all $x \in L_R$, on input (par_{sec}, x) and oracle access to $\mathcal{P}^*$, the knowledge extractor $\mathcal{E}$ outputs a witness w with $(w, x) \in R$ in expected time
$$\frac{\mathsf{poly}(k)}{P^*_{accept}(aux, x, par_{sec})}$$
where $k \in par_{sec}$ is the computational security parameter and
$$\begin{aligned} &P^*_{accept}(aux, x, par_{sec}) \\ &\quad := \mathbf{Prob}[ind = true :: ind \leftarrow prove^{\mathcal{V}}_{\mathcal{P}^*}(aux, x, par_{sec})] \end{aligned}$$

is the probability that $\mathcal{V}$ accepts in interaction with $\mathcal{P}^*$.

$\diamond$

Strong soundness: In Definition 3.7, the soundness requirement is called *weak* since the definition only considers strings $x \in L_R$.[18] For *strong soundness* we require, in addition to weak soundness, that the proof protocol $\mathsf{Prove}()$, with an appropriate generating algorithm, is a proof of language membership for L_R.[19]

Remark 3.11. The proof of knowledge is seen as a security condition and the prover as an adversary trying to convince the verifier to accept. The acceptance of the verifier should be a measure for prover's knowledge of the witness. Therefore, one is interested to relate the success probability of the extractor to the accepting probability of the verifier. In this context, one has to handle provers $\mathcal{P}^*$ who have a small success probability. Provers not knowing the witness should have negligible success probability. Thus, the extractor $\mathcal{E}$ should be successful for all provers with success probability at least the inverse of any polynomial. However, the smaller $\mathcal{P}^*$'s success probability, the smaller is the success probability of $\mathcal{E}$ in extracting the witness. Definition 3.7 captures this issue by letting $\mathcal{E}$'s running time be inversely proportional to $\mathcal{P}^*$'s success probability.[20] $\circ$

3.6.4 Zero-Knowledge

3.6.4.1 Introduction

We first informally explain the intuition behind the zero-knowledge aspect. In Section 3.6.1 we gave an example where George claims to be able to distinguish wines from a given wine palette, and we discussed a proof system (protocol) for verifying George's claim. Loosely speaking, if George does not want the proof protocol to reveal *any* information about his method of distinguishing wines, then the proof system should have the **zero-knowledge** property. It is a security requirement that the prover can fulfill in an interactive proof. The zero-knowledge property requires the proof system to reveal "no knowledge" to the verifier, except the fact that the assertion is valid, which is itself no knowledge as an honest prover can always prove

[18]For example, if $L_R := \{x | \exists w : w^2 \equiv x \mod n\}$ the definition does not consider the case where a cheating prover can convince a verifier that it has knowledge of a square root w of $x \notin \mathsf{QR}_n$. However, note that weak soundness suffices for many applications.

[19]Note that a straight forward extension of weak soundness to all x would not work, if there is an error probability, because in this case, the knowledge extractor cannot find any witness.

[20]Goldreich (2001b) gives also a stringent definition for proof of knowledge called "strong proof of knowledge" where the knowledge extractor is required to run strictly in polynomial-time.

this. In other words, the verifier should gain "no new knowledge" from the conversation with the prover (i.e., from the protocol-run).[21] With "no new knowledge" for the verifier, we mean that the verifier could easily compute its **view** of the proof by only having the common input x and no interaction with the prover! Informally, the view of the verifier is what it sees and does during the proof.[22] However, this may seem contradictory since on the one hand, the verifier should gain no knowledge about the secret, and on the other hand, it should be able to compute its view of the proof. As we have seen in our wine example, such proof systems usually consist of a conversation in form of a sequence of challenge and response pairs between the prover and the verifier. In the real proof, if the prover knows the secret, it can answer to all challenges correctly, and thus, there is a sequence of challenge-response pairs with certain order where the verifier accepts. This sequence is also called **conversation transcript**. However, if a protocol is zero-knowledge then, even without knowing the secret (and without interaction with the prover), one can construct pairs of challenges and corresponding responses such that the resulting (constructed) transcript "looks" like the real proof transcript. In our example, we may record the conversation on a videotape once between the verifier and the real prover, and once with a fake prover who does not know how to distinguish the wines. In the latter case, we agree with the fake prover in advance what answers he should give in each iteration, or we let him answer and cut out the wrong ones from the tape. Now, this faked recording looks exactly like the original proof, and thus, it cannot testify the correctness of the proof. Hence, the verifier cannot use this recording to convince anyone else!

3.6.4.2 Definition

The zero-knowledge property is a security requirement defined to protect provers and should be guaranteed as long as the provers follow the protocol. Thus, zero-knowledge considers only honest provers whereas the verifier is in general an adversary $\mathcal{V}^*$ who wants to extract knowledge from the prover. In contrast to an honest verifier, $\mathcal{V}^*$ may have an **auxiliary input** $aux_{\mathcal{V}^*}$. This input can be interpreted as the a-priori knowledge of the verifier which it may have obtained in previous protocols where the prover used the same secret. Thus, we require that whatever can be efficiently computed from x and $aux_{\mathcal{V}^*}$ after completing the interaction with the prover on any x, can be computed from x and $aux_{\mathcal{V}^*}$ without interaction with the prover. To prove

[21] In particular, the proof should not enable the verifier — after completing the interaction with the prover — to convince any other party that the corresponding statement is true.

[22] More precisely, the view consists of the messages the verifier exchanges with the prover and the content of its random tape. Another way to define the view of the verifier is the sequence of verifier's local configurations during the interaction with the prover. However, the mentioned quantities suffice to determine this sequence.

the zero-knowledge property, one must show the existence of an algorithm called **simulator** $\mathcal{S}^{\mathcal{V}^*}$ which given the inputs of the verifier (i.e., common input and the auxiliary input $aux_{\mathcal{V}^*}$) can compute the view of the verifier. Note that cheating verifiers $\mathcal{V}^*$ might deviate from the protocol specification, and might produce a different view as the honest verifiers. Hence, we are required to give a simulator $\mathcal{S}^{\mathcal{V}^*}$ for every $\mathcal{V}^*$. In the following, we consider only black box reductions, i.e., there is a universal simulator which, given any $\mathcal{V}^*$ as a black box and $\mathcal{V}^*$'s inputs, simulates the view of $\mathcal{V}^*$ where $\mathcal{S}^{\mathcal{V}^*}$ is given the capability (privilege) to reset $\mathcal{V}^*$'s state, similar to *knowledge extractor* with prover (see Section 3.6.3).

The view of the verifier $view^{\mathcal{P}}_{\mathcal{V}^*}$ is a random variable defined by the run of the proof protocol with the (honest) prover $\mathcal{P}$. Note that here we consider the entire view instead of verifier's final output *ind*. However, this makes no real difference since for any $\mathcal{V}^*$ with $view_{\mathcal{V}^*}$ there exists a verifier $\mathcal{V}^{**}$ which stores the view and outputs only *ind*.

The view simulated by the simulator $\mathcal{S}^{\mathcal{V}^*}$ is denoted by $\mathcal{S}^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*})$.

The definition for auxiliary-input zero-knowledge proof is as follows:

Definition 3.8 (Auxiliary-input zero-knowledge, Goldreich 2001b) Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system, as given in Definitions 3.6 and 3.7. The proof system $(\mathcal{P}, \mathcal{V})$ is called **perfect zero-knowledge** if for all probabilistic interactive algorithms $\mathcal{V}^*$, there exists a (non-interactive) probabilistic algorithm, *simulator* $\mathcal{S}^{\mathcal{V}^*}$, such that for all valid parameters par_{sec}, for all $(x, aux) \in [\mathsf{Gen}^L(par_{sec})]$ and for all $aux_{\mathcal{V}^*} \in \{0,1\}^*$ the following conditions hold:

1. On common input x, $\mathcal{S}^{\mathcal{V}^*}$ outputs the symbol $\perp$ with probability at most 1/2, (i.e., $\mathbf{Prob}[\mathcal{S}^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*}) = \perp] \leq 1/2$),

2. The probability distributions $view^{\mathcal{P}}_{\mathcal{V}^*}$ and $\mathcal{S}'^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*})$ are identical, i.e.,
$$view^{\mathcal{P}}_{\mathcal{V}^*} = \mathcal{S}'^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*}),$$
where $\mathcal{S}'^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*})$ is a random variable describing the distribution of $\mathcal{S}^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*})$ conditioned on $\mathcal{S}^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*}) \neq \perp$.

The proof system $(\mathcal{P}, \mathcal{V})$ is called **statistically zero-knowledge** if the probability distributions $view^{\mathcal{P}}_{\mathcal{V}^*}$ and $\mathcal{S}'^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*})$ are statistically indistinguishable, i.e.,
$$view^{\mathcal{P}}_{\mathcal{V}^*} \overset{s}{\approx} \mathcal{S}'^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*}).$$

The proof system $(\mathcal{P}, \mathcal{V})$ is called **computationally zero-knowledge** if the above distributions are computationally indistinguishable, i.e.,
$$view^{\mathcal{P}}_{\mathcal{V}^*} \overset{c}{\approx} \mathcal{S}'^{\mathcal{V}^*}(x, par_{sec}, aux_{\mathcal{V}^*}).$$

$\diamond$

Composition of zero-knowledge protocols: A very important aspect regarding the zero-knowledge proof systems is the composition of zero-knowledge proofs. In particular, one expects that the **sequential composition** of zero-knowledge proofs is also zero-knowledge, i.e., if subsequent zero-knowledge protocols are performed, then the composed protocol must also be zero-knowledge (intuitively it means that the sum of zeros must also be zero!). Moreover, this should hold even for polynomially many proofs. Fortunately, it is proven that the definition of auxiliary-input zero-knowledge fulfill this requirement, and thus, the zero-knowledge proof is closed under the sequential composition as expressed in the following lemma:

Lemma 3.3 (Sequential Composition, Goldreich 2001b) *Let $\mathcal{P}$ be an interactive algorithm that is zero-knowledge with respect to auxiliary input on some language L. Suppose that the last message sent by $\mathcal{P}$, on input x, bears a special end-of-proof symbol. Let $Q(\cdot)$ be a polynomial, and let $\mathcal{P}_Q$ be an interactive algorithm that, on common input x, proceeds in $Q(|x|)$ phases, each of them consisting of running $\mathcal{P}$ on common input x (if $\mathcal{P}$ is probabilistic, the interactive algorithm $\mathcal{P}_Q$ uses independent coin tosses in each of $Q(|x|)$ phases.) Then $\mathcal{P}_Q$ is zero-knowledge on L with respect to auxiliary input. Furthermore, if $\mathcal{P}$ is perfect zero-knowledge with respect to auxiliary input, then so is $\mathcal{P}_Q$.* $\square$

This lemma is very fundamental and useful when designing zero-knowledge protocols. Informally, one usually constructs a protocol for proving a certain assertion. This proof is also called *atomic* proof. However, the atomic proof usually does not prove the claim completely, i.e., there is a certain success probability p for a cheating prover to convince the verifier. To handle this, the atomic proof is repeated until a certain degree of confidence is achieved (see also Remark 3.10) Now, the sequential composition lemma guarantees that if the atomic proof is zero-knowledge, so is also the proof which results from the repetitions of the atomic proof.

Unfortunately, this result cannot be proven for the parallel or concurrent composition (see Goldreich (2001b)).

Remark 3.12. When proving zero-knowledge property for a proof system, resulted from repeating an atomic proof, the overall simulator $\mathcal{S}^{\mathcal{V}^*}$ consists of the iterations of the simulator algorithm except that at the end of the iteration i, it stores the state $state^i_{\mathcal{V}^*}$, and resets $\mathcal{V}^*$ to this state if the next iteration $i+1$ fails.[23] $\circ$

[23]This is because $\mathcal{V}^*$'s behavior might not be the same in each iteration.

In later sections, we will be concerned with several zero-knowledge proof systems. Using the result of the composition theorem, it suffices to only prove the zero-knowledge property for the underlying atomic proof.

3.7 Basic Protocols

In this section, we consider several protocols which usually serve as building blocks of more complex protocols such as fingerprinting protocols. These protocols are used to prove certain assertions, mainly based on discrete logarithm quantities. The used techniques have become standard in the cryptographic literature, and thus, we mainly refer to the relevant literature. However, to give the basic idea, we will explain the most basic protocol for proving knowledge of discrete logarithm in more details.

3.7.1 Proving Knowledge of Discrete Logarithm

A zero-knowledge proof protocol for proving knowledge of a discrete logarithm from Chaum, Evertse, and van de Graaf (1988) is presented in Figure 3.3. In this protocol, the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that she knows the discrete logarithm x of a given public value h (the common input) to the base g where g, h are elements of a group G known to $\mathcal{V}$. More precisely, the parameters are defined as follows: We consider any group family $\mathcal{G}$ containing finite cyclic groups G of known order $|G|$. Concrete examples are $G := \mathbb{Z}_p^*$ where $p \in_{\mathcal{R}} \mathbb{P}$ or $G := \mathbb{Z}_{p/q}^*$ the (unique) multiplicative subgroup of $\mathbb{Z}_p^*$ of prime order q.

Let k be the security parameter and $SG_{\mathcal{G}}$ a group sampler. The knowledge relation, describing the objective to be proven, is defined as follows:

$$R := \{\big(x, (G, g, h)\big) | G \in [SG_{\mathcal{G}}(1^k)] \wedge g, h \in G \wedge x \in \mathbb{Z}_{|G|} \wedge g^x = h\}.$$

We require another security parameter ϵ_S representing the soundness error we are willing to tolerate. Using this, we can determine the number of iterations γ_S of the atomic protocol in Figure 3.3 (see also Remark 3.10).

We denote the proof protocol with

$$(\mathcal{P} : -;\ \mathcal{V} : ind) \leftarrow \mathsf{ProveDL}(\mathcal{P} : \log_g(h);\ \mathcal{V} : -;\ * : G, g, h)$$

where $ind \in \{accept, reject\}$.

The idea behind the protocol in Figure 3.3 is as follows: On common input (G, g, h), the prover $\mathcal{P}$ sends to the verifier $\mathcal{V}$ a random version a of h. On binary challenge c, sent by $\mathcal{V}$, $\mathcal{P}$ sends a response r which is a linear equation depending on the random value w, the challenge c, and the secret x. $\mathcal{V}$ accepts if and only if $g^r = ay^c$ holds.

It can be shown that this protocol is a proof of knowledge with perfect zero-knowledge property (Schnorr 1991). We informally explain the reasons.

- *Proof of knowledge*: If for a value a, $\mathcal{P}$ answers correctly to both challenges, then $\mathcal{P}$ must know both values w and $r = w + x$. This means, in each iteration, a cheating prover, who does not know x, can in fact correctly respond to at most one of the two possible challenges. Thus, in each iteration $\mathcal{V}$ accepts with probability 1/2. For γ_S iterations the tolerated soundness error will be $1/2^{\gamma_S}$.[24]

  To extract the discrete logarithm, the extractor $\mathcal{E}$ sends a challenge $c = 0$ to $\mathcal{P}^*$, and lets $\mathcal{P}^*$ run until it receives the response r_0. Then $\mathcal{E}$ resets $\mathcal{P}^*$, sends the challenge $c = 1$ to $\mathcal{P}^*$, and lets $\mathcal{P}^*$ run until it receives the corresponding response r_1. If both r_1 and r_2 pass the verification, then $\mathcal{E}$ outputs discrete logarithm $x := r_1 - r_0$.

- *Perfect zero-knowledge*: In the protocol, $\mathcal{V}$ can extract information about the secret x only from the response r. However, for $c = 0$, r is a random value w, and for $c = 1$, r perfectly hides x since it is the result of an encryption of x with a one-time pad w. Thus, in both cases no information about x is revealed to $\mathcal{V}$. However, this is only the intuition behind the zero-knowledge property. To show it formally, one should construct a black-box *simulator* $\mathcal{S}^{\mathcal{V}^*}$, and show that it generates the view of the (cheating) verifier $\mathcal{V}^*$ with the same probability distribution. We explain the simulator program briefly: The simulator computes the protocol quantities backwards, i.e., it first chooses randomly and uniformly the challenge $c \in_{\mathcal{R}} \{0, 1\}$, and the value $r \in_{\mathcal{R}} \mathbb{Z}_{|G|}$, and computes the value a as $a := g^r/h^c$. Now, the simulator starts $\mathcal{V}^*$ with the given common input $x = (G, g, h)$ (and $aux_{\mathcal{V}^*}$), and sends a to $\mathcal{V}^*$ who responses with a challenge c^*. If $c^* = c$ then $\mathcal{S}^{\mathcal{V}^*}$ outputs the view $view_{\mathcal{S}^{\mathcal{V}^*}} := (x, a, c, r)$. Else ($c^* \neq c$), it repeats the above procedure. One can now show that the probability distribution of this view is exactly the same as in the real protocol, and that the probability that the simulator fails is at most 1/2.

Remark 3.13. For efficiency reasons, it is desired to have fewer iterations. A simple way to achieve better efficiency is to choose challenges with more bits, also called **deep challenges**. For the protocol in Figure 3.3 this means that we only replace $c \in \mathbb{Z}_2$ with $c \in \mathbb{Z}_{|G|}$. Now, one can show that this construction fulfills the completeness and soundness requirements but it is *not* believed that the protocol is still zero-knowledge (see also Schnorr (1991)). ○

[24]Note that the iterations are independent, i.e., each time a new w and a new c should be chosen. The verifier accepts only if all γ_S verifications in the repeated protocol end with true.

Figure 3.3 Zero-knowledge proof of knowledge of discrete logarithm: ProveDL()

$\mathcal{V}$		$\mathcal{P}$
(G, g, h)		(G, g, h, x)
		$w \in_{\mathcal{R}} \mathbb{Z}_{\lvert G \rvert}$
		$a := g^w$
	$\xleftarrow{\quad a \quad}$	
$c \in_{\mathcal{R}} \mathbb{Z}_2$		
	$\xrightarrow{\quad c \quad}$	
		$r := w + cx \bmod \lvert G \rvert$
	$\xleftarrow{\quad r \quad}$	
$g^r \overset{?}{=} ah^c$		

3.7.2 Proving Knowledge of Representation

The protocol for proving knowledge of representation is the extension of the protocol for proving knowledge of discrete logarithm to multiple bases (see, e.g., Chaum, Evertse, and van de Graaf (1988)). Given the generators $g_1, g_2, \cdots, g_n \in G$ and an element $h \in G$, prove the knowledge of the integers $x_1, x_2, \cdots, x_n$ such that $h := \prod_{i=1}^{n} g_i^{x_i}$. The vector $I = (x_1, \cdots, x_n) \in \mathbb{Z}_{\lvert G \rvert}^n$ is called a **representation** of $h \in G$ to the (generator) base $B = (g_1, ..., g_n)$. We denote this by

$$repr(h, B, I) :\iff h = g_1^{x_1} ... g_n^{x_n}.$$

The protocol for this proof is shown in Figure 3.4.

Figure 3.4 Zero-knowledge proof of knowledge of a representation $\mathsf{ProveRep}()$

$$
\begin{array}{ccc}
\mathcal{V} & & \mathcal{P} \\
\big((G, g_1, \cdots, g_n), h\big) & & \big((G, g_1, \cdots, g_n), h, (x_1, \cdots, x_n)\big) \\
\hline
 & & w_1, \cdots, w_n \in_{\mathcal{R}} \mathbb{Z}_{|G|} \\
 & & a := \prod_{i=1}^{n} g_i^{w_i} \\
 & \xleftarrow{\quad a \quad} & \\
c \in_{\mathcal{R}} \mathbb{Z}_2 & & \\
 & \xrightarrow{\quad c \quad} & \\
 & & r_i := w_i + cx_i \bmod |G| \\
 & \xleftarrow{(r_1, \cdots, r_n)} & \\
\prod_{i=1}^{n} g_i^{r_i} \overset{?}{=} ah^c & & \\
\hline
\end{array}
$$

We denote this protocol with

$$
\begin{aligned}
&(\mathcal{P} : -;\ \mathcal{V} : \mathit{ind}) \\
&\quad \leftarrow \mathsf{ProveRep}(\mathcal{P} : \mathit{repr}(h, B, I);\ \mathcal{V} : -;\ * : (G, B), h)
\end{aligned}
$$

where $\mathit{ind} \in \{\mathit{accept}, \mathit{reject}\}$.

This protocol is a zero-knowledge proof of knowledge of a representation of h (Chaum, Evertse, and van de Graaf 1988).

3.7.3 Proving Knowledge of Simultaneous Discrete Logarithm

The protocol shown in Figure 3.5 represents a proof that the discrete logarithms of two values with respect to different bases are equal. Given the generators $g_1, g_2 \in G$ and the elements $h_1, h_2 \in G$, prove knowledge of $x \in \mathbb{Z}_{|G|}$ where $h_1 \equiv g_1^x$ and $h_2 \equiv g_2^x$. We denote this protocol with

$$
\begin{aligned}
&(\mathcal{P} : -;\ \mathcal{V} : \mathit{ind}) \\
&\quad \leftarrow \mathsf{ProveEqDL}(\mathcal{P} : \log_{g_1}(h_1), \log_{g_2}(h_2);\ \mathcal{V} : -;\ * : (G, g_1, g_2), h_1, h_2)
\end{aligned}
$$

where $\mathit{ind} \in \{\mathit{accept}, \mathit{reject}\}$. The basic techniques are from Chaum, Evertse, and van de Graaf (1988). Similar protocols are also the basis for the techniques in Camenisch and Stadler (1997) for proving polynomial relations among secret exponents.

Figure 3.5 Proving knowledge of simultaneous discrete logarithm $\mathsf{ProveEqDL}()$

$$\begin{array}{ccc} \mathcal{V} & & \mathcal{P} \\ \big((G,g_1,g_2),h_1,h_2\big) & & \big((G,g_1,g_2),h_1,h_2,x\big) \\ \hline & & w' \in_R \mathbb{Z}^*_{|G|} \\ & \xleftarrow{\quad a'_1, a'_2 \quad} & a'_1 \leftarrow g_1^{w'}, a'_2 \leftarrow g_2^{w'} \\ c \in_R \mathbb{Z}_2 & \xrightarrow{\quad c \quad} & \\ g_1^r \overset{?}{=} h_1^c a'_1, g_2^r \overset{?}{=} h_2^c a'_2 & \xleftarrow{\quad r \quad} & r \leftarrow cx + w' \end{array}$$

As usual, the proofs of knowledge can either be made with small challenges, and be provably zero-knowledge, or with larger challenges for greater efficiency.

Chapter 4

Fingerprinting

This chapter focuses on fingerprinting schemes, their models, constructions and development. It presents a novel, concrete and reasonably efficient construction for anonymous fingerprinting schemes based on ideas from digital coin systems. The construction is asymmetric and collusion-tolerant, and offers direct non-repudiation, i.e., the accused buyer need not to participate in any trial protocol to deny charges. Thus, in case of illegal redistribution of digital works the scheme provides the merchant with enough evidence to convince an arbiter. This is advantageous since trials with direct non-repudiation are more useful in real life. The difference is similar to that between the "normal" and "undeniable" signatures.

4.1 Prerequisites

4.1.1 Subjects/Roles

The objects that copyrights, or rights in general, refer to are **works** representing different types of digital content (e.g., multimedia, software) denoted by *work*. The set *WORK* denotes the set of all works. A work is *created* when it is fixed in a well-defined form for the first time, e.g., in form of images, video-clips or music (see also US Copyright Office (2002)). The main roles are as follows:

- **Authors/Creators** are parties who originate (create) works. The author of a work is given all rights (defined in the legal framework) on the corresponding work, or more precisely on the corresponding work-class.

- **Owners** are parties who *legally* obtain rights on a work (and its corresponding work-class) by creating that work themselves, or by means of transactions such as transferring or licensing of rights from others. They may themselves be authorized to transfer or license rights to others. Examples are distributors, merchants, license holders. We call authors and owners **authorized parties**. For fingerprinting schemes, we will take **merchants**, denoted by $\mathcal{M}$, as representative for all authorized parties.

  Those parties who obtain/purchase rights via transactions are called **buyers** and denoted by $\mathcal{B}$.

- **Registration centers** are parties who register users or works, and are denoted by $\mathcal{RC}$.

- **Traitors** are those dishonest parties who purchase/obtain works *legally*, and violate the copyrights by copying and redistributing digital works *illegally* (i.e., without authorization). A coalition of dishonest parties is called a **collusion** and denoted by COL.

- **Arbiter** (judges) are honest parties required to be involved in certain situations such as trial. They are denoted by $\mathcal{J}$.

Each party X should have a digital identity, usually represented through a public-key pk_X. With $\mathcal{ID}_X$ we denote the set of the identities of all involved parties X, e.g., the set of identities of buyers is $\mathcal{ID}_{\mathcal{B}}$.

Each of the mentioned parties is represented by a probabilistic polynomial-time algorithm.

4.1.2 Works and Similarity

As mentioned in Chapter 1.1, rights do not only refer to a single work only, but rather to a set of closely related **similar works**. Consider for example an artist who created a digital image $work$. Then, naturally and legally, a rotated or compressed version $work'$ of $work$ is also considered to be a creation of this artist. This is because rotating or compressing a digital image is no creative achievement, and does not lead to new original works of authorship.

In the context of rights, consider the "right to use an image on a web-page". Generally, this right does not only allow a specific image $work$ to be put on the web-page of the corresponding rights owner. However, the owner of this right may also put on her web-page a compressed, thumbnail version, or more generally any *similar* version $work'$ of this image.

In the following, we assume a **similarity relation** $\rightarrow_{\mathsf{sim}}$ on works to be given. Here, $work \rightarrow_{\mathsf{sim}} work'$ denotes the fact that $work$ is similar to $work'$. Works similar to a work $work$ are called the **similarity-**

set/work-class of *work* and are denoted by the set $WORK^{sim}_{work} := \{work' \in WORK \mid work \rightarrow_{\mathsf{sim}} work'\}$.

To determine whether a work $work'$ is similar to a work $work$, i.e., whether it lies in the work-class $WORK^{sim}_{work}$, we assume the existence of an *automatic* **similarity-test** which we denote with $ind \leftarrow \mathsf{simtest}(work, work')$ where $ind \in \{true, false\}$. There are different reasonable ways for testing similarity between works, and we note that conformance with copyright-laws strongly depends on a suitable definition of similarity. Possible instantiations are (Adelsbach, Pfitzmann, and Sadeghi 2000):

- **Robust hashing:** Similar digital works have similar main features. In the context of database retrieval of multimedia data or authentication of digital content, these main features are used to compute a characteristic value of digital objects (see, e.g., Lin and Chang (1998) and Wang et al. (1998)). Using an appropriate metric, one can compare the characteristic values of works with respect to this metric instead of the works themselves.

- **Robust digital watermarks:** Robust digital watermarking schemes embed additional information in digital objects, such that this information can later be detected or retrieved again, even if the digital object has been modified (see, e.g., Hartung and Kutter (1999), or Katzenbeisser and Petitcolas (2000)). We will consider watermarking schemes in section 4.3. We define works *similar* if they contain the *same* watermark.

In the context of *fingerprinting*, any redistributed work *similar* to the original work can be traced depending on the purchased rights: If a merchant finds a redistributed work $work^{red}$ similar to one of the works $work$ he has already sold, i.e., $true \leftarrow \mathsf{simtest}(work, work^{red})$, and if he considers this work for illegally redistributed then he starts the identification procedure to trace the source of redistribution.

4.2 Classification and Research History

We classify the fingerprinting schemes for digital works according to the following properties:

- **Collusion-Tolerance:** As mentioned in Section 1.1.2.2, a digital work is fingerprinted by embedding the identifying information (fingerprint) as a *watermark* into this work. Each copy of the original work gets its own and unique fingerprint. The main security requirement on the underlying *watermarking schemes* is robustness against attacks attempting to remove the embedded watermark. This is fine as long as one watermarked copy is available to the adversary. However, a traitor (or a collusion) possessing several fingerprinted copies of a work can easily compare these copies, detect the differences, and may be able to generate a copy which does not contain any of the given fingerprints. In this manner, the traitor(s) can mask his (their) identities. We call such an attack a **collusion attack**. Thus, it should be clear that naive embedding of random fingerprints or serial numbers into works, as still done in practice, cannot resist collusion attacks.[1]

Therefore, the first enhancement to conventional fingerprinting schemes is the addition of collusion-tolerance Blakley, Meadows, and Purdy (1986), Boneh and Shaw (1995) and Cox et al. (1997). Hereby, the watermark should tolerate a collusion attack with a given maximum number of different copies.

- **Asymmetry:** The first proposed fingerprinting schemes were **symmetric**. In such schemes, the merchant is the one who fingerprints the corresponding works, and both the buyer and the merchant know the fingerprinted work. Thus, in the case of redistribution, it is not clear who has indeed redistributed the work, a dishonest buyer or the merchant himself. To solve this problem **asymmetric fingerprinting** was proposed (Pfitzmann and Schunter (1996), Pfitzmann and Waidner (1997b), Biehl and Meyer (1997)). The idea is as follows: After identifying a traitor, a proof of treachery should be found in the redistributed copy. This means that the merchant is able to find some data in the redistributed work which only the identified traitor could have computed. In contrast to symmetric schemes, asymmetric schemes require the underlying work to be fingerprinted via an interactive protocol between the buyer and the merchant where the buyer also inputs own secrets. At the end of this protocol only the buyer knows the fingerprinted work. The main construction proposed by Pfitzmann and Schunter (1996) was based on general primitives; an *explicit* construction, i.e., without such primitives, was only given for the case without significant collusions. Explicit collusion-tolerant constructions were given by Pfitzmann and Waidner (1997b) and Biehl and Meyer (1997).[2]

[1]An example of a collusion attack is the **statistical averaging attack** (Hartung and Kutter 1999). Depending on the watermarking scheme, this attack may require only a few copies to generate a copy containing non of the fingerprints by computing the average of the available copies.

[2]In the context of fingerprinting, we may distinguish between fingerprinting the actual work or fingerprinting a key for decrypting it. This special variant of fingerprinting is called **traitor tracing** (Chor, Fiat, and Naor (1994) and Naor and Pinkas (1998)): Here, the keys for broadcast encryption (Fiat and Naor 1994) are fingerprinted. Other examples for traitor tracing schemes are Boneh and Franklin (1999) and Fiat and Tassa (1999).

Asymmetric traitor tracing was introduced by Pfitzmann (1996b) with a construction based on general primitives. Explicit constructions for this case were also given by

- **Anonymity**. As in the "real-life" market places, it is desirable that also electronic market places offer privacy to customers. Buying works (pictures, books, etc.) reveals behavioristic information about individuals. Consumer may wish to use different services while maintaining their privacy. Thus, it should also be possible to purchase fingerprinted works anonymously. However, the identification of traitors should still be possible. Anonymous (asymmetric) fingerprinting was introduced by Pfitzmann and Waidner (1997a). Their construction uses general theorems like "every NP-language has a zero-knowledge proof system" (Goldreich 2001a), and is only partly explicit.

  In this thesis, we propose an explicit construction based on ideas from **digital coin** systems. It is fairly efficient in the sense that all operations are efficient computations with modular multiplications and exponentiations.

  Note that we understand the notion of anonymity in the **strong** sense of the original definition by Pfitzmann and Waidner (1997a), i.e., any coalition of merchants, central parties and other buyers should not be able to distinguish purchases of the remaining buyers. A fingerprinting scheme with **weak anonymity** can easily be constructed by using any asymmetric fingerprinting scheme under a certified pseudonym instead of a real identity. In this case, the anonymity is conditioned on the trust in honesty of some central party. Other constructions for anonymous (asymmetric) fingerprinting with weak anonymity are proposed by Domingo-Ferrer and Herrera-Joancomarti (1998) and Domingo-Ferrer (1999). The construction of the former is based on general two-party computation, and that of the latter is based on oblivious transfer protocols. However, Sadeghi (2001) shows that both schemes Domingo-Ferrer and Herrera-Joancomarti (1998) and Domingo-Ferrer (1999) are not secure.

- **Direct non-repudiation:** The asymmetric property enables the merchant to obtain a proof of treachery in case of illegal redistribution, and to convince an arbiter in a trial. This may also require the accused buyer to participate in the trial to deny the charges, if possible. However, one is interested in a construction which enables the merchant to obtain enough evidence to convince any arbiter that a certain buyer is a traitor without involving the buyer. We call this property *direct non-repudiation*. Obviously, systems offering this property are more useful in real life since users do not have the burden of storing evidences. Note that users could rightly or wrongly claim to have lost

Pfitzmann and Waidner (1997b). More efficient construction were given by Kurosawa and Desmedt (1998); however, they are not asymmetric in the usual sense but *arbitrated*, i.e., a certain number of predefined arbiters can be convinced by the merchant (similar to the difference between arbitrated authentication codes and asymmetric signature schemes).

the information needed for the trial, or the password to it, or it could occur that a dissolved company did not leave such information to its legal successors. With direct non-repudiation, the provable identification works even if the accused user is not reachable. Thus, the trial is only a 2-party protocol. The difference is similar to that between normal digital signatures (direct non-repudiation) and undeniable signatures (Chaum and van Antwerpen 1990) where the signer is needed in trial. This thesis presents an anonymous fingerprinting scheme with this property. The construction applies methods from **coin tracing**,[3] concretely from Frankel, Tsiounis, and Yung (1996). In particular, we apply a technique which we call **delayed verifiable encryption**. Using this technique, certain information is encrypted in one phase of the fingerprinting scheme, and securely proven to be correct in a later phase (delayed verification).

However, the similarity of the fingerprinting scheme to the coin tracing methods is only at the technical level: One does not require a trusted third party (trustee) as for coin tracing, otherwise one could use the simple solution with *weak anonymity*, mentioned before. Further, there is a need for closer binding between this encryption and the coin than in coin tracing to provide an unforgeable link to the license conditions.

Another approach by Camenisch (2000) for constructing an anonymous fingerprinting scheme with direct non-repudiation is based on group signatures. Regarding one aspect, this construction is more efficient than the coin-based solution: It needs only one registration for all purchased works whereas the coin-based construction requires the buyer to open a new account for each purchase. However, the coin-based construction allows **anonymous legal redistribution** whereas the proposal by Camenisch (2000) does not. Legal distribution means that a buyer can purchase "right of distribution" allowing her to copy and redistribute the underlying work after a certain time (e.g., five years after the purchase time). However, after the redistribution, an honest buyer should not be wrongly accused by any collusion against this buyer.

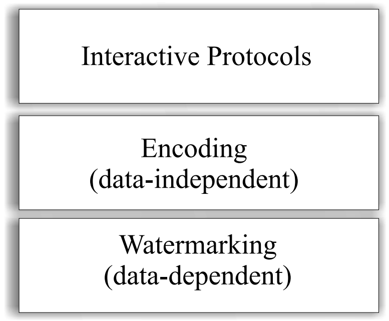
4.2.1 A Layer Model for Fingerprinting

We may represent the whole fingerprinting process by a layer model as shown in Figure 4.1. At the bottom we have the data-dependent layer representing the data to be fingerprinted, and the appropriate watermarking schemes.

[3]This is a mechanism which allows a bank, in cooperation with a trustee, to revoke the anonymity of a user, and trace the corresponding coins. Such measures are proposed to deter the misuse of anonymous payment systems for criminal purposes.

In the next layer, we place the data-independent part of the fingerprinting scheme. Here, the information to be embedded may be encoded first to have certain properties, e.g., collusion-tolerance. However, depending on the watermarking scheme, we may combine these layers in one. Symmetric fingerprinting schemes include these layers. At the top layer we place asymmetric fingerprinting which usually uses a symmetric fingerprinting scheme as a building block. At this layer, interactive proof systems are applied to securely and verifiably embed identifying information into works such that on the one hand, only the buyer knows the fingerprinted work, and on the other hand, the merchant obtains enough information to provably identify a traitor in case of illegal redistribution.

Figure 4.1 A layer model for fingerprinting



4.3 Watermarking

4.3.1 Introduction

Informally, **watermarking** is a technology to alter a work by embedding information into that work. Our concern in this thesis are those watermarking schemes which embed the corresponding information **imperceptibly**. For surveys on different kinds of watermarking schemes see Katzenbeisser and Petitcolas (2000).

The watermarking schemes we are interested in embed encoded information into works such that this information can be efficiently read only by authorized parties, and cannot be removed by unauthorized parties, at least not without rendering the work useless.

4.3.2 Model

Let Σ be a finite alphabet and $l \in \mathbb{N}$. A watermark is, in general, represented by a vector $\vec{wm} \in \Sigma^l$. A common choice is $\Sigma = \{0, 1\}$, and in this case, we denote the watermark by a binary string wm. Each symbol in the watermark represents the value of a **mark**. With **watermark signal** we mean a signal which is *combined* (e.g., added) with the values of the original signal (original work). A mark determines how the values of the **watermark signal** should be combined with the original work.

To make this more clear, consider the following example: A photographer may embed a watermark $wm \in \{0,1\}^l$ into a digital image as follows: The watermark consists of l (binary) marks. A mark with binary "1" means increase the intensity of certain pixels while a binary "0" means decreasing it. The increased/reduced intensity amount represents the value of the watermarking signal.

Having a work watermarked, one is interested in *reading* this watermark later. Reading a watermark means **detection** or **extraction** of that watermark by authorized parties. Watermark readers requiring the original work as input are called **non-blind**, otherwise they are called **blind**. Throughout this thesis we are concerned with *non-blind* readers. A concrete and well-known scheme which can be implemented for both blind and non-blind watermarking is proposed by Cox et al. (1996), and is described in Section 4.3.5.1.

There are a variety of models proposed for watermarking schemes (see Cox, Miller, and Bloom (2002)). The following definitions for the components of watermarking schemes are advantageous when we use watermarking schemes as a building block within more complex systems. Similar definitions are given by Craver, Yeo, and Yeung (1998) and Petitcolas, Anderson, and Kuhn (1999). More explicit definitions closer to the signal theoretical model are given by Cox, Miller, and McKellips (1999) and Pitas and Voyatzis (1999).

The main components of a watermarking scheme are the following polynomial-time algorithms:

- **Key generation:** On input of a (security) parameter k, the key generation algorithm $\mathsf{GenKeyWM}(k)$ generates watermarking embedding key sk_{emb}^{WM} and extracting key sk_{ext}^{WM} (or detecting sk_{det}^{WM} key).

- **Embedder:** On input of a work $work$, the watermark $\vec{wm}$, and the embedding key sk_{emb}^{WM}, the algorithm $\mathsf{EmbedMark}()$ outputs the watermarked work $work^{WM}$, also called **stego work**.

- **Extractor:** On input of a watermarked work $work'$, the secret extracting key sk_{ext}^{WM}, and the original work $work$, an extracting algorithm $\mathsf{ExtractMark}()$ extracts the embedded watermark $\vec{wm}$ from $work'$.

- **Detector:** On input of a watermarked work $work'$, the secret detecting key sk_{det}^{WM}, and the original work $work$, a detecting algorithm $\mathsf{DetectMark}()$ outputs *true* or *false* indicating whether the watermark $\vec{wm}$ is detected in $work'$ or not.

We distinguish between the following watermarking schemes:

- **Extracting Watermarking Scheme:** An extracting watermarking scheme is a tuple $(\mathsf{GenKeyWM}(), \mathsf{EmbedMark}(), \mathsf{ExtractMark}())$ such that for all $work \in WORK$, all $\vec{wm} \in \Sigma^l$, all key pairs $(sk_{emb}^{\mathsf{WM}}, sk_{ext}^{\mathsf{WM}}) \leftarrow [\mathsf{GenKeyWM}(k)]$ and $work' \leftarrow \mathsf{EmbedMark}(work, \vec{wm}, sk_{emb}^{\mathsf{WM}})$ the following holds
$$\vec{wm} \leftarrow \mathsf{ExtractMark}(work', work, sk_{ext}^{\mathsf{WM}}).$$
- **Detecting Watermarking Scheme**: A detecting watermarking scheme is a tuple $(\mathsf{GenKeyWM}()$, $\mathsf{EmbedMark}()$, $\mathsf{DetectMark}())$ such that for all $work \in WORK$, all $\vec{wm} \in \Sigma^l$, all key pairs $(sk_{emb}^{\mathsf{WM}}, sk_{det}^{\mathsf{WM}}) \leftarrow [\mathsf{GenKeyWM}()]$ and $work' \leftarrow \mathsf{EmbedMark}(work, \vec{wm}, sk_{emb}^{\mathsf{WM}})$ the following holds
$$true \leftarrow \mathsf{DetectMark}(work', work, \vec{wm}, sk_{det}^{\mathsf{WM}}).$$

In the definitions above, we still have to consider the probability for **false positive** in the extraction/detection process. False positive means to extract/detect a watermark from/in a work which does not actually contain this watermark. We require the probability of this event to be negligible.

Remark 4.1. In this thesis, we are concerned with *symmetric watermarking* schemes, i.e., we set $sk^{\mathsf{WM}} = sk_{emb}^{\mathsf{WM}} = sk_{ext}^{\mathsf{WM}}$ and $sk^{\mathsf{WM}} = sk_{emb}^{\mathsf{WM}} = sk_{det}^{\mathsf{WM}}$.[4]

Remark 4.2. We use watermarking schemes with extracting readers because we can apply them more efficiently within the fingerprinting schemes. ∘

4.3.3 Requirements

Watermarking schemes can be classified with respect to several criteria such as the type of underlying work (image, video, audio, text, etc), the degree of imperceptibility, robustness, security and so on. For our purposes, we require watermarking schemes which have the following main properties:

- **Fidelity**: This parameter is a measure for the **perceptual indistinguishability** between the original work and the watermarked work. The modifications caused by embedding a watermark should remain under an appropriate perceptible threshold. This implies that there should be some criteria or metrics according to which one can design the watermark signal values, and control the amount of distortion when watermarking a work.

  Note that perceptibility criteria depend strongly on the data type. For audio and video data the metric is defined according to human perceptual model, which is not very precise. For programs, however, perceptibility means the **semantic equivalence** as understood in computer science.

[4] The existing **asymmetric watermarking** schemes still suffer under lack of security (see Furon and Duhamel (2000) and Eggers, Su, and Girod (2000)).

- **Robustness**: The watermarking scheme should resist (passive and active) attacks attempting to remove the embedded watermark unless the attacks result in a work $work^*$ which is *not* similar to the original work $work$ anymore, i.e., $false \leftarrow \mathsf{simtest}(work, work^*)$ where we understand similarity as defined in Section 4.1.2. By *removing* a watermark we mean that this watermark is not extractable/detectable after the attack.

  For passive attacks against a watermarking scheme, we consider the following aspect:

  - **Secrecy**: The stego work (watermarked work) should not leak *any additional* information on the secret inputs of the embedder, i.e., the embedding key sk^{WM} and the original work $work$.

  Depending on the application and the type of data, various kinds of active attacks on watermarks are possible. In particular, for multimedia works, there are plenty of possibilities (and tools) to attack the watermark attempting to remove it (see e.g., Petitcolas, Anderson, and Kuhn (1998), Petitcolas and Anderson (1998) and Anderson (2001)). For multimedia works, we require the watermarking scheme to also resist typical attacks:

  - **Common signal processing**: Lossy compression, noisy transmission, digital-analog and analog-digital conversion, resampling, printing and scanning, adding noise signals, etc.
  - **Geometric distortion**: Scaling, translation, rotation, etc.

Remark 4.3. It is reasonable to consider robustness for watermarks as long as the modified works $work^*$ lie in the work-class of the original work (i.e., $true \leftarrow \mathsf{simtest}(work, work^*)$). This restriction is natural since one is not interested in works $work^*$ which do not have any perceptual similarity to the original work. For instance, one can always remove the watermark by strongly distorting the corresponding work. However, in this case, we expect the quality of the distorted work to be so poor that no one would use it.

Remark 4.4. Some literature define resistance against common signal processing as *robustness*, and the ability to resist unauthorized removal as *security*. ○

4.3.4 Main Steps of Watermarking

Generally, the watermarking process consists of the following steps:

- *Appropriate representation*: Watermarking schemes operate on certain representations of works in an appropriate **embedding domain**. This representation may be in the **original domain** (e.g., pixels for images), or in a **transform domain** (e.g., coefficients of a Fourier transformed image). By using transformations, one attempts to exploit the transform domain to achieve certain properties for watermarks, e.g., robustness against data compression. We call the values that represent a work in the embedding domain (e.g., the Fourier coefficients) *components*.

There is a large body of literature on watermarking schemes. The majority of the proposals are on watermarking images and videos (but the proposed techniques can be customized for audio data as well.) In the original domain, pixels are subject to modifications, e.g., the least significant bits of pixels are modified (see, e.g., Tirkel et al. (1993) and Kutter (1998)). In the transform domain, one applies various types of transformations such as **Discrete Cosine Transform**, **Discrete Fourier Transform**, **Wavelet Transform**, **Mellin-Fourier Transform** etc. Some proposals for watermarking schemes based on these transformations are Ruanaidh, Boland, and Sinnen (1995), Perrig, Herrigel, and Ruanaidh (1997), Cox et al. (1997), Herrigel et al. (1998), and Kundur and Hatzinakos (1998).

One advantage of applying transformations is to make the components of the data independent, since pixels are usually highly correlated.

The type of the embedding domain has great impact on the properties of watermarks. For instance, schemes which use discrete cosine or wavelet transformation are more robust against compression algorithms such as JPEG, whereas those schemes using Fourier-Mellin transformation of an image are more robust against geometric transformations such as scaling and rotation.

- *Choosing components*: After determining a representation of the work in an embedding domain, a subset of the components are chosen (e.g., pseudo randomly) and modified. These components carry the marks, and are called **marking positions**. The choice of these components is made with the goal to achieve certain properties such as maintaining fidelity, and at the same time improving the robustness as much as possible.

- *Watermarks and watermark signal*: Watermarks are usually represented as bit-strings. Each mark is embedded into the work at a marking positions by modifying the original components in the corresponding embedding domain. The modification is done by combining the components of the watermark signal with the components of the original work. The values for the watermark signal may be chosen randomly, and merely associated with the embedding information in some way. There are different ways of combining watermark signal values with the values of original components: One way is to add the watermark signal component to the corresponding original component.[5] Another way is to replace the original component by the corresponding watermark signal component.

The modified work is obtained through replacing the original components by their corresponding modified components.

To deal with the imperceptibility issue, each watermark signal value may be modified by an intensity parameter. The values this parameter assumes may be determined at key generation, or they may depend on the individual components being marked, or on the data type of the underlying work. One way to incorporate the intensity parameter is to multiply this parameter with the watermark signal value. Advanced watermarking schemes use **perceptual masking models** to determine the intensity parameters (see, e.g., Swanson, Zhu, and Tewfik (1998), Cox and Miller (1997), and Wolfgang, Podilchuk, and Delp (1999)).

- *Reading a watermark*: Given a watermarked, and eventually modified, work $work^*$, transform it to the representation in the embedding domain. Then extract the components that correspond to the marking positions. If we apply an extracting watermarking scheme, we can directly interpret the marks. Note that the extracting algorithm should output the embedded watermark with high probability. One way to achieve this is to apply error-correcting codes during the selection and reading process.

  In the case of detecting schemes, one usually applies correlation-based tests (Cox, Miller, and Bloom (2002)). Hereby, one retrieves a hypothetical watermark $\vec{wm}^*$ from $work^*$, and then compute its correlation with the original watermark $\vec{wm}$. The detection succeeds if the correlation is above a certain predefined threshold.

4.3.5 A Generic Watermarking Scheme

In Section 4.3.4, we briefly reviewed the main steps of a non-blind watermarking process. In this section, we consider a generic watermarking scheme which builds the basis of many known watermarking schemes. Let *work* be a work to be watermarked.

1. Transform *work* to the appropriate embedding domain. We denote this representation with

$$\vec{work} := (work_1, work_2, \cdots, work_m, work_\delta)$$

[5]Other operations used to combine these components can be reduced to addition by an appropriate transformation. For instance, the multiplication of these components can be converted to addition by a logarithmic transformation.

where $(work_1, work_2, \ldots, work_m)$ denote those components which can be modified satisfying the desired data fidelity, and $work_\delta$ denotes the rest of the components.

2. Select the marking positions, i.e., the components to be marked. One may make this choice by randomly choosing the indices, e.g., with respect to the secret key $sk^{\mathsf{WM}} := (i_1, \cdots, i_l)$ which is generated by the key generation algorithm. After selecting the components, we write

$$\vec{work} := (work_{i_1}, work_{i_2}, \cdots, work_{i_l}, work'_\delta)$$

where $work'_\delta$ now contains $work_\delta$ and all other components which are not selected as marking positions.

3. Let $\vec{wm} := (wm_1, wm_2, \cdots, wm_l)$ be the representation of the watermark signal, and $\vec{S} := (s_1, s_2, \cdots, s_l)$ be the desired watermark intensity. The representation of the watermarked work $\vec{work}'$ is obtained by replacing each $work_{i_j}$, $j \in [1, l]$ with $work'_{i_j} := work_{i_j} + s_j wm_j$ for $j \in [1, l]$. We denote the representation of the watermarked work with

$$\vec{work}' := (work'_{i_1}, work'_{i_2}, \cdots, work'_{i_l}, work'_\delta).$$

4. Re-transform the watermarked components to obtain $work^{WM}$.

5. The next phase is reading a watermark: Assume, we are given a work $work^*$ (received signal) with $true \leftarrow \mathsf{simtest}(work, work^*)$. First, $work^*$ is transformed to the domain where embedding took place. We denote this representation with

$$\vec{work}^* := (work^*_1, work^*_2, \ldots, work^*_m, work^*_\delta).$$

Then those components are extracted which correspond to the marked ones by using the key sk^{WM}. We denote this by

$$\vec{work}^* := (work^*_{i_1}, work^*_{i_2}, \ldots, work^*_{i_l}, work^{**}_\delta).$$

If we have access to the original components, $\vec{work}_{i_j}$, we can extract a hypothetical watermark $\vec{wm}^* := (wm^*_{i_1}, wm^*_{i_2}, \cdots, wm^*_{i_l})$ where $wm^*_{i_j} := (work^*_{i_j} - work_{i_j})/s_j$ for $j \in [1, l]$. Now, depending on the type of reader (extracting or detecting), one may apply error-correcting codes to retrieve the original watermark $\vec{wm}$, or use correlation methods to match $\vec{wm}^*$ against the original watermark signal.

Remark 4.5. It is common that the watermark is a string $wm \in \{0, 1\}^l$. Then each $work_{i_j}$ is changed according to a certain publicly known procedure. In this case, wm_j can only assume a binary value, and thus, there are only two possible marked versions of $work_{i_j}$ which we denote with $work_{i_j}^{wm_j}$, $wm_j \in \{0,1\}$. Thus, depending on the value wm_j, $work_{i_j}$ is replaced either by $work_{i_j}^0$, or by $work_{i_j}^1$.

In the reading phase, one can compare the extracted component $work_{i_j}^*$ with $work_{i_j}^0$ and $work_{i_j}^1$. The extracted mark wm_j is then in $\{0,1,?\}$ where "?" denotes that no decision can be made. ∘

4.3.5.1 Watermarking Scheme of Cox et. al

In this section, we consider a well-known watermarking scheme proposed by Cox et al. (1996). It is an instantiation of the generic watermarking scheme explained in Section 4.3.5. To achieve robustness, Cox et al. (1996) argue that the watermark must be placed in perceptually significant regions of the underlying work. However, modifying these regions would introduce visible distortion and violate the desired fidelity. To solve this problem, they use ideas from **spread spectrum** communication that is a well-known steganographic technique. The basic idea is as follows: A narrow-band message signal (watermark) is secretly and reliably transmitted over a wide-band channel (cover work). Using this technique the message signal appears like white noise, and is pseudo-randomly spread over the whole frequency range, i.e., the low-energy signal is embedded in each of the corresponding frequency bands.[6] An adversary must, therefore, manipulate a large number of frequency bands to be able to block the message from transmitting since the adversary does not know over which frequencies the message signal is spread.

In the context of watermarking, this means that the watermark is statistically spread over the whole work, and since the adversary does not know which frequency components are marked, it must modify a large number of frequency components to be able to remove the watermark. This, however, should with high probability render the resulting work useless regarding its perceptual quality.

The components $work_i$ of $\vec{work}$ are coefficients of *Discrete Cosine Transformation* (DCT) of $work$. The components to be marked are chosen from a certain frequency range (e.g., 1000 largest DCT coefficients but one can also select them pseudo-randomly.) The watermark components wm_j are chosen from a Normal distribution $N(0,1)$. Thus, we have $work_{i_j}^{WM} := work_{i_j} + s_j wm_j$ where s_i is the appropriate strength.

Reading the watermark is done by detection. The detection metric is

[6]This reconcile the following conflicting issues: High-frequencies are relevant for imperceptibility of the watermark but not for its robustness whereas low-frequencies are relevant for robustness but useless for imperceptibility.

normalized correlation as follows:

$$corr(\vec{wm}, \vec{wm}^*) := \frac{< \vec{work} - \vec{work}^*, \vec{wm} >}{\| \vec{work} - \vec{work}^* \|}$$

where $< \vec{a}, \vec{b} >$ denotes the scalar product of two vectors $\vec{a}$ and $\vec{b}$. The watermark is detected if $corr(\vec{wm}, \vec{wm}^*) > \delta$, where δ is an empirically determined threshold.

Next section considers constructions for watermarks which can resist *collusion attacks* of a certain maximum size.

4.3.6 Collusion-tolerant Encoding/Watermarking

4.3.6.1 Introduction

In addition to *robustness*, we require a watermark to be **collusion-tolerant** meaning that even if the adversary has access to a certain number of fingerprinted copies of the *same* work, it should not be able to remove any of the embedded watermarks or to generate a new one.[7] Recall that each copy of the original work contains its own unique watermark.

Wagner (1983) mentions collusion attacks in his pioneering work on a taxonomy for fingerprinting schemes (where he also proposed several methods for fingerprinting different objects.) Later, Blakley, Meadows, and Purdy (1986) proposed a construction for collusion-tolerant codes to be embedded into works. However, the code length is exponential in the collusion size. Chor, Fiat, and Naor (1994) considered the problem of *traitor tracing* in the context of broadcast distribution. They proposed constructions to fingerprint keys which the users obtain for their decoder devices. Boneh and Shaw (1995) introduced a model for constructing codes which tolerates collusions of a maximum size *collsize* (see also Boneh and Shaw (1994) and Boneh and Shaw (1998)). Their model is based on an assumption called **marking assumption**, saying that a collusion can see and change only those bits in which any two of the colluders' copies differ. Note that the model of Boneh and Shaw (1995) abstracts from the watermarking issues such as watermark security and robustness, and considers only watermark strings, i.e., it considers the data-independent part of fingerprinting (see the layer model in Figure 4.1). They present the construction of a code called **basic code** which requires a watermark roughly of the length $O\big((collsize)^3 l\big)$ bits to convey l bits in a work.[8] The basic code can tolerate collusions up to the number of all users, i.e., $collsize = n$. In this model, no shorter code with the same collusion-tolerance is known. Boneh and Shaw (1995) then applied the ideas from traitor tracing to construct codes which can handle

[7]One may generalize this attack to the case where adversary has access to *different* (watermarked) works.

[8]Note that the cost for the final watermarking is not included.

more users ($n > \mathit{collsize}$) but smaller collusions. These codes are called **concatenated codes** because they are constructed by concatenating symbols of the basic code. Their construction requires a watermark roughly of the length $O\big((\mathit{collsize})^4 l\big)$ bits to convey l bits in a work.

Attempts to improve Boneh and Shaw codes regarding the marking assumption and the length of the codes are done in Yacobi (2001) and Muratani (2001), however, no real breakthrough have been achieved.

A completely different approach is proposed by Kilian et al. (1998). Their watermarking scheme is the same as that proposed by Cox et al. (1996) described in Section 4.3.5.1. Using a statistical analysis they argue that their scheme is already collusion-tolerant. In their scheme $O\big((\mathit{collsize})^2 l\big)$ watermark components are required to convey l bits. A later work by Ergun, Kilian, and Kumar (1999) gives an almost tight lower bound for this class of schemes. They relax the assumptions of the model of Kilian et al. (1998) in the sense that the components need not be independent and normally distributed.[9] Within their model they show that no watermarking can offer a better collusion-tolerance.

The construction of Boneh and Shaw (1995) has, however, the advantage that it is data-independent. In this thesis, we will focus on this construction, review its construction in Section 4.3.6.4, and use it as a building block for our asymmetric fingerprinting schemes.

4.3.6.2 Model

We use a different notation for the components of *collusion-tolerant* watermarking than the notation we used for *conventional* watermarking in Section 4.3. This is useful for modularity because we can also handle the case where conventional watermarking schemes are used as building blocks, i.e., we can separate the data-independent part from the data-dependent part (see also Section 4.2.1).

The involved parties are a selling party, the merchant $\mathcal{M}$, and a purchasing party, the buyer $\mathcal{B}$. $\mathcal{M}$ is the one who runs the algorithms for embedding and extracting watermarks to/from the copies of the original work. We assume a similarity relation $\rightarrow_{\mathsf{sim}}$ and the corresponding similarity test $\mathsf{simtest}()$ to be given (see Section 4.1.2).

In each sold copy of the original work work, $\mathcal{M}$ embeds a different secret value $\vec{emb}_i \in \mathit{EMB}$ where $\vec{emb}_i$ is encoded as a collusion-tolerant watermark. Here, EMB denotes the domain of embedding values. On redistribution of a (fingerprinted) work work^{red}, that is similar to one of the original works, the extracting algorithm should extract the corresponding embedded value $\vec{emb}_j$ from work^{red}. The components of the schemes are as follows:

[9] However, the perceptibility measure is still defined by a threshold in Euclidean distances.

- **Key generation:** This algorithm generates parameters and keys key_{emb} required for the embedding and extracting procedure. We denote this algorithm by

$$key_{emb} \leftarrow \mathsf{GenKeyEmbed}(par_{emb})$$

where par_{emb} denotes the corresponding input parameters (e.g., security parameters).

This algorithm is performed once for all copies of the underlying work to be sold.

- **Embedding:** The embedding procedure, performed by $\mathcal{M}$, is denoted by

$$(work^{fing}, rec_{\mathcal{M}}^{work}) \leftarrow \mathsf{EmbedProt}(work, \vec{emb}, key_{emb})$$

where we now denote the watermarked work with $work^{fing}$, and corresponding record (for the merchant) with $rec_{\mathcal{M}}^{work}$.

- **Extracting:** The extracting procedure (performed by $\mathcal{M}$) is denoted by

$$\vec{emb} \leftarrow \mathsf{ExtractProt}(work^{red}, key_{emb}, REC^{work})$$

where $work^{red}$ denotes the redistributed copy for which $true \leftarrow \mathsf{simtest}(work, work^{red})$ holds, and REC^{work} denotes all records on the work $work$.

4.3.6.3 Security Requirements

We require a collusion-tolerant watermarking scheme to fulfill the following security requirement.

Security for merchant (Traceability): Given a redistributed (watermarked) work similar to one of his works, the merchant $\mathcal{M}$ can extract the embedding value of (at least) one traitor, if at most *collsize* copies were used in a collusion attack.

More precisely, we call a watermarking scheme *collsize* collusion-tolerant (or secure for the merchant) if and only if for all probabilistic polynomial adversaries $\mathcal{A}$ the following holds:

- If $\mathcal{M}$ runs $\mathsf{EmbedProt}(work, \vec{emb}, key_{emb})$ at most *collsize* time where $\mathcal{M}$ chooses the work $work \in WORK$, and $\mathcal{A}$ chooses the different embedding values $\vec{emb}_i \in EMB$ for the corresponding copies of *work* to be fingerprinted,
- and if $\mathcal{A}$ outputs a work $work^{red}$ such that $true \leftarrow \mathsf{simtest}(work, work^{red})$,

- then the extracting algorithm $\mathsf{ExtractProt}(work^{red}, key_{emb}, REC^{work})$ outputs one[10] of the embedding values $\vec{emb}_j$ which was input to $\mathsf{EmbedProt}()$ on *work* previously, except with negligible probability in a security parameter where the probability space is defined over all random choices of the involved parties.

4.3.6.4 Construction of Boneh and Shaw

The model proposed by Boneh and Shaw (1995) is a discrete framework which is data-independent and abstracts the watermarking part (see also Boneh and Shaw (1998) and Shaw (2000)).

Let $\Sigma_0 := \{sym_1, sym_2, \cdots, sym_n\}$ be a finite alphabet, and $\Gamma_0 := \{word_1, \cdots, word_n\} \subseteq \{0,1\}^l$ be a collection of n codewords where the encoding $sym_i = \Gamma_0(word_i)$ holds. Such a code is called an (l, n)**-code**. Each codeword represents a watermark (fingerprint) encoding the corresponding symbol.[11] Each bit of a codeword represents a mark, and can have one of the values $\{0, 1, ?\}$ where ? means unreadable.

For a nonempty set of codewords $A := \{word_1, \cdots, word_r\} \subseteq \{0,1\}^l$, we define $MATCH_A := \{i | word_{1,i} = \cdots = word_{r,i}\}$ as the set of all bit positions on which all codewords $word_1, \cdots, word_r$ match.

Let Γ_0 be an (l, n)-code, and $COL \subseteq \Sigma_0$ the set of symbols of a collusion, and $W := \Gamma_0(COL) \subseteq \Gamma_0$ denote the set of codewords assigned to these symbols. With $MATCH_W$ we denote the set of bit positions where marks are identical for the codewords assigned to the elements of COL. We call the set $MATCH_W$ **undetectable bits** of COL. The set of bit positions not identical for all elements of the collusion is called **detectable bits**. Thus, the detectable bits are those marks different for at least one element of the collusion COL whereas the undetectable ones are identical for all elements in COL.

Boneh and Shaw (1995) state the following assumption, called **marking assumption**: A collusive adversary $\mathcal{A}$, cannot generate a copy $work^*$ of the underlying work *work* that contains a codeword $word^*$ which differs from the codewords in W in its undetectable bit positions $MATCH_W$. In other words, marking assumption says that *only* those marks can be detected which differ among the codewords assigned to the collusion, and only these marks can be changed by the collusion. Based on this assumption, the goal is to define the set of words a collusion can produce. The set of words constructible by a collusion of a certain size *collsize* is called **feasible set**, and consists of words which match the undetectable bits of the collusion COL. This set is

[10] Here, we assumed that the algorithm outputs the embedding value of one of the traitors, although it may also find the embedding values of other collusion members.

[11] Note that Boneh and Shaw (1995) set $\Sigma_0 := \{\mathcal{U}_1, \cdots, \mathcal{U}_n\}$ where $\mathcal{U}_i$ is the identity of the user i.

denoted by $F(COL)$ and defined as follows:

$$F(COL) := \{word \in (\{0,1\} \cup \{?\})^l | word_j = word_{i,j}, j \in MATCH_W, word_i \in W\}$$

where $word_j$ and $word_{i,j}$ denote the j-th bit of the words $word$ and $word_i$.

The marking assumption implies that any collusion COL of size *collsize* is only able to generate a work with a fingerprint which lies in the feasible set of COL. If the generated fingerprint contains an unreadable mark ?, it is arbitrary set to 0.

Based on the marking assumption, Boneh and Shaw give constructions for codes that can tolerate collusions of a maximum size. We will consider their constructions in the following.

collsize-secure codes

Generally, one are interested in constructing codes with the following property: Suppose, a collusion (of buyers) of size *collsize* generates a copy $work^{red}$ with $true \leftarrow \mathsf{simtest}(work, work^{red})$. Further, assume that this copy contains the codeword $word^{red}$. Having extracted $word^{red}$ from $work^{red}$, one needs a mechanism for identifying a subset (or at least one member) of COL by using $word^{red}$.[12] A code with this property is called *collsize*-secure codes.

Boneh and Shaw (1995) show that for $collsize \geq 2$ and $n \geq 3$ there exists *no* **totally** ***collsize*-secure codes**. This means that for these parameters one cannot with *certainty* extract symbols which are in COL. However, if one allows a certain error-probability $0 < \epsilon < 1$ that the wrong symbol is extracted, then one can construct a *collsize*-secure code with **ϵ-error**: More precisely, a collusion-tolerant code is called *collsize*-secure with ϵ-error, if there exists an algorithm which, given a codeword $word^{red}$ generated by a collusion COL of size *collsize*, extracts at least one element of COL with probability $1 - \epsilon$. The probability space is defined over the random choices of the merchant, and the random choices of the collusion.

Remark 4.6. Note that, here, we talk about an algorithm extracting (or identifying) the correct symbols. In the terminology used by Boneh and Shaw (1995), the alphabet Σ_0 is the set of users' identities, and the extracting algorithm is called **tracing algorithm** which outputs the identity of at least one member of the collusion. For *collsize*-secure codes with ϵ error, this means that with the error-probability ϵ, the algorithm may wrongly identify an honest buyer. ○

[12] This is the most we can get, since we do not know the collusion's strategy of using the information available to its members. For instance, a collusion may not use a certain codeword at all, and thus, no method can extract a set of of symbols containing the symbol associated to this codeword.

To construct such a code, Boneh and Shaw first construct an n-secure code with ϵ-error called **basic code** or **inner code**. In the following, we call this code basic code. This code is denoted by Γ_0. The length of this code is $n^{O(1)}$, and hence, too large to be practical. They use the basic code to construct *collsize*-secure code with ϵ-error for *collsize* $< n$. The resulting code is called **outer code** or **concatenated code**. In the following we will call this code outer code. The length of the outer code is of order $c^{O(1)}\log(n)^{O(1)}$ where $c :=$ *collsize* (Note that for *collsize* $:= O(\log n)$ the length of the code becomes poly-logarithmic in the number of the symbols.) Next, we take a closer look at the basic code Γ_0.

Basic code $\Gamma_0(n, d)$

It consists of n codewords. Each codeword consists of $n-1$ blocks $B_1, \cdots, B_{n-1}$ of length d. The parameter d is important in determining the *error probability* of the extraction (tracing). The basic code is denoted by $\Gamma_0(n, d)$. Each codeword has binary 1 in blocks B_j, $j \geq i$, and binary 0 in other blocks. In general, the i-th basic codeword $word_i^{\Gamma_0}$ of the basic code Γ_0 is $word_i^{\Gamma_0} := 0^{di}1^{d(n-1-i)}$, and has the length $l = length_2(word_i^{\Gamma_0}) = d(n-1)$. Each symbol sym_i is then encoded with a codeword. The block length parameter d indicates how many times each column in the code is duplicated. For brevity, we often omit the repetition and set $d = 1$.

As an example, consider the code $\Gamma_0(5, 3)$ in the Table 4.1:

Table 4.1 The Basic code $\Gamma_0(5, 3)$ of Boneh and Shaw for $n = 5, d = 3$

sym	$\Gamma_0(5,3)$			
sym_1	1 1 1	1 1 1	1 1 1	1 1 1
sym_2	0 0 0	1 1 1	1 1 1	1 1 1
sym_3	0 0 0	0 0 0	1 1 1	1 1 1
sym_4	0 0 0	0 0 0	0 0 0	1 1 1
sym_5	0 0 0	0 0 0	0 0 0	0 0 0
	B_1	B_2	B_3	B_4

Next, we consider the components of the basic code:

Key generation: The merchant generates the keys and parameters required for the embedding and extracting of Boneh and Shaw Basic Codewords (BSBC). The corresponding algorithm is denoted as follows

$$key_{emb} \leftarrow \mathsf{GenKeyEmbedBSBC}(par_{emb})$$

where $key_{emb} := (sk^{\mathsf{WM}}, par_{\mathsf{CT}})$, sk^{WM} is the secret embedding and extracting key of the underlying watermarking scheme, and par_{CT} denote the (secret and public) parameters required for the basic code such as n, d, Σ_0 and the secretly and randomly selected permutation $\pi \in_{\mathcal{R}} S_l$ (S_l denotes the set of all permutations of $\{1, \cdots, l\}$).[13]

Embedding: We assume that we are given a robust watermarking scheme $(\mathsf{GenKeyWM}(), \mathsf{EmbedMark}(), \mathsf{ExtractMark}())$ which also satisfies the conditions of the *marking assumption*. For the symbol sym_i the merchant takes the codeword $word_i^{\Gamma_0}$ from $\Gamma_0(n, d)$, and applies π to its bits. The word to be embedded for the corresponding symbol sym_i is $word^{\pi} := \pi(word_i^{\Gamma_0})$ which is input to the watermarking algorithm.

$$work^{fing} \leftarrow \mathsf{EmbedMark}(work, word^{\pi}, sk^{\mathsf{WM}}).$$

Extracting: Given a redistributed work $work^{red}$ with $true \leftarrow \mathsf{simtest}(work^{red}, work)$, extract a codeword $word^{red}$ from a redistributed work $work^{red}$ using the extraction algorithm of the underlying watermarking scheme, i.e.,

$$word^{red} \leftarrow \mathsf{ExtractMark}(work^{red}, work, sk^{\mathsf{WM}}).$$

For the basic code, Boneh and Shaw give an extracting algorithm that with probability $1 - \epsilon$ can extract at least one symbol of COL of the maximum size $collsize = n$. We denote this algorithm with

$$COL \ni sym \leftarrow \mathsf{ExtractBSBC}(sym^{red}, \pi)$$

where sym^{red} is the symbol extracted from the redistributed work. Note that sym^{red} corresponds to $word^{red}$ since basic codewords encode individual symbols.

For this construction, Boneh and Shaw prove the following theorem:

Theorem 4.1 (Boneh and Shaw 1995) *For $n \geq 3$ and $\epsilon > 0$ let $d = 2n^2 \log(2n/\epsilon)$. Then the code $\Gamma_0(n, d)$ is n-secure with ϵ-error.* □

Outer code $\Gamma(L, N, n, d)$

Using the basic code Γ_0, Boneh and Shaw (1995) show how to construct shorter codes against collusions of size $collsize < n$. The basic idea is to define a code over the alphabet Σ_0 of the basic code $\Gamma_0(n, d)$. More precisely, given an n-secure (l, n)-code with ϵ-error (here $\Gamma_0(n, d)$), construct a *collsize*-secure (lL, N)-code with $N > n$ symbols by concatenating $L \geq 1$ randomly

[13]Note that the same permutation π is used for all symbols sym_i (obtaining the copies of the underlying work), and must be kept secret. This hides from users which mark in the work encodes which bit in the watermark string.

chosen codewords from $\Gamma_0(n, d)$. The resulting code is denoted by $\Gamma := \Gamma(L, N, n, d)$.

Next, we consider the construction for embedding and extracting procedures of an collusion-tolerant watermarking scheme based on the outer code of Boneh and Shaw (BSOC).

Key generation: The merchant $\mathcal{M}$ generates the keys and parameters $\mathit{key}_{\mathit{emb}}$ for the embedding and extracting procedure of the Boneh and Shaw Outer Code (BSOC). This algorithm is denoted as follows

$$\mathit{key}_{\mathit{emb}} \leftarrow \mathsf{GenKeyEmbedBSOC}(\mathit{par}_{\mathit{emb}})$$

where $\mathit{key}_{\mathit{emb}} := (\mathit{sk}^{\mathsf{WM}}, \mathit{par}_{\mathsf{CT}})$, $\mathit{sk}^{\mathsf{WM}}$ is the secret embedding and extracting key of the underlying watermarking scheme, and $\mathit{par}_{\mathsf{CT}}$ denote the (secret and public) parameters required for the outer code such as n, N, d, L, Σ_0, and Π_0 denotes the set of L permutations $\pi_i \in S_l$, $i \in [1, L]$.

Embedding algorithm: The construction for the embedding procedure is summarized in Algorithm 1. The embedding algorithm is denoted by

$$(\mathit{work}^{\mathit{fing}}, \mathit{rec}_{\mathcal{M}}^{\mathit{work}}) \leftarrow \mathsf{EmbedProt}(\mathit{work}, \vec{\mathit{emb}}, \mathit{key}_{\mathit{emb}}).$$

We separate this algorithm into two main components: The first one is an algorithm for collusion-tolerant encoding (outer codeword) denoted by

$$\mathit{word} \leftarrow \mathsf{GenWord}(\vec{\mathit{emb}}, \mathit{key}_{\mathit{emb}}[\mathsf{GenWord}])$$

where $\mathit{key}_{\mathit{emb}}[\mathsf{GenWord}]$ denotes the corresponding components from the array $\mathit{key}_{\mathit{emb}}[]$ required as inputs to $\mathsf{GenWord}()$ (here $\mathit{par}_{\mathsf{CT}}$). The second algorithm is for embedding this codeword into the underlying work denoted by

$$\mathit{work}^{\mathit{fing}} \leftarrow \mathsf{EmbedWord}(\mathit{work}, \mathit{word}, \mathit{key}_{\mathit{emb}}[\mathsf{EmbedWord}])$$

where $\mathit{key}_{\mathit{emb}}[\mathsf{EmbedWord}]$ denotes the corresponding components from the array $\mathit{key}_{\mathit{emb}}[]$. Here, this is simply $\mathit{sk}^{\mathsf{WM}}$.

Algorithm 1 Embedding algorithm based on BS outer code: $\mathsf{EmbedProt}()$

$\mathsf{EmbedProt}(work, \vec{emb}, key_{emb})$

1. *Generating outer codeword*: Run the algorithm $\mathsf{GenWord}(\vec{emb}, par_{\mathsf{CT}})$ to generate the outer codeword. The individual steps are as follows:
   (a) Check whether $\vec{emb}$ has already been encoded. If yes, use the corresponding word, otherwise continue.
   (b) Choose a new random codeword $\vec{word}' \in_{\mathcal{R}} \Sigma_0^L$, i.e.,
   $$\vec{word}' := (sym_1, sym_2, \cdots, sym_L)$$
   where $\Sigma_0 := \{0, \cdots, n-1\}$.
   (c) Encode each symbol sym_i, $i \in [1, L]$ with codewords from $\Gamma_0(n, d)$. The resulting encoding is denoted by $\Gamma_0(sym_i)$, and the resulting outer codeword with
   $$word := \big(\Gamma_0(sym_1), \cdots, \Gamma_0(sym_L)\big).$$
   (d) Remember $rec_{\mathcal{M}}^{work} \leftarrow (word, \vec{emb}, work)$.
   (e) Permute the bits of each $\Gamma_0(sym_i)$ using π_i. The result is denoted by $\Gamma_0^{\pi_i}(sym_i) := \pi_i\big(\Gamma_0(sym_i)\big)$. The permuted outer codeword is denoted by $word^{\pi} := \big(\Gamma_0^{\pi_1}(sym_1), \Gamma_0^{\pi_2}(sym_2), \cdots, \Gamma_0^{\pi_L}(sym_L)\big)$.
2. *Embedding word*: Run the algorithm $\mathsf{EmbedWord}(work, word^{\pi}, sk^{\mathsf{WM}})$ to embed the outer codeword. In our case, this is the embedding algorithm of underlying watermarking scheme, i.e.,
   $$work^{fing} \leftarrow \mathsf{EmbedMark}(work, word^{\pi}, sk^{\mathsf{WM}}).$$

Extracting: Given a redistributed work $work^{red}$ with $true \leftarrow \mathsf{simtest}(work, work^{red})$, $\mathcal{M}$ runs the extracting procedure as illustrated in Algorithm 2. The extracting procedure is denoted by

$$\vec{emb} \leftarrow \mathsf{ExtractProt}(word^{red}, key_{emb}, REC_{\mathcal{M}}^{work})$$

It consists of two algorithms: Extracting a codeword $word^{red}$ from the redistributed work by using the extracting algorithm of the underlying watermark scheme, and extracting the embedded value $\vec{emb}$ using the extracting algorithm for the outer code

$$\vec{emb} \leftarrow \mathsf{ExtractBSOC}(word^{red}, par_{\mathsf{CT}}).$$

Algorithm 2 Extracting algorithm based on BS outer code: $\mathsf{ExtractProt}()$

```
$\mathsf{ExtractProt}(word^{red}, key_{emb}, REC_{\mathcal{M}}^{work})$
  1. Extracting redistributed codeword: Extract a codeword $word^{red}$ from a redistributed work $work^{red}$ using the extraction algorithm of the underlying watermarking scheme, i.e.,
       $word^{red} \leftarrow \mathsf{ExtractMark}(work^{red}, work, sk^{\mathsf{WM}})$.
  2. Extracting codeword: Run $\mathsf{ExtractBSOC}(word^{red}, par_{\mathsf{CT}})$ to extract the outer codeword as specified below:
     (a) Apply the tracing algorithm $\mathsf{ExtractBSBC}(sym_i^{red}, \pi_i)$ to each symbol $sym_i^{red}$ of $word^{red}$ to extract a symbol $sym_i^{ext} \in \Sigma_0$. The resulting codeword is denoted by
           $\vec{word}^{ext} = (sym_1^{ext}, sym_2^{ext}, \cdots, sym_L^{ext})$.
     (b) If $word^{ext}$ matches with a codeword $word$ in $REC_{\mathcal{M}}^{work}$ in at least $L/c$ symbols, then return the corresponding embedding value $\vec{emb}$, otherwise return failed.
```

4.3.6.5 Security

Boneh and Shaw (1995) show the following result regarding the outer code:

Theorem 4.2 (Boneh and Shaw 1995) *Given $N, collsize \in \mathbb{N}$, $\epsilon \in \mathbb{R}$, $\epsilon > 0$, let $n = 2collsize$, $L = 2collsize \log(2N/\epsilon)$ and $d = 2n^2 \log(4nL/\epsilon)$. Then $\Gamma := \Gamma(L, N, n, d)$ is a collsize-secure code with ϵ-error with N symbols of length $length_2(word \in \Gamma) = O(Ldn) = O\big(collsize^4 \log(N/\epsilon) \log(1/\epsilon)\big)$.* $\square$

Assuming that the underlying watermarking scheme satisfies the conditions of the *marking assumption*, the extracting algorithm of the basis code guarantees that, with high probability, one can extract a symbol from the underlying alphabet Σ_0, and that this symbol comes from one of the traitors. Thus, if there were at most *collsize* traitors then the extracted word $\vec{word}^{ext}$ must match the word of one of these traitors in at least L/c symbols. Hence, the collusion-tolerant watermarking scheme described in Algorithms 1 and 2 is secure for the merchant.

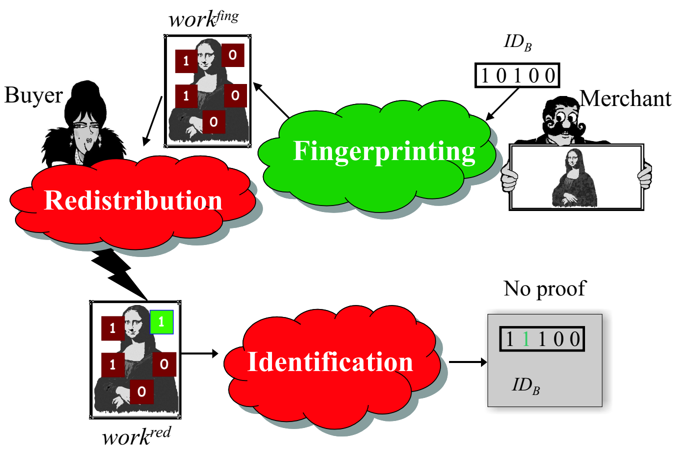
4.4 Symmetric Fingerprinting

4.4.1 Model

The main parties are merchants $\mathcal{M}$ and buyers $\mathcal{B}$. We denote the real identity of a buyer by $ID_{\mathcal{B}}$. We assume a similarity relation $\rightarrow_{\mathsf{sim}}$ and the corresponding similarity test $\mathsf{simtest}()$ to be given.

The model for symmetric fingerprinting scheme is illustrated in Figure 4.2.

Figure 4.2 Overview of symmetric fingerprinting



A symmetric fingerprinting scheme consists of the following components:

- **Key generation:** $\mathcal{M}$ generates the required parameters for the embedding and extracting procedures. This is denoted by

$$key_{emb} \leftarrow \mathsf{GenKeyEmbed}(par_{emb}).$$

- **Fingerprinting:** This is a protocol between a merchant $\mathcal{M}$ and a buyer $\mathcal{B}$. $\mathcal{M}$'s secret input is the work *work*. The common input is the identity $ID_{\mathcal{B}}$ of $\mathcal{B}$. The output to $\mathcal{M}$ is a purchase record $pur\_rec_{\mathcal{M}}^{work}$ on *work* for the buyer $\mathcal{B}$ or *failed* meaning that the protocol was not successfully completed. The output to $\mathcal{B}$ is the fingerprinted work $work^{fing}$ and the associated purchase record $pur\_rec_{\mathcal{B}}^{work}$ or *failed*. The fingerprinting protocol is denoted as follows:

$$(\mathcal{B} : work^{fing}, pur\_rec_{\mathcal{B}}^{work};\ \mathcal{M} : pur\_rec_{\mathcal{M}}^{work})$$
$$\leftarrow \mathsf{Fing}(\mathcal{B} : -;\ \mathcal{M} : work;\ * : ID_{\mathcal{B}}).$$

- **Identification:** On the redistribution of a work similar to one of his works *work*, $\mathcal{M}$ runs the identification algorithm to identify the original buyer(s) of this work. $\mathcal{M}$ inputs the redistributed work $work^{red}$ and all purchase records $PUR\_REC_{\mathcal{M}}^{work}$ on *work*. The output to $\mathcal{M}$ is the identity $ID_{\mathcal{B}^*}$ of at least one traitor $\mathcal{B}^*$. We denote the identification algorithm as follows:

$$(\mathcal{M} : ID_{\mathcal{B}^*}) \leftarrow \mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work}).$$

4.4.2 Security Requirements

In this section, we consider the main security requirements for a symmetric fingerprinting scheme.

Let $\mathcal{ID}_{\mathcal{B}*}$ denote any set of *collsize* different valid identities, $\mathcal{A}$ be a probabilistic polynomial-time interactive algorithm, and $state_{\mathcal{A}}$ denote the state of $\mathcal{A}$ which we will explicitly use in the following.

Security for merchant (Traceability): The merchant can identify at least one traitor, if at most *collsize* copies were used in a collusion attack.

More precisely, we call a symmetric fingerprinting scheme secure for $\mathcal{M}$ if and only if for all adversaries $\mathcal{A}$ the following holds:

- If $\mathcal{M}$ runs $\mathsf{Fing}(\mathcal{A} : state_{\mathcal{A}};\ \mathcal{M} : work;\ * : ID_{\mathcal{B}*})$ with $\mathcal{A}$ for at most *collsize* times where $\mathcal{M}$ chooses the work $work \in WORK$, and $\mathcal{A}$ chooses the valid identities $ID_{\mathcal{B}*} \in \mathcal{ID}_{\mathcal{B}*}$,
- and if $\mathcal{A}$ outputs a work $work^{red}$ such that $true \leftarrow \mathsf{simtest}(work, work^{red})$,
- then the identification protocol $\mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work})$ outputs the identity of a traitor $ID_{\mathcal{B}*} \in \mathcal{ID}_{\mathcal{B}*}$, except with negligible probability in a security parameter where the probability space is defined over all random choices of the involved parties.

Remark 4.7. We may relax this requirement by requiring the identification to have a fixed error probability ϵ.

Remark 4.8. For merchant's security, we may let the adversary choose the underlying works, instead of the merchant. This would result in a more general security requirement. Note that the adversary may also obtain useful information by executing the fingerprinting protocol on different works. For our purposes, however, we will use the standard definition given above. ∘

Security for buyer (frame proofness): An honest buyer is not identified as a traitor, no matter how large the size of the collusion against this buyer is.

More precisely, we call a symmetric fingerprinting scheme secure (frame proof) for the buyer if and only if the following holds: There is no adversary $\mathcal{A}$

- that runs $\mathsf{Fing}(\mathcal{A} : state_{\mathcal{A}};\ \mathcal{M} : work;\ * : ID_{\mathcal{B}*})$ arbitrarily often with an *honest* merchant $\mathcal{M}$, where $\mathcal{M}$ chooses works $work \in WORK$, and $\mathcal{A}$ chooses the valid identities $ID_{\mathcal{B}*}$ different from the identity $ID_{\mathcal{B}}$ of any honest buyer $\mathcal{B}$,
- and can generate a work $work^{red}$ such that

- the identification protocol $\mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC^{work}_{\mathcal{M}})$ outputs the identity $ID_{\mathcal{B}}$ of an honest buyer $\mathcal{B}$, except with negligible probability in a security parameter where the probability space is defined over all random choices of the involved parties.

4.4.3 Construction

To construct a collusion-tolerant symmetric fingerprinting scheme, one may think of directly using a collusion-tolerant watermarking scheme described in Section 4.3.6.2. However, the scheme must also guarantee the security for the buyer as required in Section 4.4.2. To make the framing probability negligible, the scheme should take care of the following issues: First, there must be a secret random mapping between the embedding value $\vec{emb}$ and the identity $ID_{\mathcal{B}}$ of an honest buyer. Secondly, the cardinality of the set of embedding values EMB must be sufficiently large. For appropriately chosen parameters, one can use the collusion-tolerant watermarking scheme (based on outer code of Boneh and Shaw) from Section 4.3.6.4 to construct a secure collusion-tolerant symmetric fingerprinting scheme. Note that the construction of the outer code automatically assigns random codewords to the corresponding value $\vec{emb}$, and the number of symbols N can be chosen sufficiently large while all other parameters can be computed in polynomial time.

The resulting scheme is secure for the merchant as discussed briefly in Section 4.3.6.5. It is also secure for the buyer since the word *word* chosen for an honest buyer is completely independent of a *word** which any collusion of buyers may be able to generate. Thus, we have to consider the probability that *word** matches a given *word* in at least L/c symbols, and this probability can be shown to be sufficiently small.[14]

4.5 Asymmetric Fingerprinting

In a symmetric fingerprinting scheme both parties, the merchant and the buyer, know the fingerprinted work. This, however, implies that a redistributed copy does not have to come from one of the original buyers. It might also come from a cheating merchant or a dishonest employee of the merchant. In other words, symmetric fingerprinting schemes do not provide merchants with a proof of treachery that convinces an honest third party. Thus, they offer no **non-repudiation**. To solve this problem, **asymmetric fingerprinting (AFP)** were introduced by Pfitzmann and Schunter (1996). The idea is that in case of illegal redistribution the merchant should be able

[14] This is a binomial probability distribution with parameters $n = L$, $p = 1/2c$ and mean $L/2c$, and then the **Chernoff bound** (see e.g., Feller (1968) or Alon and Spencer (1992)) can be applied to estimate that the probability for this event is sufficiently small.

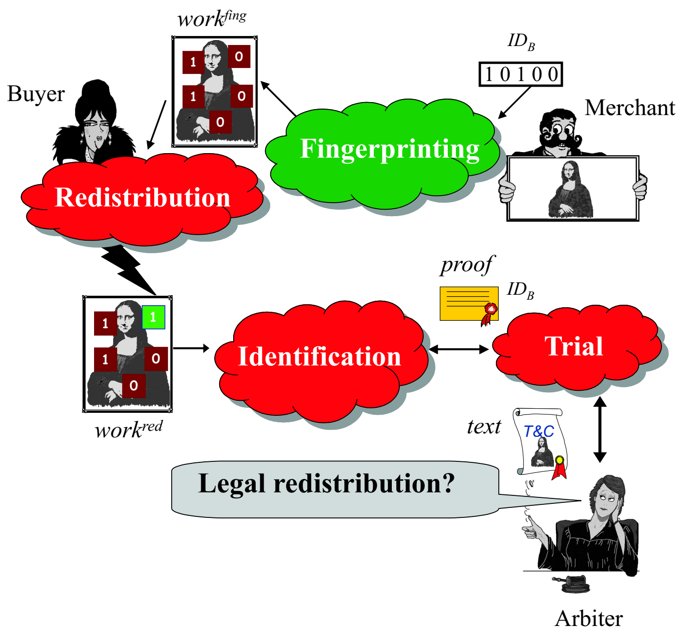
to derive a proof of treachery – similar to a digital signature of a traitor. In asymmetric schemes, only the buyer knows the fingerprinted work after the purchase. However, as soon as a redistributed work is found, the merchant can retrieve enough information to identify at least one traitor, and further, he should be able to prove this fact to any honest party. Moreover, even if legal redistribution is allowed, it should still be possible that an honest buyer cannot be wrongly accused by a cheating merchant, or by any other party colluding with the merchant against this buyer. The first proposal for a general construction of asymmetric fingerprinting schemes was introduced by Pfitzmann and Schunter (1996). Their construction uses symmetric fingerprinting schemes as building block. Later, Pfitzmann and Waidner (1997c) proposed constructions which can tolerate large collusions – the same collusion size as the underlying symmetric scheme can handle. A very similar approach was proposed independently by Biehl and Meyer (1997).

4.5.1 Model

The main parties are merchants $\mathcal{M}$, buyers $\mathcal{B}$ and arbiters $\mathcal{J}$. We identify each purchase by a purchase description *text* which contains a description of the underlying work *work*, and the rights the buyer obtains on this work.[15] Figure 4.3 illustrates the main protocols for the purchase and redistribution in an asymmetric fingerprinting scheme.

[15]More precisely, we are concerned with rights on a *work-class* $WORK_{work}^{sim}$. However, for readability reasons, we use the notation for a single work in the following.

Figure 4.3 The model of asymmetric fingerprinting



An asymmetric fingerprinting scheme consists of the following protocols:

Setup: $\mathcal{M}$ and $\mathcal{B}$ generate the required keys and parameters. The main procedures are as follows:

- The (global) system parameters par_{AsyFP} (e.g., algebraic structure and related parameters) are generated. This procedure is denoted by

$$par_{\mathsf{AsyFP}} \leftarrow \mathsf{GenParAsyFP}(par_{sec}).$$

- Each party X generates a key pair (sk_X, pk_X) of an underlying (secure) signature scheme, and distributes pk_X reliably and authenticated to other involved parties, if required. We will use pk_X as representative for X's identity. The key pair of $\mathcal{B}$ is denoted by $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$.
- $\mathcal{M}$ generates the required keys and parameters for the embedding procedure, i.e.,

$$key_{emb} \leftarrow \mathsf{GenKeyEmbed}(par_{emb}).$$

Fingerprinting: This is a protocol between a merchant $\mathcal{M}$ and a buyer $\mathcal{B}$. $\mathcal{M}$'s secret input is the work *work*. The secret input of $\mathcal{B}$ is her secret key $sk_{\mathcal{B}}$. $\mathcal{M}$ and $\mathcal{B}$ also input *text*, and the identity of the buyer, represented by her public-key $pk_{\mathcal{B}}$. The output to $\mathcal{M}$ is a purchase record $pur\_rec_{\mathcal{M}}^{work}$ or *failed* indicating that the protocol has failed. The main outputs to $\mathcal{B}$ are the fingerprinted work $work^{fing}$ and a purchase record $pur\_rec_{\mathcal{B}}^{work}$ or *failed*. We denote the fingerprinting protocol as follows:

$$\begin{aligned}(\mathcal{B} : work^{fing}, pur\_rec_{\mathcal{B}}^{work};\ \mathcal{M} : pur\_rec_{\mathcal{M}}^{work})\\ \leftarrow \quad \mathsf{Fing}(\mathcal{B} : sk_{\mathcal{B}};\ \mathcal{M} : work;\ * : text, pk_{\mathcal{B}}).\end{aligned}$$

Identification: This is an algorithm the merchant $\mathcal{M}$ runs to identify the buyer of a redistributed copy $work^{red}$ of the work *work* for which $true \leftarrow \mathsf{simtest}(work, work^{red})$ holds. He inputs $work^{red}$ and all his purchase records $PUR\_REC_{\mathcal{M}}^{work}$ for *work*. Note that we require the construction to allow *legal redistribution*. It means that a buyer can purchase "right of distribution" allowing her to redistribute the underlying work, e.g., 5 years after the purchase time. However, it should still not be possible to wrongly accuse an honest buyer (by a cheating merchant or any other party colluding with him against this buyer.) The output is the identity of a buyer $pk_{\mathcal{B}}$, the text *text* used in the corresponding purchase, and a proof string *proof* (e.g., a signature of $\mathcal{B}$). We denote the identification protocol as follows:

$$(\mathcal{M} : pk_{\mathcal{B}}, text, proof) \leftarrow \mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work}).$$

Trial: This is a protocol performed when a merchant $\mathcal{M}$ wants to convince a third party (an arbiter $\mathcal{J}$) that a certain buyer $\mathcal{B}$ has violated the terms and conditions regarding a certain purchase described in *text*. The protocol involves $\mathcal{M}$ and $\mathcal{J}$, and, depending on the construction, it may also involve the buyer $\mathcal{B}$. Thus, the trial is either a 2-party or a 3-party protocol. $\mathcal{M}$ inputs $pk_{\mathcal{B}}, text$ and the proof string *proof* which he retrieved in the identification. If $\mathcal{B}$ participates, she inputs her purchase record $pur\_rec_{\mathcal{B}}^{work}$ she obtained on *work* during the fingerprinting protocol.[16] The main output is the arbiter's result. It may be *guilty*, meaning that she considers $\mathcal{B}$ as a traitor[17] or *not_guilty*, meaning that she rejects the accusation. We denote the trial protocol as follows:

- Two-party trial:

$$(\mathcal{M} : -;\ \mathcal{J} : ind) \leftarrow \mathsf{Trial}(\mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{J} : -)$$

- Three-party trial:

$$\begin{aligned}&(\mathcal{B} : -;\ \mathcal{M} : -;\ \mathcal{J} : ind)\\ &\quad \leftarrow \mathsf{Trial}(\mathcal{B} : pur\_rec_{\mathcal{B}}^{work};\ \mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{J} : -)\end{aligned}$$

  where $ind \in \{guilty, not\_guilty\}$. Whenever convenient, we use the notation $\mathsf{Trial}([\mathcal{B} : pur\_rec_{\mathcal{B}}^{work}];\ \mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{J} : -)$ to consider both possibilities.

[16]Note that *text* is not taken as common input since $\mathcal{B}$ and $\mathcal{M}$ may not be in agreement with.

[17]One may consider the output as $(guilty, pk_{\mathcal{B}})$ explicitly indicating the identity $pk_{\mathcal{B}}$ of the traitor.

4.5.2 Security Requirements

In this section, we consider the main security requirements of an asymmetric fingerprinting scheme. In contrast to symmetric fingerprinting (Section 4.4.2), the requirements concern the trial protocol where the merchant presents to an arbiter a proof of treachery *proof* against one or more traitors.

Let $\mathcal{ID}_{\mathcal{B}^*}$ denote any set of *collsize* different valid identities, $\mathcal{A}$ be a probabilistic polynomial-time interactive algorithm, and let $state_{\mathcal{A}}$ denote the state of $\mathcal{A}$ which we will explicitly use in the following. In general, the adversary comprises any possible collusions of other parties against the party whose security is being considered.

Security for merchant: An honest merchant $\mathcal{M}$ must be able to identify at least one traitor, and win in the corresponding trial for every illegally redistributed copy similar to one of his original works, unless the collusion is larger than the tolerated limit.

More precisely, we call an asymmetric fingerprinting scheme secure for the merchant if and only if for all $\mathcal{A}$ the following holds:

- If $\mathcal{M}$ runs the fingerprinting protocol $\mathsf{Fing}(\mathcal{A} : state_{\mathcal{A}};\ \mathcal{M} : work;\ * : text, pk_{\mathcal{B}^*})$ with $\mathcal{A}$ at most *collsize* time where $\mathcal{M}$ chooses the work $work \in WORK$, and $\mathcal{A}$ chooses *text* and the valid identities $pk_{\mathcal{B}^*} \in \mathcal{ID}_{\mathcal{B}^*}$, and where other transactions ($\mathsf{Trial}()$) with $\mathcal{A}$ may occur between the executions of $\mathsf{Fing}()$,
- and if $\mathcal{A}$ outputs a work $work^{red}$ such that $true \leftarrow \mathsf{simtest}(work, work^{red})$,
- then the identification protocol $\mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work})$ outputs $(pk_{\mathcal{B}^*}, text, proof)$ with $pk_{\mathcal{B}^*} \in \mathcal{ID}_{\mathcal{B}^*}$ such that
- the trial protocol $\mathsf{Trial}([\mathcal{A} : state_{\mathcal{A}}];\ \mathcal{M} : proof, pk_{\mathcal{B}^*}, text;\ \mathcal{J} : -)$ outputs *guilty* to $\mathcal{J}$, except with negligible probability in a security parameter where the probability space is defined over the random choices of all the involved algorithms.

We may additionally require the merchant to be protected from wrong accusations since it would harm his reputation if he loses a trial even though the identification was successful. Hence, it should be infeasible for an adversary, even if the collusion size is greater than *collsize*, to generate a *text* such that the identification succeeds, but the trial outputs *not_guilty*.

More precisely, we may extend the previous requirement as follows: We call an asymmetric fingerprinting scheme secure for the merchant if and only if for all $\mathcal{A}$ the following holds:

- If $\mathcal{M}$ runs the fingerprinting protocol $\mathsf{Fing}(\mathcal{A} : state_{\mathcal{A}};\ \mathcal{M} : work;\ * : text, pk_{\mathcal{B}^*})$ with $\mathcal{A}$ at most *collsize* time where $\mathcal{M}$ chooses the work

$work \in WORK$, and $\mathcal{A}$ chooses $text$ and the valid identities $pk_{\mathcal{B}^*} \in \mathcal{ID}_{\mathcal{B}^*}$, and where other transactions ($\mathsf{Trial}()$) with $\mathcal{A}$ may occur between the executions of $\mathsf{Fing}()$,

- and if $\mathcal{A}$ outputs a work $work^{red}$ such that $true \leftarrow \mathsf{simtest}(work, work^{red})$,
- then the identification protocol $\mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC^{work}_{\mathcal{M}})$ outputs $(pk_{\mathcal{B}^*}, text, proof)$ with $pk_{\mathcal{B}^*} \in \mathcal{ID}_{\mathcal{B}^*}$, except with negligible probability in a security parameter where the probability space is defined over the random choices of all the involved algorithms.

and,

- on successful identification, with the result $pk_{\mathcal{B}^*}$, the trial protocol $\mathsf{Trial}([\mathcal{A} : state_{\mathcal{A}}];\ \mathcal{M} : proof, pk_{\mathcal{B}^*}, text;\ \mathcal{J} : -)$ outputs *guilty* to $\mathcal{J}$, except with negligible probability in a security parameter where the probability space is defined over the random choices of all the involved algorithms.

Remark 4.9. The requirement for merchant's security requires the success probability of the merchant to be overwhelming in both the identification and the trial protocols. However, in case of the identification, we may weaken this requirement by allowing a certain bound on the success probability. With a suitable choice such as 1/2, we may still achieve the *determent effect*, however we do not want to harm merchant's reputation by letting him fail in the trial with non-negligible probability. ∘

Security for buyer: An honest buyer $\mathcal{B}$ should not be found guilty by an arbiter $\mathcal{J}$, i.e., the output of the trial protocol for this buyer will not be *guilty*, no matter how large the size of the collusion against this buyer is. This requirement is also called *frame proofness*.

More precisely, we call an asymmetric fingerprinting scheme secure for the buyer if and only if the following holds: There is no adversary $\mathcal{A}$

- that runs the fingerprinting protocol $\mathsf{Fing}(\mathcal{B} : sk_{\mathcal{B}};\ \mathcal{A} : state_{\mathcal{A}}, work;\ * : text, pk_{\mathcal{B}})$ arbitrarily often with an honest buyer $\mathcal{B}$ where $\mathcal{A}$ chooses the purchase descriptions $text$, and works $work \in WORK$, and $\mathcal{A}$ may perform any other transactions ($\mathsf{Trial}()$) between the executions of $\mathsf{Fing}()$,
- and obtains the data of some purchases (e.g., all works $work^{fing}$ for which the purchase descriptions $text$ allow legal distribution),
- can generate a work $work^{*fing}$, a text $text^*$ and a proof string $proof^*$,
- and then run the trial protocol $\mathsf{Trial}([\mathcal{B} : pur\_rec^{work}_{\mathcal{B}}];\ \mathcal{A} : state_{\mathcal{A}}, proof^*, pk_{\mathcal{B}}, text^*;\ \mathcal{J} : -)$ with an arbiter $\mathcal{J}$ where the output to $\mathcal{J}$ is *guilty*, except with negligible probability in a security parameter where the probability space is defined over the random choices of all the involved algorithms.

4.5.3 Construction Framework

The main task during the fingerprinting protocol is to embed the secret identifying information $\vec{emb}$ into the underlying work *work* where only the buyer obtains the fingerprinted work $work^{fing}$ at protocol completion. The construction has to consider the following main issues:

- *Verification*: The merchant must be provided with mechanisms for (i) verifying that $\vec{emb}$ has the correct form, (ii) establishing a relation between $\vec{emb}$ and the (public) parameters representing $\mathcal{B}$'s real identity (here $pk_{\mathcal{B}}$), and (iii) verifying this relation. All these mechanisms should not reveal any useful information about $\vec{emb}$ to the merchant.
- *Embedding*: There must be a mechanism to embed $\vec{emb}$ into *work* such that the merchant is convinced that $\vec{emb}$ is correctly embedded without obtaining any useful information about it, and $\mathcal{B}$ obtains a fingerprinted version of the underlying work without getting any information enabling her to remove the embedded watermark.
- *Extracting*: The merchant must be provided with a mechanism for extracting the value $\vec{emb}$ from a redistributed work $work^{red}$ similar to one of his original works.

The embedding protocol is the most complex part of the fingerprinting protocol. We will first consider its model and security requirements. Then we go through some more details by briefly reviewing a construction for an asymmetric collusion-tolerant fingerprinting scheme proposed by Pfitzmann and Waidner (1998). Later, in Chapter 5, we present an explicit construction for the embedding protocol of an (anonymous) asymmetric collusion-tolerant scheme.

4.5.4 Embedding

4.5.4.1 Model

The involved parties are the merchant $\mathcal{M}$ and the buyer $\mathcal{B}$. The main components are the key generation algorithm, the embedding and the extracting procedures. We assume a similarity relation $\rightarrow_{\mathsf{sim}}$ and the corresponding similarity test $\mathsf{simtest}()$ to be given.

In contrast to the symmetric fingerprinting scheme, the embedding procedure is an interactive protocol between $\mathcal{M}$ and $\mathcal{B}$ where the embedding information $\vec{emb}$ ($\mathcal{B}$'s secret input) should be securely embedded into the underlying work *work* ($\mathcal{M}$'s secret input). Hereby, the embedding information $\vec{emb}$ is hidden from $\mathcal{M}$ in a commitment denoted by C_{emb}.

The extracting procedure is performed by $\mathcal{M}$ to extract the embedded information $\vec{emb}$ from a redistributed work $work^{red}$ similar to a work *work* the merchant has already sold.

- **Key generation:** $\mathcal{M}$ generates the required keys and parameters key_{emb} for the embedding and extracting procedures. We denote the key generation algorithm by

$$key_{emb} \leftarrow \mathsf{GenKeyEmbed}(par_{emb})$$

where par_{emb} denotes the corresponding parameters (e.g., security parameters), and $key_{emb} := (key^s_{emb}, key^p_{emb})$ consists of a secret part key^s_{emb} and a public part key^p_{emb}. This algorithm is performed once for all copies of the underlying work to be sold.

- **Embedding protocol:** The embedding procedure is denoted by

$$\begin{aligned}&(\mathcal{B} : work^{fing};\ \mathcal{M} : rec^{work}_{\mathcal{M}})\\ &\quad \leftarrow \mathsf{EmbedProt}(\mathcal{B} : state^{C_{emb}}_{\mathcal{B}};\ \mathcal{M} : work, key^s_{emb};\ * : C_{emb}, key^p_{emb}).\end{aligned}$$

The common input is the commitment $C_{emb} := com(\vec{emb})$ to $\vec{emb}$ (which is usually handed over from the previous protocol), and the public parameters key^p_{emb}. $\mathcal{B}$ inputs the secret opening information $state^{C_{emb}}_{\mathcal{B}}$ for this commitment, and $\mathcal{M}$ secretly inputs the work $work$ to be fingerprinted, and the keys key^s_{emb} for embedding. The output of the protocol to $\mathcal{B}$ is the fingerprinted work $work^{fing}$, and to $\mathcal{M}$ the record $rec^{work}_{\mathcal{M}}$ which contains $work$, and the tracing information denoted by $\vec{trace\_inf}$.

Note that, in the following, whenever it is clear from the context, we will not explicitly mention the commitment parameters par_{com} (as a part of common input)

- **Extracting procedure:** The extracting procedure is performed by $\mathcal{M}$, and is denoted by

$$\vec{emb} \leftarrow \mathsf{ExtractProt}(work^{red}, key_{emb}, REC^{work}_{\mathcal{M}})$$

where $work^{red}$ denotes a redistributed copy for which $true \leftarrow \mathsf{simtest}(work, work^{red})$ holds, and $REC^{work}_{\mathcal{M}}$ denotes the set of all records $rec^{work}_{\mathcal{M}}$ on $work$.

4.5.4.2 Security Requirements

The embedding protocol must fulfill the following security requirements in *addition* to those of a symmetric embedding procedure.

Security for merchant: $\mathcal{M}$ must be able to verify that the content of C_{emb} is correctly[18] embedded into the underlying work at protocol completion without leaking any useful information about his secret inputs. We classify these requirements as follows:

[18]e.g., as a collusion-tolerant watermark or any other specified encoding

- *Soundness*: Let $\vec{emb}$ denote the content of the commitment C_{emb}. On common input C_{emb}, if $\mathcal{B}$ obtains any fingerprinted work $work^{fing}$ with $true \leftarrow \mathsf{simtest}(work, work^{fing})$ from a run of $\mathsf{EmbedProt}()$, then $\vec{emb}$ is correctly embedded in $work^{fing}$ according to the specified encoding (e.g., collusion-tolerant) and the underlying watermarking scheme.
- *Secrecy*: The embedding protocol should not leak *any additional* information on $\mathcal{M}$'s secret inputs (the embedding keys key^{s}_{emb}, the tracing information $\vec{trace\_inf}$, the original work $work$) *beyond* the fingerprinted work $work^{fing}$.[19]

Security for buyer: The embedding protocol should not reveal any additional information on the embedding value $\vec{emb}$. We express this requirement as follows:

- *Secrecy*: On common input C_{emb}, the embedding protocol should be zero-knowledge.

Note that we are primarily interested in collusion-tolerant fingerprinting schemes. In this case, we may see the embedding protocol for the asymmetric fingerprinting scheme as an extension to the collusion-tolerant watermarking (Section 4.3.6.2) that must *additionally* fulfill the mentioned requirements.

Remark 4.10. Note that the buyer does not have any soundness requirement. This is because, due to our construction, $\mathcal{M}$ requires the value $\vec{emb}$ for the purpose of traitor identification, and thus, it is not in his interest to embed any information other than $\vec{emb}$ into the underlying work. ○

4.5.4.3 A Construction Framework for Embedding

We describe a construction framework for the embedding protocol of our fingerprinting scheme. We separate this protocol into two parts. The first part is the *encoding* procedure where the embedding information $\vec{emb}$ is encoded as specified by the protocol. In our case, this is a collusion-tolerant codeword. The second part is the *watermarking* part where this codeword is embedded into the underlying work. Note that, here, we focus on data-independent encoding. However, one may combine these two parts if one applies a data-dependent watermarking scheme such as the one explained in Section 4.3.5.1.

[19]Note that we cannot simply require the embedding protocol to be zero-knowledge (for $\mathcal{M}$) since $work^{fing}$ is revealed to the buyer.

Encoding: The protocol for the secure encoding $\vec{emb}$ in form of a collusion-tolerant codeword *word* is denoted by

$$(\mathcal{B} : -;\ \mathcal{M} : C_{word}, \vec{trace\_inf}) \\ \leftarrow \mathsf{GenWord}(\mathcal{B} : state_{\mathcal{B}}^{C_{emb}};\ \mathcal{M} : key^{s}_{emb}[\mathsf{GenWord}];\ * : C_{emb}, key^{p}_{emb}[\mathsf{GenWord}])$$

where $C_{word} := com(word)$ is the commitment to *word*, $\vec{trace\_inf}$ is the tracing information, $key^{s}_{emb}[\mathsf{GenWord}], key^{p}_{emb}[\mathsf{GenWord}]$ are the corresponding components from the array $key_{emb}[]$ required as inputs to $\mathsf{GenWord}()$.

We may further refine this protocol as follows:

- *Generating tracing information*: The tracing information $\vec{trace\_inf}$ for the merchant $\mathcal{M}$ is generated. We assume that $\mathcal{M}$ generates this information by himself, and denote the corresponding algorithm as follows:
$$\vec{trace\_inf} \leftarrow \mathsf{GenTraceWord}(par_{trace\_inf})$$
where $par_{trace\_inf} := key_{emb}[\mathsf{GenWord.GenTraceWord}]$ denotes the parameters required for $\mathsf{GenTraceWord}()$ taken from the array $key_{emb}[\mathsf{GenWord}]$.

- *Encoding embedding information*: $\mathcal{B}$ may be required to encode the embedding information $\vec{emb}$ using a publicly known encoding (e.g., an Error-and-Erasure-Correcting Code, etc). We denote this procedure by
$$\vec{emb\_inf} \leftarrow \mathsf{GenEmbWord}(\vec{emb}, par_{emb\_inf})$$
where $par_{emb\_inf} := key^{p}_{emb}[\mathsf{GenWord.GenEmbWord}]$ denotes the corresponding parameters in the array $key^{p}_{emb}[\mathsf{GenWord}]$.

- *Proving correct encoding*: Now, $\mathcal{B}$ has to securely prove to $\mathcal{M}$ that $\vec{emb\_inf}$ has the correct form. For this, $\mathcal{B}$ applies a (zero-knowledge) proof system where she commits to $\vec{emb\_inf}$, denoted by C_{emb_inf}, and proves to $\mathcal{M}$ that its content is the encoding of the content of C_{emb} according to the given encoding function. This protocol is denoted by
$$(\mathcal{B} : -;\ \mathcal{M} : ind) \\ \leftarrow \mathsf{ProveEncEmbCom}(\mathcal{B} : state_{\mathcal{B}}^{C_{emb}}, state_{\mathcal{B}}^{C_{emb\_inf}};\ \mathcal{M} : -; \\ * : C_{emb}, C_{emb\_inf}, par_{emb\_inf})$$
where $ind \in \{accept, reject\}$.

- *Collusion-tolerant encoding*: $\mathcal{M}$ and $\mathcal{B}$ securely combine the tracing *trace_inf* and embedding *emb_inf* information in form of a collusion-tolerant codeword. We denote the corresponding protocol by
$$(\mathcal{B} : -;\ \mathcal{M} : C_{word}) \\ \leftarrow \mathsf{CollTolWord}(\mathcal{B} : state_{\mathcal{B}}^{C_{emb\_inf}};\ \mathcal{M} : par^{s}_{\mathsf{CT}}, \vec{trace\_inf}; \\ * : C_{emb\_inf}, par^{p}_{\mathsf{CT}})$$

where $par^{s}_{\mathsf{CT}} := key^{s}_{emb}[\mathsf{GenWord.CollTolEnc}]$ denotes the corresponding secret parameters from the array $key^{s}_{emb}[\mathsf{GenWord}]$, and $par^{p}_{\mathsf{CT}} := key^{p}_{emb}[\mathsf{GenWord.CollTolEnc}]$ the public ones.

Embedding codeword: The collusion-tolerant codeword *word* is embedded into *work*. The corresponding protocol is denoted by

$$\begin{aligned}&(\mathcal{B} : work^{fing};\ \mathcal{M} : -)\\ &\quad \leftarrow \mathsf{EmbedWord}(\mathcal{B} : key_{\mathsf{com}};\ \mathcal{M} : work, C_{word}, key^{s}_{emb}[\mathsf{EmbedWord}];\\ &\qquad\qquad\qquad\qquad\qquad\qquad * : key^{p}_{emb}[\mathsf{EmbedWord}])\end{aligned}$$

where key_{com} denotes the secret key which allows the buyer $\mathcal{B}$ to open any commitment computed with the underlying commitment scheme, $work^{fing}$ denotes the fingerprinted work, and $key^{s}_{emb}[\mathsf{EmbedWord}]$ and $key^{p}_{emb}[\mathsf{EmbedWord}]$ denote the secret and the public parameters input to this protocol.

Summary: We summarize the main protocols in Algorithms 3 and 4.

Algorithm 3 Overview of embedding protocol $\mathsf{EmbedProt}()$

```
$\mathsf{EmbedProt}(\mathcal{B} : state^{C_{emb}}_{\mathcal{B}};\ \mathcal{M} : work, key^{s}_{emb};\ * : C_{emb}, key^{p}_{emb})$
  $(\mathcal{B} : -;\ \mathcal{M} : C_{word}, \vec{trace\_inf}) \leftarrow \mathsf{GenWord}(\mathcal{B} : state^{C_{emb}}_{\mathcal{B}};\ \mathcal{M} : key^{s}_{emb}[\mathsf{GenWord}];\ * : C_{emb}, key^{p}_{emb}[\mathsf{GenWord}]);$
  $(\mathcal{B} : work^{fing};\ \mathcal{M} : -) \leftarrow \mathsf{EmbedWord}(\mathcal{B} : key_{\mathsf{com}};\ \mathcal{M} : work, C_{word}, key^{s}_{emb}[\mathsf{EmbedWord}]\ * : key^{p}_{emb}[\mathsf{EmbedWord}]);$
  $(\mathcal{M} : rec^{work}_{\mathcal{M}} \leftarrow (work, \vec{trace\_inf}));$
  RETURN $(\mathcal{B} : work^{fing};\ \mathcal{M} : rec^{work}_{\mathcal{M}})$
```

Algorithm 4 Overview of word construction protocol $\mathsf{GenWord}()$

```
$\mathsf{GenWord}(\mathcal{B} : state^{C_{emb}}_{\mathcal{B}};\ \mathcal{M} : key^{s}_{emb}[\mathsf{GenWord}];\ * : C_{emb}, key^{p}_{emb}[\mathsf{GenWord}])$
  $(\mathcal{M} : \vec{trace\_inf}) \leftarrow \mathsf{GenTraceWord}(\mathcal{M} : par_{trace\_inf});$
  $(\mathcal{B} : \vec{emb\_inf}) \leftarrow \mathsf{GenEmbWord}(\mathcal{B} : \vec{emb}, par_{emb\_inf});$
  $(\mathcal{B} : -;\ \mathcal{M} : ind) \leftarrow \mathsf{ProveEncEmbCom}(\mathcal{B} : state^{C_{emb}}_{\mathcal{B}}, state^{C_{emb\_inf}}_{\mathcal{B}};\ \mathcal{M} : -;\ * : C_{emb}, C_{emb\_inf}, par_{emb\_inf});$
  $(\mathcal{B} : -;\ \mathcal{M} : C_{word}) \leftarrow \mathsf{CollTolWord}(\mathcal{B} : state^{C_{emb\_inf}}_{\mathcal{B}};\ \mathcal{M} : par^{s}_{\mathsf{CT}}, \vec{trace\_inf};\ * : C_{emb\_inf}, par^{p}_{\mathsf{CT}});$
  RETURN $(\mathcal{B} : -;\ \mathcal{M} : C_{word}, \vec{trace\_inf})$
```

Remark 4.11. Note that the above illustrations should only give an overview of the main protocols and algorithms. One may use any other illustrations for this, e.g., by presenting the protocols and algorithms for each party separately. ∘

4.5.5 Construction

In this section, we briefly review the construction of the collusion-tolerant asymmetric fingerprinting scheme (with 3-party trial) proposed by Pfitzmann and Waidner (1997b). It uses a symmetric scheme as a building block which is based on the outer codes from (Boneh and Shaw 1995) described in Section 4.4. The idea is as follows:

The buyer $\mathcal{B}$ chooses a codeword to be embedded in the underlying work. She hides this codeword in commitments. In a secure 2-party protocol the following computations are performed: $\mathcal{B}$ proves that the content of the commitment has the correct form, and $\mathcal{M}$ randomly and secretly selects the half of the symbols of the codeword, and the corresponding commitments are opened to him. This part of the codeword represents the tracing information containing enough information for $\mathcal{M}$ to identify a traitor in case he finds a redistributed work – as long as this work is generated by a collusion not larger than the maximum tolerable collusion size.

Next, the codeword is securely embedded into the underlying work. On redistribution of a work, similar to one of his (fingerprinted) works, $\mathcal{M}$ starts the identification procedure. In case of illegal redistribution, the construction should provide $\mathcal{M}$ with a proof of treachery which convinces an arbiter. For this, $\mathcal{M}$ shows to the arbiter that he knows “enough” symbols from those commitments which were not opened to him. However, this is not straight forward to implement since (i) $\mathcal{M}$ usually does not find the entire codeword of a traitor due to collusions, and (ii) he does not know the secret opening key of the commitments, and thus, he will not be able to show the content of these commitments.

In the following, we give a high-level description of the individual components of the scheme:

Setup: $\mathcal{B}$ generates a key pair $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$ and distributes $pk_{\mathcal{B}}$ reliably to all involved parties. $\mathcal{M}$ generates the parameters key_{emb} for embedding and extracting the outer codeword. Here, we have $key^{s}_{emb} := (sk^{\mathsf{WM}}, \Pi_0)$ where sk^{WM} is the secret embedding key, and Π_0 the set of permutations π for the bits of Boneh and Shaw basic codewords. Following our notation from Section 4.5.4.3, the required parameters for the outer code are $par^{s}_{\mathsf{CT}} := \Pi_0$ and $par^{p}_{\mathsf{CT}} := (k, n, d, L, N, \epsilon, collsize)$ where L and n are chosen as $L := 64 collsize \log(N/\epsilon), n := 48 collsize$ and $d := 2n^2(log(4nL/\epsilon)$ (see also Section 4.3.6.4).

Fingerprinting:

1. $\mathcal{B}$ chooses a random codeword $\vec{word}'$ of length L over the alphabet $\Sigma_0 := \{1, \cdots, n\}$. We denote this codeword with

$$\vec{word}' := (sym_1, sym_2, \cdots, sym_L).$$

2. $\mathcal{B}$ computes the commitment to $\vec{word}'$, and sends the result $C_{word'} := com(\vec{word}')$ to $\mathcal{M}$. Then $\mathcal{B}$ proves (in zero-knowledge) that the content of the commitment has the correct form.
3. $\mathcal{B}$ signs $text$ and $C_{word'}$, and sends the result $sig_{\mathcal{B}} := \mathsf{Sign}(sk_{\mathcal{B}}; text, C_{word'})$ to $\mathcal{M}$.
4. $\mathcal{M}$ verifies the signature and accepts if and only if $true \leftarrow \mathsf{VerSign}(pk_{\mathcal{B}}; sig_{\mathcal{B}})$ holds.
5. $\mathcal{M}$ and $\mathcal{B}$ run the embedding protocol

   $(\mathcal{B} : work^{fing};\ \mathcal{M} : rec_{\mathcal{M}}^{work})$

   $\leftarrow \mathsf{EmbedProt}(\mathcal{B} : state_{\mathcal{B}}^{C_{word'}};\ \mathcal{M} : work, key_{emb}^{s};\ * : C_{word'}, key_{emb}^{p})$

   as described in Section 4.5.5.1.

Identification: On redistribution of a work $work^{red}$ similar to one of his (fingerprinted) original works, i.e., $true \leftarrow \mathsf{simtest}(work^{red}, work)$, $\mathcal{M}$ extracts the codeword $word^{accuse}$ by applying the extracting algorithm

$$word^{accuse} \leftarrow \mathsf{ExtractProt}(word^{red}, key_{emb}, REC_{\mathcal{M}}^{work})$$

as described in Section 4.5.5.1. Now, $\mathcal{M}$ retrieves $sig_{\mathcal{B}}$, $text$ and $C_{word'}$ from the corresponding purchase record of the given work, and accuses this buyer $\mathcal{B}$ of treachery.

Trial: $\mathcal{M}$ presents

$$proof := (sig_{\mathcal{B}}, text, C_{word'}, word^{accuse})$$

to the arbiter $\mathcal{J}$. Then

1. $\mathcal{J}$ verifies the signature $sig_{\mathcal{B}}$ using the public key $pk_{\mathcal{B}}$ of the accused buyer (as $\mathcal{M}$ did in the fingerprinting protocol).
2. $\mathcal{M}$ claims to $\mathcal{J}$ that $word^{accuse}$ has at least $L/2 + L/(16 collsize)$ symbols in common with $word'$ whose symbols are hidden in the commitments $C_{word'}$ (Pfitzmann and Waidner 1997b). If this claim is not true, the buyer can defeat it by opening the commitments. Note that $\mathcal{B}$ does not have to reveal the content of her commitments, since she may prove in zero-knowledge that the claim is not true.

   As one can see, this construction requires a *3-party trial* since the buyer must be present in the trial to deny the charges.

4.5.5.1 Instantiation of Embedding and Extracting

In the following, we will give a brief overview of the embedding and extracting procedures. Later, in Chapter 5, we will present a detailed and explicit construction for these procedures in the context of anonymous fingerprinting schemes (with two party trial), and consider their security. The common input C_{emb} to the embedding protocol is already the commitment to the codeword $\vec{word}' := (sym_1, sym_2, \cdots, sym_L)$ (randomly selected from Σ_0^L by the buyer $\mathcal{B}$.) Thus, in the following, we consider the commitment $C_{\vec{word}'}$.

Encoding $\mathsf{GenWord}()$**:** The main steps of this protocol are as follows:

1. *Generating tracing and embedding halfword*: $\mathcal{M}$ randomly and secretly chooses an index set $I_{\mathcal{B}} \subset \{1, \cdots, L\}$, with $|I_{\mathcal{B}}| = L/2$. It is verified that $|I_{\mathcal{B}}| = L/2$, and the symbols of $\vec{word}'$ at the positions in $I_{\mathcal{B}}$ are selected, and output to $\mathcal{M}$, We denote the resulting tracing halfword with $\vec{halfword\_trace}$ and the remaining embedding halfword with $\vec{halfword\_emb}$. The tracing halfword $\vec{halfword\_trace}$ can be represented by a codeword consisting of the symbols of $\vec{halfword\_trace}$ at the positions in $I_{\mathcal{B}}$ and "?" at the positions not in $I_{\mathcal{B}}$, i.e., something like $halfword\_trace := (?, \cdots, sym\_trace_s, \cdots, ?)$.

2. *Collusion-tolerant encoding*: The symbols sym_s of $\vec{word}'$ are encoded with the corresponding codewords from the basic code $\Gamma_0(n, d)$. The resulting codeword, to be embedded for $\mathcal{B}$, is a codeword *word* from the outer code Γ.

Pfitzmann and Waidner (1997b) propose an efficient 2-party protocol for the above computations.[20]

Embedding word $\mathsf{EmbedWord}()$**:** An explicit construction for this protocol is given in Section 5.3.1, and therefore, we do not repeat it here. Note that the construction not only works for marking schemes represented in original pixel representation but also for any scheme where all marks are disjoint subsets of the data components represented in embedding domain, and where two versions of the data are given for each mark (see also Remark 4.5).

[20]For this, the buyer may compute the commitments C_{word} immediately to the outer codeword *word* and apply an efficient zero-knowledge proof from Pfitzmann and Schunter (1996) to prove that the content of the commitment has the correct form, i.e., the basic codewords of $\Gamma_0(n, d)$ are the inner symbols of *word*. (Note that in our definition of the embedding protocol, the common input C_{emb} (here C_{word}) to the embedding protocol should always have the correct form).

To obtain the half of the symbols, the merchant may blind and permute the commitments C_{word}, prove the correctness of this transformation in zero-knowledge, and let the buyer open the commitments at the desired positions and send back the result.

Extracting word ExtractProt()**:** $\mathcal{M}$ applies the algorithm ExtractProt$(word^{red}, key_{emb}, REC_{\mathcal{M}}^{work})$ for the outer code similar to the extracting algorithm described in Section 4.3.6.4, Algorithm 2. The differences to that algorithm are as follows:

- The search is done – among purchase records $PUR\_REC_{\mathcal{M}}^{work}$ – for the pair $(I_{\mathcal{B}}, \overrightarrow{halfword\_trace})$ which has at least $L/4collsize$ symbols in common with $\overrightarrow{word}^{ext}$. For the choice of parameters see Pfitzmann and Waidner (1997b).
- The output of the extracting algorithm is a codeword $word^{accuse}$ which consists of $halfword\_trace$ at the positions in $I_{\mathcal{B}}$, and of the symbols from $\overrightarrow{word}^{ext}$ at the remaining positions.

4.5.5.2 Security

Pfitzmann and Waidner (1997b) prove that if marking assumption holds for the underlying watermarking scheme, and all the underlying cryptographic primitives are secure, then for the chosen parameters the construction presented in Section 4.5.5 is a secure asymmetric fingerprinting scheme.

4.6 Anonymous Fingerprinting

Anonymous fingerprinting was introduced by Pfitzmann and Waidner (1997a). In the following, we consider a model and a construction framework for anonymous fingerprinting schemes.

4.6.1 Model

The main parties are merchants $\mathcal{M}$, buyers $\mathcal{B}$, registration $\mathcal{RC}$ centers and arbiters $\mathcal{J}$. Before buyers can purchase fingerprinted works, they must register with a registration center $\mathcal{RC}$.

As before, we identify each purchase by a purchase description *text* which contains a description of the underlying work *work*, and the rights the buyer obtains on this work. Figures 4.4 and 4.5 illustrate the main protocols for the purchase and redistribution in an anonymous fingerprinting scheme.

Figure 4.4 The model of anonymous fingerprinting: Purchase

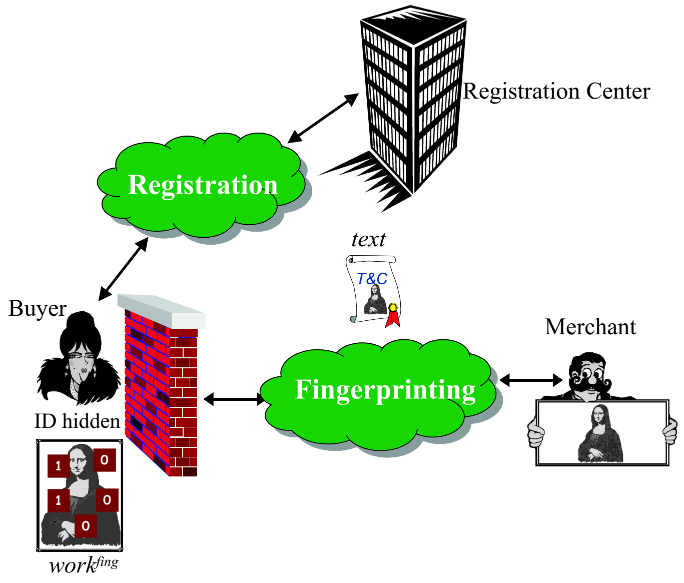
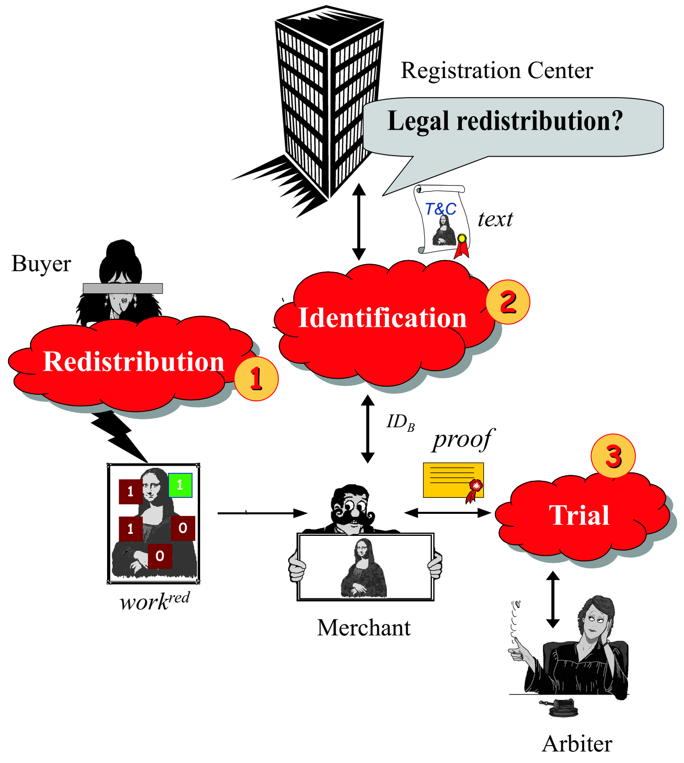


Figure 4.5 The model of anonymous fingerprinting: Redistribution



The main components of an anonymous fingerprinting scheme are as follows:

Setup: This is similar to that of asymmetric scheme described in Section 4.5.1. We denote the corresponding algorithm for generating the global system parameters with $par_{\mathsf{AnoFP}} \leftarrow \mathsf{GenParAnoFP}(par_{sec})$, and the key pair of the registration center $\mathcal{RC}$ with $(sk_{\mathcal{RC}}, pk_{\mathcal{RC}})$.

Registration: This protocol is performed between a buyer $\mathcal{B}$ and a registration center $\mathcal{RC}$. The common inputs are $\mathcal{B}$'s public-key $pk_{\mathcal{B}}$ (representative for $\mathcal{B}$'s digital identity), and $\mathcal{RC}$'s public key $pk_{\mathcal{RC}}$. The secret input of $\mathcal{RC}$ is its secret key $sk_{\mathcal{RC}}$. The outputs are the registration records $reg\_rec_{\mathcal{RC}}$, $reg\_rec_{\mathcal{B}}$ to $\mathcal{RC}$ and $\mathcal{B}$ or *failed* if the protocol fails. We denote the registration protocol as follows:

$$\begin{aligned}&(\mathcal{B} : reg\_rec_{\mathcal{B}};\ \mathcal{RC} : reg\_rec_{\mathcal{RC}})\\ &\qquad \leftarrow \mathsf{Register}(\mathcal{B} : -;\ \mathcal{RC} : sk_{\mathcal{RC}};\ * : pk_{\mathcal{B}}, pk_{\mathcal{RC}}).\end{aligned}$$

Remark 4.12. One may ask why $\mathcal{RC}$ is then needed as the merchants could play this untrusted role themselves. However, buyers will only be anonymous among all people registered at the same registration center, and corresponding groups per merchant would be too small to achieve meaningful anonymity. ○

Fingerprinting: This protocol is performed between the merchant $\mathcal{M}$ and an (anonymous) buyer $\mathcal{B}$. $\mathcal{M}$'s secret input is the work *work*. Further, $\mathcal{M}$ may input the public key $pk_{\mathcal{RC}}$ of the registration center at which $\mathcal{B}$ has been registered. We consider this key as common input. $\mathcal{B}$ inputs her registration record $reg\_rec_{\mathcal{B}}$. $\mathcal{M}$ and $\mathcal{B}$ also input the corresponding text *text*. The output to $\mathcal{M}$ is a purchase record $pur\_rec_{\mathcal{M}}^{work}$ or *failed*. The outputs to $\mathcal{B}$ are a fingerprinted work $work^{fing}$ and a purchase record $pur\_rec_{\mathcal{B}}^{work}$ or *failed*. We denote the fingerprinting protocol as follows:

$$\begin{aligned}&(\mathcal{B} : work^{fing}, pur\_rec_{\mathcal{B}}^{work};\ \mathcal{M} : pur\_rec_{\mathcal{M}}^{work})\\ &\qquad \leftarrow \mathsf{Fing}(\mathcal{B} : reg\_rec_{\mathcal{B}};\ \mathcal{M} : work;\ * : text, pk_{\mathcal{RC}}).\end{aligned}$$

Identification: This protocol is performed between the merchant $\mathcal{M}$ and the registration center $\mathcal{RC}$. $\mathcal{M}$ inputs a redistributed work $work^{red}$ which he wants to trace back to a traitor. $\mathcal{M}$ also inputs all purchase records $PUR\_REC_{\mathcal{M}}^{work}$ for *work*. Note that we require the construction to allow *legal redistribution*. It means that a buyer can purchase "right of distribution" allowing her to redistribute the underlying work, e.g., 5 years after the purchase time. However, it should still not be possible to wrongly accuse an honest buyer (by a cheating merchant or any other party colluding with him against this buyer.) Therefore, $\mathcal{RC}$ first checks whether the redistribution is legal. If negative, $\mathcal{RC}$ inputs its registration records $REG\_REC_{\mathcal{RC}}$. The output to $\mathcal{M}$ is the public-key of a buyer $pk_{\mathcal{B}}$, the text *text* used in the corresponding purchase, and a proof (string) *proof*. We denote the identification protocol as follows:

$$\begin{aligned}&(\mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{RC} : -)\\ &\qquad \leftarrow \mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work};\ \mathcal{RC} : REG\_REC_{\mathcal{RC}}).\end{aligned}$$

If $\mathcal{RC}$ refuses to cooperate[21] then **enforced identification** is performed. In this case, $\mathcal{M}$ should have enough evidence to convince an arbiter $\mathcal{J}$ to enforce the cooperation of $\mathcal{RC}$. Enforced identification is similar to identification except that now, we involve also an arbiter $\mathcal{J}$. $\mathcal{M}$ should obtain the same outputs as in the identification protocol, and $\mathcal{J}$ obtains an indicator $ind \in \{ok, center\_guilty\}$. The output *center_guilty* indicates that $\mathcal{J}$ has noticed misbehavior of the registration center $\mathcal{RC}$. We denote the enforced identification – as a protocol – as follows:

$$\begin{aligned}&(\mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{RC} : -;\ \mathcal{J} : ind)\\ &\quad \leftarrow \mathsf{EnforceIdent}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work};\ \mathcal{RC} : REG\_REC_{\mathcal{RC}};\ \mathcal{J} : -).\end{aligned}$$

Trial: This protocol is performed when the merchant $\mathcal{M}$ wants to convince an arbiter $\mathcal{J}$ that a buyer $\mathcal{B}$ is a traitor. The protocol involves $\mathcal{M}$ and $\mathcal{J}$. $\mathcal{M}$ inputs the identity of the accused buyer $pk_{\mathcal{B}}$, the text *text*, and also the string proof *proof* which he has obtained in the identification protocol. The main output is the arbiter's result $ind \in \{guilty, not\_guilty\}$.[22] If it is *guilty* then she is convinced that $\mathcal{B}$ is a traitor, if it is *not_guilty* then she rejects the accusation. We denote the trial protocol as follows:

$$(\mathcal{M} : -;\ \mathcal{J} : ind) \leftarrow \mathsf{Trial}(\mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{J} : -).$$

Remark 4.13. If the construction requires the buyer to participate in the trial, then we have

$$\begin{aligned}&(\mathcal{B} : -;\ \mathcal{M} : -;\ \mathcal{J} : ind)\\ &\qquad \leftarrow \mathsf{Trial}(\mathcal{B} : reg\_rec_{\mathcal{B}};\ \mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{J} : -)\end{aligned}$$

where $\mathcal{B}$ inputs her registration record $reg\_rec_{\mathcal{B}}$. Note that in this case, $\mathcal{J}$'s result may be *center_guilty* meaning that no resolution could be achieved

[21] This may be indicated by an indicator output to $\mathcal{M}$.

[22] One may consider the output as $(guilty, pk_{\mathcal{B}})$ explicitly indicating the identity $pk_{\mathcal{B}}$ of the traitor.

because $\mathcal{RC}$ has behaved wrongly. Whenever convenient, we use the notation

$$\begin{aligned}&([\mathcal{B} : -];\ \mathcal{M} : -;\ \mathcal{RC} : -;\ \mathcal{J} : ind)\\ &\quad \leftarrow \mathsf{Trial}([\mathcal{B} : reg\_rec_{\mathcal{B}}];\ \mathcal{M} : pk_{\mathcal{B}}, text, proof;\ \mathcal{J} : -)\end{aligned}$$

to consider both possibilities. ∘

4.6.2 Security Requirements

In this section, we describe the main security requirements for an anonymous fingerprinting scheme. The requirements formulated for asymmetric fingerprinting (Section 4.5.2) must be adapted by new requirements, namely, anonymity for the buyer and security for the registration center.

Let $\mathcal{ID}_{\mathcal{B}^*}$ denote any set of *collsize* different valid identities, $\mathcal{A}$ be a probabilistic polynomial-time interactive algorithm, and let $state_{\mathcal{A}}$ denote the state of $\mathcal{A}$ which we will explicitly use in the following. In general, the adversary $\mathcal{A}$ comprises any possible collusions of other parties against the party whose security is being considered.

Security for merchant: An honest merchant $\mathcal{M}$ must be able to identify at least a traitor and win in the corresponding trial for every illegally redistributed copy similar to one of his original works, unless the collusion is larger than the tolerated limit. This should hold even if $\mathcal{RC}$ belongs to the collusion. In this case, $\mathcal{M}$ may require enforced identification, if the identification fails, and the output of this protocol to the arbiter may be *center_guilty*. As discussed in the case of asymmetric fingerprinting (Section 4.5.2), we may immediately formulate the requirement such that merchant is protected from wrong accusations.

More precisely, we call an anonymous fingerprinting scheme secure for the merchant if and only if for all $\mathcal{A}$ the following holds:

- If $\mathcal{M}$ runs the fingerprinting protocol $\mathsf{Fing}(\mathcal{A} : state_{\mathcal{A}};\ \mathcal{M} : work;\ * : text, pk_{\mathcal{RC}})$ with $\mathcal{A}$ at most *collsize* time where $\mathcal{M}$ chooses the work $work \in WORK$, and $\mathcal{A}$ chooses *text*, the valid public keys $pk_{\mathcal{B}^*} \in \mathcal{ID}_{\mathcal{B}^*}$, and the valid public key $pk_{\mathcal{RC}^*}$, and where other transactions ($\mathsf{Register}()$, $\mathsf{Ident}()$, $\mathsf{EnforceIdent}()$) with $\mathcal{A}$ may occur between the executions of $\mathsf{Fing}()$,
- and if $\mathcal{A}$ outputs a work $work^{red}$ such that $true \leftarrow \mathsf{simtest}(work, work^{red})$,
- then the identification $\mathsf{Ident}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work};\ \mathcal{A} : state_{\mathcal{A}})$ or the enforced identification $\mathsf{EnforceIdent}(\mathcal{M} : work^{red}, PUR\_REC_{\mathcal{M}}^{work};\ \mathcal{A} : state_{\mathcal{A}};\ \mathcal{J} : -)$ outputs $(pk_{\mathcal{B}^*}, text, proof)$ to $\mathcal{M}$ with $pk_{\mathcal{B}^*} \in \mathcal{ID}_{\mathcal{B}^*}$ or *center_guilty* to $\mathcal{J}$, except with negligible probability in a security parameter where the probability space is defined over random choices of all the involved algorithms.

and,

- on successful identification, with the result $pk_{\mathcal{B}^*}$, the trial protocol $\mathsf{Trial}(\mathcal{M} : pk_{\mathcal{B}^*}, text, proof;\ \mathcal{J} : -)$ outputs *guilty*, except with negligible probability in a security parameter where the probability space is defined over random choices of all the involved algorithms.

Remark 4.14. As discussed in Remark 4.9, we may relax the requirement on $\mathcal{M}$'s success probability in the (enforced) identification by introducing a certain bound for this probability whereas the success probability in the trial should be overwhelming. Moreover, we may introduce a further bound for the success probability in the enforced identification. ○

Security for buyer: The security for the buyer considers the aspects of *frame proofness* and *unlinkability*. Thus, we call an anonymous fingerprinting scheme secure for the buyer if and only if the following holds:

Frame proofness: An honest buyer $\mathcal{B}$ should not be found guilty by an arbiter $\mathcal{J}$, i.e., the output of the trial protocol for this buyer will not be *guilty*, no matter how large the size of the collusion against this buyer is. In particular, as some redistributions may be legal, a proof of redistribution must be unambiguously linked to a purchase description *text* used during fingerprinting. More precisely, there is no adversary $\mathcal{A}$

- that runs the registration protocol $\mathsf{Register}(\mathcal{B} : -;\ \mathcal{A} : state_{\mathcal{A}};\ * : pk_{\mathcal{B}}, pk_{\mathcal{RC}^*})$ and the fingerprinting protocol $\mathsf{Fing}(\mathcal{B} : reg\_rec_{\mathcal{B}};\ \mathcal{A} : state_{\mathcal{A}}, work;\ * : text, pk_{\mathcal{RC}^*})$ arbitrarily often with an honest buyer $\mathcal{B}$ where $\mathcal{A}$ chooses the valid key $pk_{\mathcal{RC}^*}$, purchase descriptions *text*, and works $work \in WORK$, and where $\mathcal{A}$ may perform other transactions ($\mathsf{EnforceIdent}()$, $\mathsf{Trial}()$) between the executions of $\mathsf{Register}()$ and $\mathsf{Fing}()$,
- and obtains the data of some purchases (e.g., all works $work^{fing}$ for which the corresponding purchase descriptions *text* allow legal distribution),
- and can generate a work $work^{*fing}$, a text $text^*$, and a proof string $proof^*$,
- and then run the trial protocol $\mathsf{Trial}(\mathcal{A} : state_{\mathcal{A}}, proof^*, pk_{\mathcal{B}}, text^*;\ \mathcal{J} : -)$ with an arbiter $\mathcal{J}$ where the output to $\mathcal{J}$ is *guilty*, except with negligible probability in a security parameter where the probability space is defined over the random choices of all the involved algorithms.

Unlinkability: Purchases of honest buyers should not be linkable even by a collusion of all merchants, central parties and other buyers. This means that a collusive adversary learns nothing about the purchase behavior of honest buyers except for facts that can simply be derived from the knowledge of who has registered, and for what number of purchases, and at what time protocols are executed. This property should even hold for the remaining purchases of a buyer (or buyers) if the adversary obtains some works this buyer (or these buyers) bought. Here, we require that the views of the adversary $\mathcal{A}$ from the registration $view^{reg}_{\mathcal{A}}$ and from the fingerprinting $view^{fing}_{\mathcal{A}}$ are unlinkable. We consider the unlinkability of these views in two cases, namely, with and without legal redistribution, and for the same *and* for two different buyers.[23] We start with the case, where *text* does not allow legal redistribution, and where there are two different buyers.

More precisely, we call an anonymous fingerprinting scheme unlinkable for two different buyers, if and only if the following holds: For all adversary $\mathcal{A}$,

- if $\mathcal{A}$ carries out with two different (honest) buyers, $pk_{\mathcal{B},0}$ and $pk_{\mathcal{B},1}$, two registrations

$$\mathsf{Register}(\mathcal{B} : sk_{\mathcal{B},0};\ \mathcal{A} : state_{\mathcal{A}};\ * : pk_{\mathcal{B},0}, pk_{\mathcal{RC}}),$$
$$\mathsf{Register}(\mathcal{B} : sk_{\mathcal{B},1};\ \mathcal{A} : state_{\mathcal{A}};\ * : pk_{\mathcal{B},1}, pk_{\mathcal{RC}^*})$$

  and then carries out the corresponding fingerprintings

$$\mathsf{Fing}(\mathcal{B} : reg\_rec_{\mathcal{B},b};\ \mathcal{A} : work_b, state_{\mathcal{A}};\ * : text_b, pk_{\mathcal{RC}^*}),$$
$$\mathsf{Fing}(\mathcal{B} : reg\_rec_{\mathcal{B},\bar{b}};\ \mathcal{A} : work_{\bar{b}}, state_{\mathcal{A}};\ * : text_{\bar{b}}, pk_{\mathcal{RC}^*})$$

  with these buyers in *random order* according to a secretly, randomly and uniformly chosen bit $b \in_{\mathcal{R}} \{0,1\}$, where $\mathcal{A}$ chooses the valid key $pk_{\mathcal{RC}^*}$, purchase descriptions *text*, and works $work \in WORK$, and where $\mathcal{A}$ may perform other transactions ($\mathsf{EnforceIdent}()$, $\mathsf{Trial}()$) between the executions of these registrations and the corresponding fingerprintings,
- then $\mathcal{A}$ cannot guess the bit b with probability significantly better than 1/2 (i.e., which of adversary's views $view^{reg}_{\mathcal{A},0}$, $view^{reg}_{\mathcal{A},1}$ in the two registrations corresponds to its views $view^{fing}_{\mathcal{A},b}$, $view^{fing}_{\mathcal{A},\bar{b}}$ in the corresponding fingerprintings) where the probability space is defined over the random choices of all the involved algorithms.

Next, we consider the transactions for two different buyers, but in the case, where *text* allows *legal distribution*. In this case, $\mathcal{RC}$ must be trusted regarding the recovery of buyer's anonymity for that purchase after the distribution, and thus, the scheme can only offer *weak anonymity* (see also Section 4.2). More precisely, we call an anonymous fingerprinting scheme with legal distribution unlinkable for two different buyers, if and only if the following holds: For all adversary $\mathcal{A}$

[23]Note that unlinkability for two different buyers does not necessarily imply unlinkability for the same buyer: Consider a fictive system where transactions of buyers are numbered (e.g., incrementally) in registration, and the buyers have to reveal this number in fingerprinting. For different buyers this system is unlinkable since the transactions of these buyers have the same number. However, the transactions of a single buyer are linkable.

- if two different (honest) buyers, $pk_{\mathcal{B},0}$ and $pk_{\mathcal{B},1}$, carry out two registrations

$$\begin{aligned}&\mathsf{Register}(\mathcal{B} : sk_{\mathcal{B},0};\ \mathcal{RC} : sk_{\mathcal{RC}};\ * : pk_{\mathcal{B},0}, pk_{\mathcal{RC}}),\\ &\mathsf{Register}(\mathcal{B} : sk_{\mathcal{B},1};\ \mathcal{RC} : sk_{\mathcal{RC}};\ * : pk_{\mathcal{B},1}, pk_{\mathcal{RC}})\end{aligned}$$

  and then carry out with $\mathcal{A}$ the corresponding fingerprintings

$$\begin{aligned}&\mathsf{Fing}(\mathcal{B} : reg\_rec_{\mathcal{B},b};\ \mathcal{A} : work_b, state_{\mathcal{A}};\ * : text_b, pk_{\mathcal{RC}}),\\ &\mathsf{Fing}(\mathcal{B} : reg\_rec_{\mathcal{B},\bar{b}};\ \mathcal{A} : work_{\bar{b}}, state_{\mathcal{A}};\ * : text_{\bar{b}}, pk_{\mathcal{RC}})\end{aligned}$$

  in *random order* according to a secretly, randomly and uniformly chosen bit $b \in_{\mathcal{R}} \{0,1\}$, where $\mathcal{A}$ chooses purchase descriptions *text*, and works $work \in WORK$, and where $\mathcal{A}$ may perform other transactions (Register(), EnforceIdent(), Ident(), Trial()) between the executions of these registrations and the corresponding fingerprintings,
- and then $\mathcal{A}$ obtains the outputs $work_b^{fing}$ and $work_{\bar{b}}^{fing}$ of the two fingerprinting protocols,
- then $\mathcal{A}$ cannot guess the bit b with probability significantly better than 1/2, where the probability space is defined over the random choices of all the involved algorithms.

Remark 4.15. Similarly, we can formulate the above requirements for the same buyer, except that the same key pair $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$ is used as input to the corresponding protocols (i.e., the key generation algorithm for the buyer is executed only once.)

○

Security for registration center: An honest registration center will never be found guilty, i.e., the output of the enforced identification to the arbiter $\mathcal{J}$ will not be *center_guilty*.

More precisely, we call an anonymous fingerprinting scheme secure for the registration center if and only if the following holds: There is no adversary $\mathcal{A}$

- that runs $\mathsf{Register}(\mathcal{A} : state_{\mathcal{A}};\ \mathcal{RC} : sk_{\mathcal{RC}};\ * : pk_{\mathcal{B}^*}, pk_{\mathcal{RC}})$ with $\mathcal{RC}$ polynomially many times where $\mathcal{A}$ chooses the valid identities $pk_{\mathcal{B}^*}$, and where other transactions (Ident(), Enforce(), Trial()) with $\mathcal{A}$ may occur between the executions of Register(),
- and can generate a work $work^{red}$ and the corresponding record $pur\_rec_{\mathcal{A}}^{work}$,
- and then run the enforced identification $\mathsf{Enforce}(\mathcal{A} : state_{\mathcal{A}}, work^{red};\ \mathcal{RC} : REG\_REC_{\mathcal{RC}};\ \mathcal{J} : -)$ with an arbiter $\mathcal{J}$ where the output to $\mathcal{J}$ is *center_guilty*, except with negligible probability in a security parameter where the probability space is defined over random choices of all the involved algorithms.

Remark 4.16. One may additionally consider the requirements **completeness** and **no jamming by registration center**: The former requirement means that the registration and fingerprinting succeed if all parties behave correctly. The latter one means that even if $\mathcal{RC}$ is dishonest, $\mathcal{B}$ will convince any honest merchant in fingerprinting, if she has accepted the corresponding registration. ○

4.6.3 Construction Framework

The main task during the fingerprinting protocol is to embed the secret identifying information $\vec{emb}$ into the underlying work *work* where only the buyer obtains the fingerprinted work $work^{fing}$ at protocol completion. Similar to Section 4.5.3, we require the construction to consider the following issues:

- *Verification*: The merchant must be provided with mechanisms for (i) verifying that $\vec{emb}$ has the correct form, (ii) establishing a relation between $\vec{emb}$ and the public parameters of the registration center (e.g., its public key), and (iii) verifying this relation. All these procedures should not enable the merchant to (i) identify buyers, (ii) link the purchases of buyers, and (iii) obtain any useful information about the values $\vec{emb}$.
- *Embedding*: There must be a mechanism to embed $\vec{emb}$ into *work* such that the merchant is convinced that $\vec{emb}$ is correctly embedded without obtaining any useful information about $\vec{emb}$, and $\mathcal{B}$ obtains a fingerprinted version of the desired work without getting any information enabling her to remove the embedded fingerprint (see Section 4.5.4)
- *Extracting*: The merchant must be provided with a mechanism to extract the value $\vec{emb}$ from a redistributed work (which is similar to one of his original works.)

  Note that for an anonymous fingerprinting scheme the merchant needs to extract the identifying information of one traitor as a whole. This is more involved than, e.g., the constructions of non-anonymous asymmetric schemes described in Section 4.5. The reason is that there the identifying information is not directly extracted from a redistributed work, but rather in combination with other information (or through interaction with the accused buyer).

We first review the construction framework proposed by Pfitzmann and Waidner (1997c). This construction is modular, and is based on certificates. It presents solutions for the verification, embedding and extracting components of the fingerprinting scheme. However, as mentioned above, these solutions are partially explicit. Understanding this construction framework will be helpful in later sections to precisely see the achievements of this part of this thesis.

The construction of Pfitzmann and Waidner (1997c) is as follows:

Registration: The buyer $\mathcal{B}$ chooses a **pseudonym** in form of a key pair $(sk_{pseud_{\mathcal{B}}}, pk_{pseud_{\mathcal{B}}})$ of a signature scheme, and signs under her real digital identity (here $pk_{\mathcal{B}}$) that she will be responsible for this pseudonym. She sends the result $sig_{pseud_{\mathcal{B}}} := \mathsf{Sign}(sk_{\mathcal{B}}; pk_{pseud_{\mathcal{B}}})$ to the registration center $\mathcal{RC}$, and obtains a certificate $cert_{\mathcal{B}} := \mathsf{Sign}(sk_{\mathcal{RC}}; pk_{pseud_{\mathcal{B}}})$ from $\mathcal{RC}$. With this signature, $\mathcal{RC}$ declares that it knows the identity $pk_{\mathcal{B}}$ of the buyer who has chosen this pseudonym.

Fingerprinting: The following steps are performed:
Verification:

1. The anonymous buyer $\mathcal{B}$ secretly computes a signature $sig_{text} := \mathsf{Sign}(sk_{pseud_{\mathcal{B}}}; text)$ on the text $text$. The value to be embedded is

$$\vec{emb} := (text, sig_{text}, pk_{pseud_{\mathcal{B}}}, cert_{\mathcal{B}}).$$

2. $\mathcal{B}$ hides $\vec{emb}$ in a commitment $C_{emb} := com(\vec{emb})$, and sends it to $\mathcal{M}$.
3. $\mathcal{B}$ proves in zero-knowledge the validity of the signature sig_{text} with respect to $pk_{pseud_{\mathcal{B}}}$, and the validity of the certificate $cert_{\mathcal{B}}$ with respect to $\mathcal{RC}$'s public key $pk_{\mathcal{RC}}$.

Embedding: $\mathcal{B}$ and $\mathcal{M}$ engage in an embedding protocol $\mathsf{EmbedProt}()$ as defined in Section 4.5.4.

Identification: $\mathcal{M}$ finds a redistributed version $work^{red}$ of his work for which holds $true \leftarrow \mathsf{simtest}(work^{red}, work)$. $\mathcal{M}$ applies the extracting algorithm as for the asymmetric scheme (Section 4.5.5) where he extracts $\vec{emb}$. Then

1. $\mathcal{M}$ retrieves a proof string $proof_{\mathcal{RC}} := (text, sig_{text}, pk_{pseud_{\mathcal{B}}})$. Note that $\mathcal{M}$ verifies sig_{text} using $pk_{pseud_{\mathcal{B}}}$. The string $proof_{\mathcal{RC}}$ proves that the owner of the pseudonym $pk_{pseud_{\mathcal{B}}}$ has redistributed the work related to the text $text$.
2. $\mathcal{M}$ sends $proof_{\mathcal{RC}}$ to the registration center $\mathcal{RC}$, and asks $\mathcal{RC}$ to reveal the real identity of the owner of $pk_{pseud_{\mathcal{B}}}$. If $\mathcal{RC}$ refuses, its cooperation is enforced. For this, $\mathcal{M}$ shows $(proof_{\mathcal{RC}}, cert_{\mathcal{B}})$ to an arbiter $\mathcal{J}$, where $cert_{\mathcal{B}}$ proves that $\mathcal{RC}$ must know the corresponding identity. Thus, in enforced identification, $\mathcal{RC}$ either has to identify or will be found guilty.

   In any case, $\mathcal{RC}$ has to send the buyer's signature $sig_{pseud_{\mathcal{B}}}$ showing that she is responsible for this pseudonym $pk_{pseud_{\mathcal{B}}}$. Now, the final proof is $proof := (proof_{\mathcal{RC}}, sig_{pseud_{\mathcal{B}}})$. Note that $\mathcal{M}$ verifies all the values before making an accusation.

Trial: The arbiter $\mathcal{J}$ verifies that:

1. $sig_{pseud_{\mathcal{B}}}$ is a valid signature of the accused buyer, i.e., whether $true \leftarrow \mathsf{VerSign}(pk_{\mathcal{B}}; sig_{pseud_{\mathcal{B}}})$ holds.
2. sig_{text} is a valid signature on *text* corresponding to this pseudonym, i.e., whether $true \leftarrow \mathsf{VerSign}(pk_{pseud_{\mathcal{B}}}; sig_{text})$ holds.

If these verifications are positive then $\mathcal{J}$ declares $\mathcal{B}$ for guilty, otherwise she rejects the accusation.

Discussion

Pfitzmann and Waidner (1997a) do not give an explicit construction for the verification phase of the fingerprinting explained above – more precisely, for Step 3. They only employ general results such as "every NP language has an zero-knowledge proof system" (see, e.g., Goldreich (2001b)). Furthermore, their framework of the embedding protocol is partially explicit.

In the following section, we present an explicit and fairly efficient solution to the verification part by exploiting ideas from **electronic coin systems**. Based on the framework of Pfitzmann and Waidner (1997a) and the explicit construction for the verification part, we will give a concrete construction for the embedding part in Chapter 5.

4.6.3.1 Security

Pfitzmann and Waidner (1997a) show that if the underlying primitives (embedding protocol, signatures, commitments, etc.) are secure then the construction framework yields a provably secure anonymous fingerprinting scheme.

4.7 Coin-Based Anonymous Fingerprinting

4.7.1 Motivation

Based on ideas from **coin systems**, we present the first explicit and fairly efficient protocol for the construction framework that we reviewed in Section 4.6.3. Before going into the details of the construction, we first give an overview of the most important aspects of coin systems. Then we explain how to use the basic ideas of coin systems to construct an anonymous fingerprinting scheme, discuss the problems that arise and how to solve them, and give an overview of the ideas for achieving direct non-repudiation.

4.7.2 Anonymous Electronic Cash Systems

4.7.2.1 Model

The main parties in an electronic payment system are a **bank**, a **payer** and a **recipient**. The system consists of several phases or protocols in which different parties may interact. In **electronic cash systems** these phases are **system setup** where system parameters are generated, **registration** where a user opens an account at the bank, **withdrawal** where a user withdraws electronic coins at the bank, **payment** where a user (payer) spends coins to a recipient, and **deposit** where a recipient deposits the obtained coins to her account at the bank. In an **online** system payment and deposit are combined while in an **offline** system they are separate transactions.

4.7.2.2 Main Security Requirements

The main security properties in payment systems are **security against fraud (integrity)** and **privacy (confidentiality)**. Both properties should be fulfilled in the sense of multi-party security, i.e., no party should be forced to trust the others a priori. Privacy means that the electronic payment system should provide at least the privacy offered by traditional cash systems, i.e., payments of small amounts can be performed anonymously, so that no profiles can be collected (at least from the payment data) on what kind of items people purchase in daily life.

Electronic payment systems preserving privacy are called **anonymous payment systems**. There exist quite a lot of proposals for **anonymous cash** systems in the literature, e.g., Franklin and Yung (1993), Chaum (1983), Chaum (1985), Chaum (1989), Chaum, Fiat, and Naor (1990), Brands (1993).

Anonymous payment systems offer different types of anonymity such as

- which of the involved parties is anonymous (payer, recipient or both),
- in which transactions (withdrawal, payment) which of the involved parties (payer, recipient or both) is/are anonymous and
- **unlinkability**, i.e., whether several transactions are anonymous relative to each other.

Most anonymous cash systems offer only *payer anonymity* and this only in the *payment*, but with *unlinkability*.

The best-known systems in this class apply *blind signatures* to realize the anonymity property (see Section 3.4.4). These cash systems are also known as **coin systems**, where a digital coin is a message (representing a monetary value) blindly signed by the bank (i.e., the bank cannot see the content of what it is signing). The unlinkability in a coin system means that even a collusion of the bank and the recipient cannot link the withdrawal and the payment of a payer. In other words, the views of the bank in withdrawal and the view of the recipient in payment are independent where this property may hold unconditionally or computationally.

An important security requirement on coin systems is **double spender identification**. This is a mechanism that allows the bank to link the withdrawal with the payment of a cheating payer who spent a coin more than once without authorization. Obviously, this is easy in an **online coin system** where the bank is online in all payments, and immediately checks in its database whether the corresponding coin has already been spent. However, in an **offline coin system** this check can only be done after the fact, and thus, the payment scheme has to provide the bank with a doublespender identification mechanism.

For our fingerprinting scheme, we require similar mechanisms as in some offline coin systems. Thus, as the subject of the next section, we explain how typical offline coin systems work.

4.7.2.3 Typical Offline Coin System

We briefly explain the typical scenario for an offline coin system with double spender identification. In the registration and withdrawal phase, the payer and the bank perform the following transactions:

1. A user (payer) $\mathcal{U}$ opens an account at the bank by identifying herself to the bank. Here, we may consider user's public key $pk_{\mathcal{U}}$ representative for her digital identity.

2. The bank (here the signer $\mathcal{S}$) blindly signs a secret value called **identity proof** i of the user. The resulting signature of the bank output to the user is attached to a monetary value, and is interpreted as the underlying coin. We denote the coin with $coin$.

   The type of coin scheme we are going to use requires the user $\mathcal{U}$ to choose i secretly and randomly, apply a one-way function $f()$ to it, and send the result $m := f(i)$ blinded to the bank. m can be interpreted as an account number. The coin will then have the form $coin := (m', \tau')$ where m', is the unblinded version of m and $\tau := \mathsf{Sign}(sk_{\mathcal{S}}; m')$ denotes the signature of the bank on m'.

3. The user has to securely commit herself to the identity proof i by a signature (under her real identity) $sig_{\mathcal{U}}$ and a proof of knowledge of i to the signer. The signature indicates that the user is responsible for the signed content.

The payment protocol consists of a challenge and responses for each spent coin. The payment and deposit are as follows:

1. The user $\mathcal{U}$ first gives the coin *coin* to the recipient. The recipient verifies its validity, i.e., verifies whether it is a valid signature of the bank, and accepts if and only if this verification is true.
2. The recipient sends a challenge C to the payer who answers with a response $r(i)$ which is a function of the identity proof i.
3. The recipient verifies this response by a verification function $\mathsf{Verify}(r)$, and accepts the coin if and only if the output is *true*. Then the recipient stores $(coin, C, r)$.
4. Later, the recipient reveals this record to the bank at deposit.
5. The bank checks for doublespending in its database, and if negative, it deposits the corresponding amount to the payee's account.

For such a construction, the *doublespender identification* is of the following form:

1. If the payer spends a coin *coin* more than once, then there will be a second challenge and response (C', r') for this coin.[24] At deposit, the bank checks its database, and recognizes that this coin is spent more than once. Note that r should perfectly hide the identity proof i so that the recipient cannot extract i from one response. The construction of the coin system enables the bank to compute the identity proof i using both challenge and responses, i.e., $i \leftarrow g(C, C', r, r')$ where $g()$ denotes the corresponding function fixing the relation between these parameters.
2. Having computed i, the bank can compute the account number m as $f(i)$, and thus, link i to the user's signature $sig_{\mathcal{U}}$ obtained from the user at the withdrawal. Given the one-wayness of $f()$ and the fact that i is perfectly hidden in a single payment, this signature serves as a proof for payer's doublespending.

4.7.2.4 Brands' Offline Coin System

Brands (1994) extended the blind signature of Chaum-Pederson (CP) (Section 3.4.4.1), and used the resulting scheme to construct an anonymous offline payment scheme. It is the best-known scheme since it is very efficient. In the following, we will explain the main aspects of Brands' system where our focus is on the withdrawal part.

Key generation: In addition to the parameters for CP signature, two new random generators g_1 and g_2 different from g are selected.

[24]Note that the domain from which the challenge is chosen is sufficiently (exponentially) large, and since an honest recipient chooses his challenges randomly, the probability that the same challenge is selected for both coins is exponentially small.

Encoding identity proof into a message: To efficiently realize double-spender identification, a secret value, called **identity proof** i, only known to the user, is securely encoded into a coin, and recovered only if the user doublespends this coin. Therefore, in Brands (1994) the message m to be signed must have a special form, namely, $m := g_1^i g_2$ (which represents the function $f()$.) This kind of blind signature that hides a specific structure, such as identity, is called **restrictive blind signature**.

Now, assume that the blinding can *only* be done as specified by the CP blind signing protocol, then the result of the blinding will be $m' \equiv g_1^{is} g_2^s g^t$. The construction of the corresponding payment protocol in Brands' system is such that the payer can only pass the corresponding verifications if she knows a pair of values (i_1, i_2) such that the unblinded version m' of m is $m' \equiv g_1^{i_1} g_2^{i_2}$. The pair (i_1, i_2) is called a *representation* of m' with respect to the generators g_1 and g_2. Now, one can show that the two representations of m', i.e., $(i_1, i_2, 0)$ and (is, s, t) are the same (Chaum and Pedersen 1993). Thus, the identity proof as $i :\equiv i_1/i_2$ remains intact, and as mentioned above, doublespending leads to recovery of i. For completeness, this signature is illustrated in Figure 4.6.

Figure 4.6 The restrictive blind signature of Brands

$$
\begin{array}{rcl}
\mathcal{R} & & \mathcal{S} \\
 & & z \leftarrow m^x \\
 & & w \in_R \mathbb{Z}_q \\
 & & a \leftarrow g^w \bmod p \\
s \in_R \mathbb{Z}_q^*,\ u,\ v \in_R \mathbb{Z}_q & \xleftarrow{\quad z, a, b \quad} & b \leftarrow m^w \bmod p \\
m' \leftarrow m^s,\ z' \leftarrow z^s & & \\
x_1, x_2 \in_R \mathbb{Z}_q & & \\
pk_{coin} \leftarrow g_1^{x_1} g_2^{x_2} \bmod p & & \\
a' \leftarrow a^u g^v,\ b' \leftarrow b^{su} m'^v & & \\
c' \leftarrow hash(m', z', a', b', pk_{coin}) \bmod q & & \\
c \leftarrow c' u^{-1} \bmod q & \xrightarrow{\quad c \quad} & \\
g^r \overset{?}{\equiv} ah^c \bmod p,\ m^r \overset{?}{\equiv} bz^c \bmod p & \xleftarrow{\quad r \quad} & r \leftarrow cx + w \bmod q \\
r' \leftarrow ru + v \bmod q & & \\
\sigma' \leftarrow (z', a', b', c', r') & & \\
coin' \leftarrow (m', pk_{coin}, \sigma') & &
\end{array}
$$

We denote the signature protocol of Brands (Br) with

$$(\mathcal{R} : (m', pk_{coin}), \sigma';\ \mathcal{S} : -) \leftarrow \mathsf{BlindSignBr}(\mathcal{R} : -;\ \mathcal{S} : sk_{\mathcal{S}};\ * : m).$$

The coin $coin'$ contains a CP signature σ' on m' and pk_{coin} called **public coin key**. This value is fixed for the corresponding coin by being hashed with other values.

The verification of the coin is denoted by

$$ind \leftarrow \mathsf{VerBlindSignBr}(pk_{\mathcal{S}}; (m', pk_{coin}, \sigma'))$$

where $ind \in \{accept, reject\}$.

In the payment, the payer sends $coin'$ to the recipient together with the values r_1, r_2 as responses to the challenge C sent by the recipient. These responses are computed as $r_1 := Ci_1 + x_1, r_2 := Ci_2 + x_2$ where (x_1, x_2) are the secret values chosen by the payer, and (i_1, i_2) is a representation of m' – more precisely, the payer gives a proof of knowledge of a representation. Using a verification equation the recipient can verify these responses. Thus, each payment reveals two linear equations on the secrets (i_1, i_2, x_1, x_2).

In the case of doublespending, the recipient will have four linear equations and can extract the identity proof as $i := (r_1 - r_1')/(r_2 - r_2')$ where C', r_1', r_2' are the challenge and valid responses of the second payment.

Now, one has to justify why it is hard for an adversary to find any other form of blinding such that the required verifications are passed but the representation (i_1, i_2) of m' is different than (is, s) such that the identity proof cannot be extracted in case of doublespending. Here, we require an assumption called **restrictiveness assumption**. This assumption was mentioned in Brands (1993) and Brands (1994) but fairly informal. Here, we will give a formal and general definition for it.

Recall from Section 3.7.2, by $repr(m, B, I)$ we denote that a vector $I = (i_1, ..., i_n) \in \mathbb{Z}_q^n$ is the representation of a message $m \in G$ to the (generator) base $B = (g_1, ..., g_n)$, i.e.,

$$repr(m, B, I) :\Longleftrightarrow \; m = g_1^{i_1} ... g_n^{i_m}.$$

We now consider an adversary $\mathcal{A}$ who executes the blind signing protocol with an honest Brands' signer $\mathcal{S}$ polynomially many times. $\mathcal{A}$ outputs a representation I_j of each input message m_j. Note that we make the assumption with actual outputs in places where the real protocols only have proofs of knowledge. We let the output of the blind signature protocol to adversary be its state $state_{\mathcal{A}}^{j}$ (instead of the pair (m', σ')). At the end, $\mathcal{A}$ outputs valid signatures, again with representations of the signed messages m'. Finally, for any vector (representation) $I \neq 0$ let $\overline{I}$ denote the line it generates (going through the points $(0, 0)$ and I), i.e., the set of its scalar multiples.

The assumption is that all the output representations $I'_{j'}$ are scalar *multiples of the input representations* I_j except with negligible probability in the security parameter k.

Assumption 1 (Restrictiveness Assumption)

$\forall Q \in \mathsf{poly}(k);\ \forall \mathcal{A} \in \mathcal{UPTM};$
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0;$
$\mathbf{Prob}[\neg\big(\{\overline{I'_1}, \ldots, \overline{I'_{l'}}\} \subseteq \{\overline{I_1}, \ldots, \overline{I_l}\}\big) \wedge \forall j = 1, \ldots, l :$
$\mathit{repr}(m_j, B, I_j) \wedge \forall j' = 1, \ldots, l' : \mathit{repr}(m'_{j'}, B, I'_{j'}) \wedge \mathit{true} \leftarrow$
$\mathsf{VerBlindSignBr}(pk_{\mathcal{S}}; (m'_{j'}, pk_{coin,j'}, \sigma'_{j'})) \wedge$
all pairs $(m'_{j'}, pk_{coin,j'})$ are different ::

```
    $par_{\mathsf{SignBr}} \leftarrow \mathsf{GenParSignBr}(par_{sec});$
    $l \leftarrow Q(k); (sk_{\mathcal{S}}, pk_{\mathcal{S}}) \leftarrow \mathsf{GenKeySign}(par_{\mathsf{SignBr}}[\mathsf{GenSignKey}]);$
    $rand \xleftarrow{\mathcal{R}} \mathcal{U};$
    $(state^1_{\mathcal{A}}, m_1, I_1) \leftarrow \mathcal{A}(k, rand, pk_{\mathcal{S}});$
    $(\mathcal{A} : state^2_{\mathcal{A}};\ \mathcal{S} : -) \leftarrow \mathsf{BlindSignBr}(\mathcal{A} : state^1_{\mathcal{A}};\ \mathcal{S} : sk_{\mathcal{S}};\ * : m_1);$
    $(state^3_{\mathcal{A}}, m_2, I_2) \leftarrow \mathcal{A}(state^2_{\mathcal{A}});$
    $(\mathcal{A} : state^4_{\mathcal{A}};\ \mathcal{S} : -) \leftarrow \mathsf{BlindSignBr}(\mathcal{A} : state^3_{\mathcal{A}};\ \mathcal{S} : sk_{\mathcal{S}};\ * : m_2);$
    $\vdots$
    $(\mathcal{A} : state^l_{\mathcal{A}};\ \mathcal{S} : -) \leftarrow \mathsf{BlindSignBr}(\mathcal{A} : state^{l-1}_{\mathcal{A}};\ \mathcal{S} : sk_{\mathcal{S}};\ * : m_{l-1});$
    $(state^{l+1}_{\mathcal{A}}, m_l, I_l) \leftarrow \mathcal{A}(state^l_{\mathcal{A}});$
    $(\mathcal{A} : state^{l+2}_{\mathcal{A}};\ \mathcal{S} : -) \leftarrow \mathsf{BlindSignBr}(\mathcal{A} : state^{l+1}_{\mathcal{A}};\ \mathcal{S} : sk_{\mathcal{S}};\ * : m_l);$
    $\big(l', ((m'_1, pk_{coin,1}, \sigma'_1), \ldots, (m'_{l'}, pk_{coin,l'}, \sigma'_{l'})), (I'_1, \ldots, I'_{l'})\big) \leftarrow$
    $\mathcal{A}(state^{l+2}_{\mathcal{A}})$
  $] \le 1/k^{d_1}.$
```

Here, par_{SignBr} denotes the global parameters (group, generators, etc.) generated by the generation algorithm $\mathsf{GenParSignBr}()$, and $par_{\mathsf{SignBr}}[\mathsf{GenSignKey}]$ denotes the parameters required for the key generation algorithm of the signer. To obtain the original Brands' signature, we set $n = 2$.

In the next section, we will explain, how we can apply the ideas of such a coin scheme to construct and anonymous fingerprinting scheme.

4.7.3 General Ideas

The basic idea for using an offline coin system with double-spender identification to construct an anonymous fingerprinting scheme is as follows: Registration corresponds to withdrawing a coin. Note that "coins" only serve as a cryptographic primitive and have no monetary value. During fingerprinting, the coin is given to the merchant, and in principle a first payment with the coin is made.[25] So far, the unlinkability of the cash system should guarantee that the views of the registration center and the merchant are

[25] Actually the protocol is simpler, more like "zero-spendable" coins where the coin as such can be shown but any response to a challenge leads to identification. For intuitiveness, we nevertheless still call this response "second payment" in the informal part.

unlinkable. Then a second payment with the same coin is started. Now, instead of giving the buyer's response to the merchant, it is embedded in the work. After a redistribution, the merchant can extract the second response from the underlying work, and carry out double-spender identification.

However, we are now concerned with new problems, since there are important differences to the cash systems. We require

1. an efficient mechanism for the verification phase of the fingerprinting. In particular, a mechanism for establishing an unambiguous link to the purchase description *text* (as also required to allow legal redistribution). Recall that in cash systems, double-spender identification has no such property: The merchant simply obtains one fixed secret *identity proof* i, independent of which coins were double spent and how often.

2. an efficient mechanism to secretly and verifiably embed the second payment response into the underlying work.

To solve the first problem, the first idea is to sign *text* with a secret key whose corresponding public key pk_{text}, the *coin key*, is included in the coin. However, the registration center, as the signer of the coins, can forge coins even in such a way that they can be linked to a certain withdrawal (where the buyer may have signed the withdrawal data). Hence, the real problem is how to show that the particular coin with pk_{text} is in fact one that the accused buyer has withdrawn. Pfitzmann and Sadeghi (1999) propose an explicit solution to this problem, however, their solution requires the buyer to repudiate an accusation with a wrong coin: For this, she should present a different coin, and the blinding elements that link it to the specific withdrawal from which this coin is supposed to come. Thus, a 3-party trial is required. Although the results in Pfitzmann and Sadeghi (1999) build a part of the research done within this thesis, we do not go through its details. Instead, we will present a more enhanced solution which is also coin-based, but, in contrast to the mentioned solution, requires only a 2-party trial, i.e., the merchant is provided with a direct proof that does not involve the buyer (*direct non-repudiation*).

Further, in Chapter 5, we will present an explicit solution to the second problem, i.e., an explicit construction for secure embedding.

4.7.4 Ideas for Achieving Direct Non-Repudiation

In this section, we give an informal overview of the construction with direct non-repudiation, i.e., where the merchant can convince an arbiter without participation of the accused buyer. As described in Section 4.7.3, we want to fix the actual purchase description *text* by signing it with respect to a key pk_{text} contained in the coin, and it remains to link this key unforgeably to a particular buyer after a redistribution.

The basic idea is to *commit* to this coin-key pk_{text} (unconditionally binding) during the registration, and such that the identity proof i is the secret key needed for decryption, i.e., $pk_{text} := \mathsf{Decrypt}(i; enc)$ where enc represents this commitment. The buyer must sign this encryption enc under his real identity so that he is bound to it. Hence, once the merchant learns i due to a redistribution, it is possible to decrypt enc, and verify which coin key pk_{text} the buyer planned to use. Note that the buyer is not needed in this step; this is essential for the direct *non-repudiation*.

Each i is only used for one coin so that the link between the particular coin and the corresponding encryption enc will be unique.

The next question is how to force the buyer to encrypt the same pk_{text} in enc as he uses in the coin – clearly, if he can encrypt another value, his real coin will later not be attributed to him. Hence, we need a kind of verifiable encryption. However, at this point there is nothing to verify the encryption against since pk_{text} is deep inside the perfectly blinded coin.

Here, we can apply the ideas from *coin tracing*.[26] In particular, we can apply the ideas from Frankel, Tsiounis, and Yung (1996) for Brands' cash scheme, where a similar problem exists with an encryption enc^* for a trusted third party. The solution is to provide an additional specific encoding $M := f'(pk_{text})$ (with f' as the one-way encoding function) of pk_{text} whose content is invariant under blinding. During registration (withdrawal), the buyer proves in zero-knowledge that enc and M have the same content. The registration center $\mathcal{RC}$ then blindly signs M, and the buyer transforms it to M'. This is Brands' blind signature that has the form (M', pk_{text}, σ') where $\sigma' := \mathsf{Sign}(sk_{\mathcal{RC}}; (M', pk_{text}))$ (see Section 4.7.2.3).

Later, in fingerprinting (payment), the merchant sees the real pk_{text}, used in the coin, in clear. The buyer then opens the blinded encoding M', which has the same content as M, and the merchant verifies that this content is really pk_{text}. Overall, this implies that also enc contained the correct pk_{text}.

However, there are two main differences to the ideas in Frankel, Tsiounis, and Yung (1996): Firstly, we use the identity proof as a key instead of a trusted third party's key as common in coin tracing. Secondly, in Frankel, Tsiounis, and Yung (1996), the coin and M are blindly signed in two different signatures. If we did this, traitors could successfully attack the scheme by combining wrong pairs of coins and M's. Hence, we need a combined blind signature on the pair (M, m), where the pair can be uniquely decomposed both in the blinded and unblinded form. Thus, we need another modification to that scheme. More concretely, in our scheme, this combination is the product N of the encoding of the identity proof $m = f(i)$ (in the coin) and the encoding $M = f'(pk_{text})$ of pk_{text}, i.e., $N := mM$. We denote the

[26]This is a mechanism which allows a bank, in cooperation with a trustee, to revoke the anonymity of a user, and trace the corresponding coins. Such measures are proposed to deter the misuse of anonymous payment systems for criminal purposes.

resulting blind signature on this combination by $coin' := (N', pk_{text}, \tau')$ where $\tau' := \mathsf{Sign}(sk_{\mathcal{RC}}; (N', pk_{text}))$, and N' has the property that it is the product of the blinded versions m', N' of the encodings m and N.

Hence, while the coins and the encodings M in Frankel, Tsiounis, and Yung (1996) are constructed using the same pair of group generators in a discrete-logarithm setting, we use four generators, and construct coins and M using different pairs. The blind signature is made on the product. Note that more generators in conjunction with Brands' system have been used several times in the past, e.g., in Brands (1993), Brickell, Gemmell, and Kravitz (1995), Frankel, Tsiounis, and Yung (1998). The *restrictiveness assumption* of the blind signature scheme (see Assumption 1), together with proofs of knowledge that the values are formed over the correct generators, guarantee that a buyer cannot decompose the product in two non-corresponding ways at both sides. The security of $\mathcal{RC}$ relies on the correct decomposition, and $\mathcal{RC}$ cannot trust the merchants to verify zero-knowledge proofs in fingerprinting correctly. Hence, one aspect of the decomposition, i.e., the fact that the buyer knows the discrete logarithm of pk_{text} over the correct generator, is only substantiated by a Schnorr signature (Schnorr 1991) towards $\mathcal{RC}$. We will explain this in more details in Section 4.7.6.1 where we handle $\mathcal{RC}$'s security.

4.7.5 Construction

We now present the details of the construction which we informally described in the previous section.

4.7.5.1 System Setup and Prerequisites

For simplicity, we assume that there is only one registration center. The merchant $\mathcal{M}$ and the buyer $\mathcal{B}$ generate the required keys and parameters as follows:

- **Algebraic structure:** We use a cyclic group G of order $|G| := q$ where efficient algorithms are known for multiplying, inverting, determining equality of elements, testing membership and randomly selecting of elements. Further requirements are: The computation of *Decisional Strong Diffie-Hellman* (DSDH) (see Sections 2.2 and 2.6) in G should be infeasible, the generators must be truly random,[27] and no party, even $\mathcal{RC}$ who will typically generate the underlying algebraic group, should know the relative discrete logarithms of the generators to each other. We use the notation $par_{\mathsf{AnoFP}} \leftarrow \mathsf{GenParAnoFP}(par_{sec})$

[27]One way to verify the randomness of the generators is as follows: Select a non-secret string r of a certain length uniformly and randomly, e.g., by using an old random number table. Using r, generate primes q and p and elements $e_i \in \mathbb{Z}_p^*$ deterministically. Compute the generators as $g_i \equiv e_i^{(p-1)/q}$. If a g_i is not a generator, repeat its choice.

for the procedure of generating the (global) systems parameters (e.g., the group and generators, etc).

For concrete constructions, one can take the group family $\mathbb{Z}_{p/q}^*$ (see Sections 2.1.7 and 2.2). Once and for all, a group G from this family, and different generators g, g'', h'', g_1, g_2, g_3, $g_4 \in G \setminus \{1\}$ are selected.

- **Hash functions:** The hash functions *hash* and *hash'* are fixed for the underlying protocols of Brands (Sections 4.7.2.4 and 3.4.4.1) and Schnorr (3.4.3.3) signatures.
- **Key generation and distribution:** Each party X generates a key pair (sk_X, pk_X) of an underlying signature scheme, and distributes pk_X reliably and authenticated to other involved parties, if required. We will use pk_X as representative for X's identity. For $\mathcal{RC}$, this key pair has the following form $sk_{\mathcal{RC}} := x \in_{\mathcal{R}} \mathbb{Z}_q^*$ and $pk_{\mathcal{RC}} := h \equiv g^x \bmod p$. We will use (x, h) in the following.
- **Embedding keys:** $\mathcal{M}$ generates the keys and other required parameters for the embedding/extracting procedure, i.e., $key_{emb} \leftarrow \mathsf{GenKeyEmbed}(par_{emb})$ where $key_{emb} := (key_{emb}^s, key_{emb}^p)$ with key_{emb}^s denoting the secret part of the embedding/extracting parameters, and key_{emb}^p denotes the public part (see Sections 4.5.4 and 5.3.1)
- **Commitment schemes:** The underlying commitment schemes are the QR and the DL commitment schemes denoted by ($\mathsf{GenParQR}()$, $\mathsf{ProtComQR}()$, $\mathsf{ProtOpenQR}()$) and ($\mathsf{GenParDL}()$, $\mathsf{ProtComDL}()$, $\mathsf{ProtOpenDL}()$) (see Sections 3.5.4 and 3.5.5).

  The parameters for the DL-commitment are $par_{\mathsf{com}}^{\mathsf{DL}} = (p, q, g'', h'')$. The parameter for the QR-commitment is $par_{\mathsf{com}}^{\mathsf{QR}} = n'$ where $n' := p'q'$, and is generated by the buyer $\mathcal{B}$ during the embedding subprotocol of each fingerprinting protocol-run.[28] She should be the only party who knows the factorization of n', denoted by $key_{\mathsf{com}}^{\mathsf{QR}} := (p', q')$, which enables her to open any QR-commitment computed with the public parameter n'.

4.7.5.2 Registration

The registration protocol is given in Figures 4.7 and 4.8. In the following we explain the figures and relate them to the detailed descriptions.

[28]The generation of n' is such that only $\mathcal{B}$ knows the factorization of n' while other parties (here $\mathcal{M}$) should be convinced that n' has the correct form (see Section 3.5.4)

Figure 4.7 Proof of correctness of the registration

$$
\begin{array}{rcl}
\mathcal{B} & & \mathcal{RC} \\
i, j, i', y \in_R \mathbb{Z}_q^* & & \\
h_1 := g_1^i; h_3 := g_3^i & & \\
pk_{text} := g_4^{i'} & & \\
enc := (d_1, d_2) := (h_3^y pk_{text}, g_3^y) & & \\
sig_{coin} \leftarrow \mathsf{Sign}(pk_{\mathcal{B}}; h_1, h_3, enc) & & \\
(M_1, M_2) := (g_3^j, pk_{text}^j) & & \\
 & \xrightarrow[sig_{coin}, M_1, M_2]{h_1, h_3, enc,} & \\
 & & h_1 g_2 \neq 1? \\
 & & \text{Verify } sig_{coin} \\
 & & M_1 \neq 1? \\
 & \xleftrightarrow{\text{correctness proof}} & \\
N := h_1 g_2 M_1 M_2 & & N := h_1 g_2 M_1 M_2
\end{array}
$$

Figure 4.8 Blind signature part of the registration

$$
\begin{array}{rcl}
\mathcal{B} & & \mathcal{RC} \\
 & & z \leftarrow N^x \\
 & & w \in_R \mathbb{Z}_q \\
 & & a \leftarrow g^w \bmod p \\
s \in_R \mathbb{Z}_q^* & \xleftarrow{z, a, d} & d \leftarrow N^w \bmod p \\
z' \leftarrow z^s,\ N' \leftarrow N^s & & \\
u,\ v \in_R \mathbb{Z}_q & & \\
a' \leftarrow a^u g^v,\ b' \leftarrow d^{su} N'^v & & \\
c' \leftarrow hash(N', z', a', b', pk_{text}) & & \\
c \leftarrow c'/u \quad \bmod q & \xrightarrow{c} & \\
g^r \overset{?}{\equiv} ah^c,\ N^r \overset{?}{\equiv} dz^c \bmod p & \xleftarrow{r} & r \leftarrow cx + w \bmod q \\
r' \leftarrow ru + v \bmod q & & \\
coin' \leftarrow (N', pk_{text}, \tau') & &
\end{array}
$$

1. **Opening a one-time account:** $\mathcal{B}$ chooses the identity proof $i \in_R \mathbb{Z}_q^*$ randomly and secretly, and computes $h_1 :\equiv g_1^i$ (with $h_1 g_2 \neq 1$), the "account number" from Brands' system, and $h_3 :\equiv g_3^i$, which we

introduced specially as a public key for the ElGamal encryption.

2. **Coin key and encryption:** The value i', also selected secretly and randomly by $\mathcal{B}$, serves as the **secret coin key** and $pk_{text} :\equiv g_4^{i'} \bmod p$ as the corresponding public key. $\mathcal{B}$ encrypts this public coin key into a ciphertext enc using ElGamal encryption where h_3 is the applied public key. She computes a signature $sig_{coin} \leftarrow \mathsf{Sign}(pk_{\mathcal{B}}; h_1, h_3, enc)$ under her normal identity, and sends it to $\mathcal{RC}$, who verifies it. This signature later shows that $\mathcal{B}$ is responsible for this "account" identified by the keys h_1 and h_3 and for the public key pk_{text} encrypted in enc.

3. **Encoding for delayed verifiable encryption:** The pair $(M_1, M_2) := (g_3^j, pk_{text}^j)$ is the additional encoding of pk_{text} whose content is invariant under the subsequent blinding operation. $\mathcal{RC}$ will verify that $M_1 \neq 1$. The content is uniquely defined because $M_1 \neq 1$ uniquely defines $j \neq 0$, and then M_2 and j uniquely define pk_{text}.

4. **Correctness proofs:** $\mathcal{B}$ sends the public values to $\mathcal{RC}$ and gives certain correctness proofs. Intuitively, this is in particular that h_1 and h_3 contain the same identity proof i, and that the content of the encryption (which is uniquely defined given h_3) equals the content of the pair (M_1, M_2) as defined above. Formally, $\mathcal{B}$ has to give a zero-knowledge proof of language membership that the values (i, j, i', y) exist such that the public values, i.e., h_1, h_3, enc, M_1, M_2 fulfill the prescribed equations, i.e., the underlying language is

$$\begin{aligned} L \;:=\; & \{(h_1, h_3, d_1, d_2, M_1, M_2) \mid \exists\, (i, i', j, y) \in (\mathbb{Z}_q^*)^4 : \\ & h_1 \equiv g_1^i \wedge h_3 \equiv g_3^i \wedge d_1 \equiv g_3^{iy} g_4^{i'} \wedge \\ & d_2 \equiv g_3^y \wedge M_1 \equiv g_3^j \wedge M_2 \equiv g_4^{i'j}\}. \end{aligned}$$

We denote the protocol for the proof of correctness with $\mathsf{ProveCor}()$.

One way to implement this protocol is to apply the simple protocol $\mathsf{ProveEqDL}()$ (Section 3.7.3) for i, and the specific **indirect discourse proof** from Frankel, Tsiounis, and Yung (1996) for the remaining parameters. However, there is also a general efficient zero-knowledge proof system for proving low-degree polynomial relations in exponents comprising this and many similar situations. Such systems are proposed in Camenisch and Stadler (1997), Camenisch (1998), Camenisch and Michels (1999a). Note that it is not possible to use exactly the same type of proof as in Section 3.7.3 for the other values because one equation is $M_2 = g_4^{ji'}$, where neither g_4^j nor $g_4^{i'}$ can be public. Here, we can apply Camenisch's techniques for polynomials, e.g., by first committing to the secret inputs using information-theoretically hiding commitments, and then proving the desired relations in zero-knowledge. The commitments can be interpreted as blinded versions

$g^r g_4^{i'}$, $g^{r'} g_4^j$ of the required intermediate values $g_4^{i'}$, g_4^j where $r, r' \in_{\mathcal{R}} \mathbb{Z}_q$ and $g \in G$ is a random generator. For our purposes, we will employ this approach, and use Camenisch's zero-knowledge proofs (for various types of modular relation between secret exponents) in a straight forward and modular manner. For this, the buyer $\mathcal{B}$ proceeds as follows:

(a) Commit to the secret values $(i, i', j, y, i'j, iy)$ using information-theoretically hiding DL-commitment using new random generators $g, h \in G$ (no confusion with other generators such as $\mathcal{RC}$'s public key should occur.) The resulting commitments are $A_0 \equiv g^i h^{r_0}, A_1 \equiv g^i h^{r_1}, A_2 \equiv g^{i'} h^{r_2}, A_3 \equiv g^j h^{r_3}, A_4 \equiv g^y h^{r_4}, A_5 \equiv g^{i'j} h^{r_5}, A_6 \equiv g^{iy} h^{r_6}$ where $r_0, \cdots, r_6 \in_{\mathcal{R}} \mathbb{Z}_q$. We call $A_0, \cdots A_6$ **auxiliary commitments**, and call the secret exponent of the generator g in an auxiliary commitment A_i the *g-part* of A_i.

(b) Send $A_0, \cdots, A_6$ to $\mathcal{M}$, and prove the following predicates in zero-knowledge:

   i. the g-part of A_0 is equal to $\log_{g_1}(h_1)$ and
   ii. the g-part of A_0 is equal to $\log_{g_3}(h_3)$ and
   iii. the g-part of A_0 is equal to the g-part of A_1 and
   iv. the g-part of A_2 is equal to the g_4-part of the representation of d_1 and
   v. the g-part of A_3 is equal to $\log_{g_3}(M_1)$ and
   vi. the g-part of A_4 is equal to $\log_{g_3}(d_2)$ and
   vii. the g-part of A_5 is equal to $\log_{g_4}(M_2)$ and
   viii. the g-part of A_5 is equal to the g-part of A_2 multiplied by the g-part of A_3 and
   ix. the g-part of A_6 is equal to the g-part of A_1 multiplied by the g-part of A_4.

For each of the these proofs there is an explicit efficient zero-knowledge proof (see e.g., Camenisch (1998), Camenisch and Michels (1999a), Boudot (2000)). Thus, there exists a simulator for each of them, and the overall simulator can be constructed from these simulators. Note that one may construct more efficient protocol(s) for proving this instead of the straight forward use of the above building blocks. However, for the reasons we will see in Section 4.7.6.3 on buyer's anonymity, we choose this type of construction.

5. **Withdrawal:** Now, $\mathcal{RC}$ gives a blind signature on the combination of a coin and the encoding (M_1, M_2). Let $m :\equiv g_1^i g_2 = h_1 g_2$ be the value (typically signed in Brands' blind signature scheme), $M :\equiv M_1 M_2$, and $N \equiv mM$. This N is the common input to the blind signing protocol, i.e.,

$$(\mathcal{B} : (N', pk_{text}), \tau';\ \mathcal{RC} : -) \leftarrow \mathsf{BlindSignBr}(\mathcal{B} : -;\ \mathcal{RC} : x;\ * : N)$$

shown in Figure 4.8 for completeness. In Brands (1994), an additional value pk_{coin} is included in the hashing; we use pk_{text} in that place (see also Section 4.7.2.4). As a result, $\mathcal{B}$ obtains the "coin" $coin' = (N', pk_{text}, \tau')$, where $N' \equiv (mM)^s$, and $\tau' = (z', a', b', r')$ is called the signature on (N', pk_{text}).[29] We denote the blinded versions of m and M by $m' \equiv m^s \equiv g_1^{is} g_2^s$ and $M' \equiv M^s \equiv g_3^{s'} pk_{text}^{s'}$, where $s' = sj$.

4.7.5.3 Fingerprinting

The main common input in fingerprinting is the purchase description $text$. We assume that each $text$ is fresh for both buyer and merchant in this protocol, i.e., neither of them uses a value $text$ twice. This can be achieved by a number of standard techniques. The individual protocol steps are as follows:

1. **Text signing and coin verification:** $\mathcal{B}$ selects an unused coin $coin' = (N', pk_{text}, \tau')$. He uses the corresponding secret key i' to make a Schnorr signature sig_{text} on $text$ where we include pk_{text} in the hashing). $\mathcal{B}$ sends ($coin'$, m', M', s', sig_{text}) to $\mathcal{M}$. Now, $\mathcal{M}$ first verifies the blind signature: He computes $c' \equiv hash(N', z', a', b', pk_{text}) \bmod q$ and tests whether $g^{r'} \equiv a'h^{c'}$ and $N'^{r'} \equiv b'z'^{c'} \bmod p$ hold. We say that a coin is valid if and only if it passes these tests. He then verifies sig_{text} using pk_{text} from $coin'$.

2. **Verification of decomposition:** $\mathcal{M}$ first verifies that $N' \equiv m'M'$, $N' \neq 1$ and $m' \neq 1$. Then $\mathcal{B}$ proves to $\mathcal{M}$ in zero-knowledge that he knows a representation of m' with respect to (g_1, g_2), and of pk_{text} with respect to g_4. The zero-knowledge proof protocol $\mathsf{ProveRep}()$ for the former is given in Section 3.7.2, and for the latter the protocol $\mathsf{ProveDL}()$ is given in Section 3.7.1.

3. **Delayed part of verifiable encryption:** $\mathcal{M}$ verifies whether $M' \equiv g_3^{s'} pk_{text}^{s'}$ holds. Details why this verification is sufficient can be seen in the proof of the security for the registration center in Section 4.7.6.1.

4. **Proving equality of committed and encoded numbers:** $\mathcal{B}$ takes the representation (is, s) of $m' \equiv g_1^{is} g_2^s$ as the value $\vec{emb}$ to be embedded in the underlying work, i.e., $\vec{emb} := (is, s)$. In the following we set $r_1 = is$, $r_2 = s$. Since $\mathcal{M}$ should not get any useful information on $\vec{emb}$, $\mathcal{B}$ hides it in a commitment. While the certificate-based framework in Pfitzmann and Waidner (1997a) leaves the type of commitment open (Section 4.6.3), we have to provide an efficient link from the given "encoding" m' of $\vec{emb} := (r_1, r_2)$ to the type of commitments

[29] In the sense of Section 4.7.4 this is not only the coin, but also still contains the blinded encoding of pk_{text}. However, in the following, it is simpler to call this unit a coin.

needed in the further stages of the embedding procedure $\mathsf{EmbedProt}()$. For normal fingerprinting (with and without collusion tolerance), these are *quadratic residue commitments* to individual bits of $\vec{emb}$.[30] In our construction, $\mathcal{B}$ first computes the DL commitments to a binary representation of r_1 and r_2. These commitments are denoted by

$$\begin{aligned} C^{\mathsf{DL}}_{emb} &:= com^{\mathsf{DL}}(\mathsf{bin}(\vec{emb}), \vec{x}, p, q, g'', h'') \\ &= \left[\ com^{\mathsf{DL}}(r_{u,w}, x_{u,w}, p, q, g'', h'')\ \right]_{1\leq u\leq 2,\, 0\leq w\leq l-1} \end{aligned}$$

where $x_{u,w} \in_{\mathcal{R}} \mathbb{Z}_q$.

$\mathcal{B}$ sends C^{DL}_{emb} to $\mathcal{M}$, and proves (in zero-knowledge) that C^{DL}_{emb} is a DL-commitment to the same value encoded in m', i.e., the content of the commitments C^{DL}_{emb} is a pair $(r_1, r_2) \in \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ with $m' \equiv g_1^{r_1} g_2^{r_2} \bmod p$. This can be done using the zero-knowledge protocol shown in Figure 4.9.

Figure 4.9 Proving equality of the contents of m' and C^{DL}_{emb}: $\mathsf{ProveEqComEmb}()$

$$\begin{array}{rcl}
\mathcal{B} & & \mathcal{M} \\
 & u = 1,2 \text{ and } 0 \leq w \leq l-1 & \\
(r_1, r_2) \leftarrow \vec{emb} := (is, s) & & \\
r_u = \sum_{w=0}^{l-1} r_{u,w} 2^w & & \\
x_{u,w} \in_R \mathbb{Z}_q & & \\
x_u \leftarrow \sum_{w=0}^{l-1} x_{u,w} 2^w & & \\
com^{\mathsf{DL}}_{u,w} \leftarrow g''^{r_{u,w}} h''^{x_{u,w}} & \xrightarrow{\quad com^{\mathsf{DL}}_{u,w} \quad} & com^{\mathsf{DL}}_{u} \leftarrow \prod_{w=0}^{l-1} (com^{\mathsf{DL}}_{u,w})^{2^w} \\
\text{REPEAT } \gamma_s \text{ times} & & \\
e_u, y_u \in_R \mathbb{Z}_q^* & & \\
V \leftarrow g_1^{e_1} g_2^{e_2} & & \\
rcom^{\mathsf{DL}}_{u} \leftarrow g''^{e_u} h''^{y_u} & \xrightarrow{\quad rcom^{\mathsf{DL}}_{u}, V \quad} & \\
res_u \leftarrow cr_u + e_u \bmod q & \xleftarrow{\quad c \quad} & c \in_R \mathbb{Z}_2 \\
z_u \leftarrow cx_u + y_u \bmod q & \xrightarrow{\quad res_u, z_u \quad} & g_1^{res_1} g_2^{res_2} \overset{?}{\equiv} V m'^c \\
 & & g''^{res_u} h''^{z_u} \overset{?}{\equiv} rcom^{\mathsf{DL}}_{u} (com^{\mathsf{DL}}_{u})^c \\
\text{END REPEAT} & &
\end{array}$$

[30]For traitor tracing, they are quadratic residue commitments to small blocks of emb represented in unary. Such a representation can also be derived efficiently from commitments to the bits.

This proof protocol is similar to other proofs concerning knowledge of representations of numbers with respect to certain generators as described in Section 3.7.2. As usual, we could also use larger challenges c at the cost of the real zero-knowledge property. We denote this protocol by

$$(\mathcal{B} : -;\ \mathcal{M} : \mathit{ind}) \leftarrow \mathsf{ProveCorComEmb}(\mathcal{B} : \mathit{state}_{\mathcal{B}}^{C_{emb}^{\mathsf{DL}}};\ \mathcal{M} : -;\ * : C_{emb}^{\mathsf{DL}}, m').$$

Note that this protocol proves only that $\sum r_{u,w} = r_u$ but not that the values $r_{u,w}$ are binary; such a proof will be a side effect of the embedding protocol when DL commitments are mapped to QR-commitments. Moreover, in our definition of embedding protocol (Section 4.5.4), we assumed the common input C_{emb} to the embedding protocol to have the correct form. For our particular instantiation, however, the embedding protocol itself takes care of this issue, and verifies that C_{emb} indeed consists of the commitments to the bits of $\vec{emb}$.

5. **Embedding:** On common input C_{emb}^{DL}, $\mathcal{B}$ and $\mathcal{M}$ run the embedding protocol

$$(\mathcal{B} : \mathit{work}^{\mathit{fing}};\ \mathcal{M} : \mathit{rec}_{\mathcal{M}}^{\mathit{work}}) \leftarrow \mathsf{EmbedProt}(\mathcal{B} : \mathit{state}_{\mathcal{B}}^{C_{emb}^{\mathsf{DL}}};\ \mathcal{M} : \mathit{work}, \mathit{key}_{emb}^{s};\ * : C_{emb}^{\mathsf{DL}}, \mathit{key}_{emb}^{p}).$$

Section 5.3 gives a detailed construction for this protocol.

4.7.5.4 Identification

1. **Merchant retrievals:** On redistribution of a work $\mathit{work}^{\mathit{red}}$ for which $\mathit{true} = \mathsf{simtest}(\mathit{work}, \mathit{work}^{\mathit{red}})$ holds, $\mathcal{M}$ extracts a value $\vec{emb} = (r_1, r_2)$ from the redistributed work using the extraction algorithm

$$\vec{emb} \leftarrow \mathsf{ExtractProt}(\mathit{work}^{\mathit{red}}, \mathit{key}_{emb}, \mathit{REC}_{\mathcal{M}}^{\mathit{work}})$$

described in Section 5.3 (see also Section 4.5.4.1). This value should be $\vec{emb} = (r_1, r_2)$ with $r_2 \neq 0$; thus he sets $s := r_2$ and $i := r_1/r_2$. He computes $m' \equiv g_1^{is} g_2^{s} \bmod p$, and uses it to retrieve coin', M', text and $\mathit{sig}_{\mathit{text}}$ from the corresponding purchase record of the given work. If any of these steps do not succeed, he gives up. (The collusion tolerance of the underlying code might have been exceeded.) Otherwise, he sends to $\mathcal{RC}$ the triple $\mathit{proof}_{\mathcal{RC}} := (i, \mathit{text}, \mathit{sig}_{\mathit{text}})$.

2. **Registration center retrieval:** On input $\mathit{proof}_{\mathcal{RC}}$, the registration center $\mathcal{RC}$ searches in its registration database for a buyer who has

registered the one-time account number $h_1 \equiv g_1^i$, and retrieves the values $(pk_{\mathcal{B}}, enc, sig_{coin})$, where $pk_{\mathcal{B}}$ corresponds to the real identity of $\mathcal{B}$.

$\mathcal{RC}$ refuses identification if it is clear from *text* that the redistribution was legal. Otherwise, $\mathcal{RC}$ decrypts *enc* using i to obtain pk_{text}, and verifies that sig_{text} is a valid signature on the values $(text, pk_{text})$ for this public key pk_{text} with respect to the generator g_4.[31] If positive, $\mathcal{RC}$ sends the retrieved values to $\mathcal{M}$.

3. **Merchant verification:** If $\mathcal{M}$ gets an answer $(pk_{\mathcal{B}}, enc, sig_{coin})$ from $\mathcal{RC}$, he first verifies that sig_{coin} is a valid signature with respect to $pk_{\mathcal{B}}$ on the triple $(h_1 \equiv g_1^i,\ h_3 \equiv g_3^i,\ enc)$. He also verifies that *enc* correctly decrypts to the value pk_{text} contained in $coin'$ with respect to the secret key i and the generator g_3. If one of these tests fails, or $\mathcal{M}$ receives no answer, he starts enforced identification.

4.7.5.5 Enforced Identification

If $\mathcal{M}$ has to enforce the cooperation of $\mathcal{RC}$, he sends

$$proof'_{\mathcal{RC}} := (coin', s', i, s, text, sig_{text})$$

to an arbiter $\mathcal{J}$. $\mathcal{J}$ verifies the validity of $coin'$ and calls its components (N', pk_{text}, τ') as usual. Then she verifies that $N' \equiv m'M'$ for $m' \equiv g_1^{is} g_2^s \bmod p$ and $M' \equiv g_3^{s'} pk_{text}^{s'} \bmod p$. Finally, she verifies that sig_{text} is a valid signature on the values $(text, pk_{text})$ for the public key pk_{text} with respect to the generator g_4.

If any of these tests fails, $\mathcal{J}$ rejects $\mathcal{M}$'s claim. Otherwise, she sends $proof_{\mathcal{RC}} := (i, text, sig_{text})$ to $\mathcal{RC}$ and requires values $(pk_{\mathcal{B}}, enc, sig_{coin})$. Then $\mathcal{J}$ verifies them as $\mathcal{M}$ does in Step 3 of identification.

4.7.5.6 Trial

$\mathcal{M}$ tries to convince an arbiter $\mathcal{J}$ that $\mathcal{B}$, with the identity $pk_{\mathcal{B}}$, has illegally redistributed the work bought under the rights described in *text*. Note that in the following $\mathcal{B}$ is *not* required to participate in the trial.

1. **Proof string:** $\mathcal{M}$ sends to $\mathcal{J}$ the proof string

$$proof = (coin', s', i, s, sig_{text}, enc, sig_{coin}).$$

[31] This is necessary for the security of $\mathcal{RC}$ by guaranteeing that the division of N' into m' and M' is correct, even if $\mathcal{RC}$ is supposed to identify all redistributors independent of *text*.

2. **Verification of identity proof** i**:** $\mathcal{J}$ computes $h_1 \equiv g_1^i$ and $h_3 \equiv g_3^i \bmod p$, and verifies that sig_{coin} is a valid signature on (h_1, h_3, enc) with respect to $pk_{\mathcal{B}}$. If yes, it means that i, the discrete logarithm of an account number h_1 for which $\mathcal{B}$ was responsible, has been recovered by $\mathcal{M}$, and thus, as we will see, $\mathcal{B}$ has redistributed some work. It remains to verify the link to *text*.

3. **Verification of** *text***:** $\mathcal{J}$ verifies the validity of $coin'$, and calls its components $(N'$, pk_{text}, $\tau')$. She then verifies that $N' \equiv m'M'$ for $m' \equiv g_1^{is} g_2^s \bmod p$ and $M' = g_3^{s'} pk_{text}^{s'} \bmod p$. She also verifies the signature sig_{text} on the disputed text *text* with respect to pk_{text} and the generator g_4. These verifications imply that if the accused buyer owned this coin, he must have spent it in the disputed purchase on *text*. Finally, $\mathcal{J}$ verifies that this coin belongs to $\mathcal{B}$: She tests whether enc correctly decrypts to pk_{text}, if one uses i as the secret key. If all verifications are passed, $\mathcal{J}$ finds $\mathcal{B}$ guilty of redistribution, otherwise she rejects ($\mathcal{M}$ should be declared as the cheating party).

4.7.6 Security Analysis

In this section, we present proofs for the security of the construction according to the security requirements outlined in Section 4.6.2. We recall these requirements briefly in the corresponding security analysis section.

Note that the requirements *completeness* and *no jamming by registration center* are straightforward to verify for our scheme (see also Remark 4.16). Hence, we immediately proceed with the main security requirements.

4.7.6.1 Security for Registration Center

We prove that if the registration center $\mathcal{RC}$ is honest, an honest arbiter will never output that $\mathcal{RC}$ is guilty.

We need the restrictiveness of the underlying blind signature scheme for showing that the value m' used in fingerprinting "contains" the same identity proof value i as the original m, and also that the delayed verification of pk_{text} works. Brands (1994) only works with two generators g_1, g_2, while we use four. However, in the underlying report (Brands 1993) the same assumptions are made for any number of generators $g_1, \ldots, g_n$. Coin systems with more than two generators have also been presented in Brickell, Gemmell, and Kravitz (1995) and Frankel, Tsiounis, and Yung (1998). The exact assumption we need is the following:

Assumption 2 (Restrictiveness with Schnorr signature) Let k be a security parameter, $Q \in \mathsf{poly}(k)$, and $\mathcal{A} \in \mathcal{UPTM}$ the adversary that can interact with a Brands' signer (as in Figure 4.8) Q times for messages N of its choice where $\mathcal{A}$ also has to output representations of all these messages, i.e., quadruples $(i_1, \ldots, i_4)$ such that

$$N = g_1^{i_1} \cdots g_4^{i_4}.$$

At the end, $\mathcal{A}$ has to output a message N' with a valid signature τ' and a representation $(i'_1, \ldots, i'_4)$ of N' except that it need not show i'_4, but only values $(h'_4, i''_4, \mathit{msg}, \mathit{sig}_{\mathit{msg}})$ such that $N' = g_1^{i'_1} \cdots {h'_4}^{i''_4}$ and $\mathit{sig}_{\mathit{msg}}$ is a valid Schnorr signature on msg for the public key h'_4, and the generator g_4 (with h'_4 included in the hashing). We then define $i'_4 := i''_4 \log_{g_4}(h'_4)$.

Then for all Q and all $\mathcal{A}$ the probability that $\mathcal{A}$ fulfills all the mentioned conditions, and that the vector $(i'_1, \ldots, i'_4)$ is not a scalar multiple of one of the vectors $(i_1, \ldots, i_4)$ is negligible in k where the probability is taken over the random choices of the signer and $\mathcal{A}$.[32]

Remark 4.17. Note that in the "normal" restrictiveness assumption (see Section 4.7.2.4), the adversary has to output complete representations of both, the blinded and unblinded values, i.e., also i'_4. In our case, we had to make the above, *stronger* assumption. This is because the adversary only outputs a factor i''_4 of i'_4, and, instead of the other factor $i' := i'_4/i''_4$, a Schnorr signature with respect to the corresponding public key $h'_4 = g_4^{i'}$. The intuitive idea why this is reasonable is that a Schnorr signature is a non-interactive proof of knowledge of the secret key in random oracle model. Such arguments are mentioned, e.g., in Brands (1994), Frankel, Tsiounis, and Yung (1996), Frankel, Tsiounis, and Yung (1998) and Camenisch (1998). ∘

Lemma 4.1 *The construction for the anonymous fingerprinting scheme from Section 4.7.5 is secure for the registration center under the restrictiveness assumption with Schnorr signature (Assumption 2).* □

Proof. $\mathcal{RC}$ can be found guilty by an arbiter $\mathcal{J}$ only in enforced identification, because in a trial $\mathcal{J}$ only finds either $\mathcal{B}$ or $\mathcal{M}$ guilty. There are two possibilities how this could happen:

1. If the adversary can convince $\mathcal{J}$ to enforce identification by showing $\mathit{proof}'_{\mathcal{RC}}$, but $\mathcal{RC}$ does not find a registration under the corresponding account number h_1.

2. If $\mathcal{RC}$ finds such a registration, but the values $(pk_{\mathcal{B}}, \mathit{enc}, \mathit{sig}_{\mathit{coin}})$ do not pass the arbiter's tests. Since $\mathcal{RC}$'s own tests during registration guarantee the correctness of $\mathit{sig}_{\mathit{coin}}$, the remaining possibility is that enc does not decrypt to the value pk_{text} shown in $\mathit{proof}'_{\mathcal{RC}}$.

[32]One may formulate this assumption in a general way, as we have done for the normal restrictiveness assumption (Assumption 1, in Section 4.7.2.4).

In both cases, the tuple $\mathit{proof}'_{\mathcal{RC}}$ shown by the adversary must contain a valid coin $\mathit{coin}' = (N', \mathit{pk}_{\mathit{text}}, \tau')$ and values (i, s, s') such that $N' = m'M'$ with $m' \equiv g_1^{is} g_2^{s}$ and $M' \equiv g_3^{s'} \mathit{pk}_{\mathit{text}}^{s'}$. Moreover, it must contain a Schnorr signature $\mathit{sig}_{\mathit{text}}$ on some *text*, which is valid with respect to $\mathit{pk}_{\mathit{text}}$ and the generator g_4. Hence, as far as the output is concerned, the adversary needs exactly what it also needs in our restrictiveness assumption, with $(i'_1, \ldots, i'_4) = (is, s, s', i's')$ where i' is defined as the discrete logarithm of $\mathit{pk}_{\mathit{text}}$ with respect to g_4.

Furthermore, the registration protocol guarantees that the adversary has to prove knowledge of a representation of each common input N for which the blind signature protocol is executed. These are normal interactive proofs of knowledge (see Section 3.7.2); hence from any adversary successful in our scenario we can, together with the extractors, construct another adversary that actually outputs the representations, so that the assumption applies to it.

By the actual predicate proved, these representations are of the form $(i_1, 1, i_3, i_4)$ where $h_1 = g_1^{i_1}$, $M_1 = g_3^{i_3}$, $M_2 = g_4^{i_4}$ hold, and additionally $h_3 = g_3^{i_1}$. The restrictiveness assumption, therefore, guarantees that the quadruple $(is, s, s', i's')$ defined by $\mathit{proof}'_{\mathcal{RC}}$ is a scalar multiple of such a quadruple $(i_1, 1, i_3, i_4)$. The factor can only be s, and if it were 0, the entire quadruple would be 0, and thus, $N' = 1$, in contradiction to the validity of the coin. Thus, $s \neq 0$ and $i_1 = i$, $i_3 = s'/s$, and $i_4 = i's'/s$.

The first equation means that the value i shown in $\mathit{proof}'_{\mathcal{RC}}$ was actually used in a registration. This excludes Case 1 of how $\mathcal{RC}$ could be found guilty.

During this registration, $\mathcal{RC}$ verified in the correctness proof ($\mathsf{ProveCor}()$) that the content of the encryption $\mathit{enc} = (d_1, d_2)$ equals the content of M_2 blinded by the exponent $\log_{g_3}(M_1)$. We have now shown that $M_1 = g_3^{s'/s}$ and $M_2 = g_4^{i's'/s}$. The content of this pair is $g_4^{i'} = \mathit{pk}_{\mathit{text}}$ by the definition in Section 4.7.5.2, Step 3. This also excludes Case 2 of how $\mathcal{RC}$ could be found guilty. ∎

4.7.6.2 Security for Merchant

We prove, if an adversary who engages in at most *collsize* executions of fingerprinting with the merchant for a certain work, and then produces another copy similar to the original, then the merchant will obtain a valid digital identity as an output in identification, together with a *text* used, and a string *proof*, and then wins a trial with any honest arbiter. This should hold even if the registration center is cheating, and belongs to the collusion against the merchant. In this case, the protocol for enforced identification may be needed if normal identification fails, and the output for the arbiter in this protocol may indicate that $\mathcal{RC}$ is guilty.

Lemma 4.2 *Given a collsize collusion-tolerant and secure embedding scheme, the construction of the anonymous fingerprinting scheme from Section 4.7.5 is secure for the merchant under the restrictiveness assumption (Assumption 1, Section 4.7.2.4).* □

Proof. The embedding and extracting algorithms $\mathsf{EmbedProt}()$ and $\mathsf{ExtractProt}()$ guarantee that whenever a collusion of the maximum size *collsize* redistributes a work $work^{red}$, for which $true \leftarrow \mathsf{simtest}(work, work^{red})$ holds, then with very high probability, $\mathcal{M}$ can extract a value $\vec{emb}$ that belongs to a traitor (see also Section 5.3.3). More precisely, $\vec{emb}$ is a pair (r_1, r_2) such that $g_1^{r_1} g_2^{r_2} \equiv m'$ where (r_1, r_2) should be the content of the commitment C_{emb}^{DL} which is input to the embedding protocol $\mathsf{EmbedProt}()$.[33]

Using the value m', $\mathcal{M}$ can retrieve the corresponding values *text* and $(coin'$, M', s', $sig_{text})$ where $coin'$ is a valid coin, sig_{text} is a valid signature on *text*, and for the component N' of $coin'$, the equations $N' \equiv m'M'$ and $M' \equiv g_3^{s'} pk_{text}^{s'}$ hold. Setting $(i, s) \equiv (r_1/r_2,\ r_2)$ implies $g_1^{is} g_2^{s} \equiv m'$. Assuming the restrictiveness assumption (Section 4.7.2.4) holds, r_2 is nonzero, or $\mathcal{M}$ will obtain an evidence that $\mathcal{RC}$ has cheated, i.e., colluded with traitors and signed values with wrong representation. Now $proof'_{\mathcal{RC}} = (coin', s', i, s, text, sig_{text})$ enables him to ask $\mathcal{RC}$ for identification and, in the worst case, have it *enforced* by $\mathcal{J}$, because $\mathcal{J}$ only performs verifications whose validity has just been listed.

Together with the values that $\mathcal{RC}$ must return, $\mathcal{M}$ obtains a valid proof string *proof*: In a trial, $\mathcal{J}$ only performs verifications that $\mathcal{M}$ also performed in the identification or fingerprinting protocol, so that $\mathcal{M}$ will not lose.

$\mathcal{M}$ is also protected from making wrong accusations: Even if there are more than the tolerated number of traitors, $\mathcal{M}$'s verifications in identification (Section 4.7.5.4) guarantee that whenever he makes an accusation he will not lose in the trial.[34] ■

4.7.6.3 Security for Buyer

Frame proofness

The security requirement for the buyer considers an honest buyer $\mathcal{B}$ who correctly follows the protocols and keeps the obtained results secret, in particular the (fingerprinted) work bought. No matter what the other parties do, the output of the trial with an arbiter will not declare this buyer guilty

[33]Note that this is guaranteed by the soundness of the zero-knowledge proof (of knowledge) $\mathsf{ProveCorComEmb}()$ proving that the content encoded in m' is the same as the one in C_{emb}^{DL}, and by the soundness of the embedding protocol for merchant (Section 5.3.3.1).

[34]These verifications verify the unambiguous link between the quantities i, *coin*, pk_{text}, *text*, sig_{coin} and $pk_{\mathcal{B}}$.

of illegal redistribution, even if the adversary can obtain fingerprinted works of $\mathcal{B}$ for certain texts (e.g., by means of past legal redistribution).

Lemma 4.3 *The construction for the anonymous fingerprinting from Section 4.7.5 is frame proof for the buyer under the Quadratic Residuosity Assumption* $1/\mathsf{poly}(k)$-$\mathrm{CQR}(\mathrm{c{:}u; g{:}l; f{:}}\mathbb{Z}_n^{(+1)})$ *and the discrete logarithm assumption* $1/\mathsf{poly}(k)$-$\mathrm{CDL}(\mathrm{c{:}u; g{:}l; f{:}}\mathbb{Z}_{p/q}^*)$*, and under the assumption that the signature scheme* sig_{text} *is secure.*[35] $\square$

Proof. Consider a trial with $\mathcal{B}$ for a specific text $text^*$ generated by $\mathcal{A}$. The verification of the identity proof i in the trial guarantees that $\mathcal{B}$ is only held responsible for one of his own *one-time account* numbers $h_1 \equiv g_1^i$, and that adversary's proof $proof^*$ in the trial must contain i (see also Section 4.7.5.6). Note that one-time account means that the identity proof i is used only once. We consider the possible situations for the adversary to recover the identity proof i:

We first consider the case where the adversary is not given access to the data of a purchase.[36] In this case, the only knowledge the adversary can obtain about i is from:

1. h_1 and h_3,
2. the correctness proofs in registration,
3. the proof of knowledge of a representation of m' with respect to (g_1, g_2) within the fingerprinting protocol, and
4. the embedding protocol $\mathsf{EmbedProt}()$ also within the fingerprinting protocol (Note that the decryptions the buyer makes cannot be used as an oracle here because the results only become known to the adversary in a redistribution.)

In all the other steps, e.g., the ElGamal encryption, the buyer only uses values that are already known, like h_3.

Now, the correctness proofs in the registration protocol are zero-knowledge, and so is also their sequential composition due to the composition theorem for auxiliary input zero-knowledge proofs. The same holds for the proof of knowledge of a representation of m' during fingerprinting. Moreover, the embedding protocol $\mathsf{EmbedProt}()$ is (computationally) zero-knowledge under QRA. This follows from Lemma 5.13 (see Section 5.3.3). Note that in the embedding protocol, it is not in $\mathcal{M}$'s interest to embed any

[35] Here, this is a Schnorr signature which is secure under CDL in the random oracle model (Section 3.4.3.3).

[36] The data may come from a legal distribution, or from an illegal one the buyer may have done once in the past.

other information than $\vec{emb}$ into the underlying work.[37] Further, the QR-commitments are semantically secure under QRA, and the DL-commitments are perfectly hiding. Thus, the adversary $\mathcal{A}$ could try to compute i from h_1, h_3 and the QR-commitments. This, however, implies that $\mathcal{A}$ can either break the QRA or the DL assumption.

Hence, the only way for the adversary to compute i is in fact from the resulting work in a purchase to which the adversary has access. Let *text* be the text used in that purchase, and *coin'* the corresponding coin withdrawn from the account h_1. The adversary may try to frame $\mathcal{B}$ by generating a different text $text^* \neq text$, and linking to $\mathcal{B}$'s identity. For this, it requires a signature sig_{text} on $text^*$ with the pk_{text} from $coin'$. The adversary cannot use any other pk^*_{text} to sign $text^*$ with respect to h_1 because on the one hand, the signature sig_{coin} fixes the encryption *enc* and the public key h_3 to be used with the account number h_1, and on the other hand, *enc* and h_3 together uniquely fix pk_{text}. Thus, $\mathcal{A}$ has to generate a signature sig_{text} on $text^*$ where the verification key is pk_{text}. However, the secret key i' corresponding to pk_{text} in $coin'$ is known only to $\mathcal{B}$, and the only information $\mathcal{B}$ reveals on i' are zero-knowledge proofs in registration and fingerprinting protocols, and the signature sig_{text} on *text*. If the signature scheme is secure against active attacks, this does not help the adversary to forge a valid signature with respect to pk_{text} on $text^*$. Here, sig_{text} is a Schnorr signature which is secure against active attacks in the random oracle model under DL assumption (Section 3.4.3.3). ■

Anonymity

Purchases of honest buyers should not be linkable even by a collusion of all merchants, central parties and other buyers. In our construction, the only information common to all registrations of a buyer is her global key pair $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$.

Lemma 4.4 *The construction for the anonymous fingerprinting scheme from Section 4.7.5 is unlinkable in the random oracle model under the Quadratic Residuosity Assumption* $1/\mathsf{poly}(k)$-CQR(c:u; g:l; f:$\mathbb{Z}_n^{(+1)}$)*, the decisional strong Diffie-Hellman assumption* $1/\mathsf{poly}(k)$-DSDH(c:u; g:l; f:$\mathbb{Z}^*_{p/q}$)*, and the assumption that* $\mathsf{Sign}(;)$ *is a secure signature scheme.* □

Proof. First, we consider the case where *text* allows legal redistribution. In this case, $\mathcal{RC}$ must be *trusted* regarding the recovery of buyers' anonymity *after* a (legal) redistribution. Recall that we use each identity proof i only once. The buyer $\mathcal{B}$ only uses the key pair $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$ to generate the signature sig_{coin}, and uses neither the keys nor this signature in the fingerprinting

[37]Otherwise, he cannot obtain a proof of treachery against cheating buyers.

protocol. Further, due to the buyer's frame proofness requirement (see Section 4.7.6.3), $\mathcal{M}$ should not be able to produce a fake sig_{text} on a fake text $text^*$ even if he is provided with the data from redistributions of a buyer. Therefore, $\mathcal{RC}$ would not provide $\mathcal{M}$ with the identity of this buyer whom $\mathcal{M}$ accuses as traitor. Thus, other fingerprintings and possible redistributions of a buyer are statistically independent of one registration and the corresponding fingerprinting.

Next, we consider the case where *text* does not allow legal distribution. We focus on the question whether a pair of views from the registration $view^{reg}$ and the corresponding fingerprinting $view^{fing}$ protocol are linkable.

Following the requirement for unlinkability given in Section 4.6.2, we let an adversary carry out two registrations and then the two corresponding fingerprintings in random order according to a uniformly chosen bit $b \in_{\mathcal{R}} \{0, 1\}$. We show that the adversary will not be successful in guessing which views correspond to each other, otherwise one can use this adversary to break a reasonable cryptographic assumption. The proof consists of several scenarios where each scenario defines certain views of the registrations and the corresponding fingerprintings. We consider the main scenarios $S_1, \cdots, S_8$ (Note that each scenario may also contain several sub-scenarios). The first scenario S_1 is called the real world scenario where the registration and fingerprinting views (in random order) are given as in the real protocols. The last scenario S_8 defines "constructed" views of registrations and fingerprinting which are completely independent of the bit b, i.e., no distinguisher can ever guess this bit with a probability better than pure guessing. We prove that all subsequent pairs of scenarios between S_1 and S_8 are indistinguishable, otherwise we can use the corresponding scenario distinguisher, denoted with D_S, to construct an algorithm which breaks a cryptographic assumption. Finally, it follows from the transivity of indistinguishability that also in the real world no distinguisher for the bit b can exist, and thus, the views are unlinkable.

We proceed with the proof as outlined above for the transactions of two *different* buyers. The proof for unlinkability of the transactions of the same buyer is similar.

S_1**:** The adversary runs the first registration protocol $\mathsf{Register}()$ with one of the buyers. The outputs are this buyer's view $view^{reg}_{\mathcal{B},0}$ and the view of the adversary $view^{reg}_{\mathcal{A},0}$. We denote this view by the pair $(recv^{reg}_{\mathcal{A},0}, state_{\mathcal{A}})$ where $recv^{reg}_{\mathcal{A},0}$ denotes the messages from $\mathcal{B}$ to $\mathcal{A}$, while the variables $state_{\mathcal{A}}$ model the adversary's entire state between protocol executions. Similarly, the second registration protocol is executed.

Now a bit b is uniformly chosen; it denotes on which registration the first execution of the fingerprinting protocol $\mathsf{Fing}()$ is based, assuming that the registrations succeeded from the buyers' point of view.

The values sent by $\mathcal{B}$ are

$$\begin{aligned} recv_{\mathcal{A},0}^{\mathsf{Reg}} &= (pk_{\mathcal{B},0}, h_{1,0}, h_{3,0}, M_{1,0}, M_{2,0}, enc_0, sig_{coin,0}, c_0, recv_{\mathcal{A},0}^{\mathsf{ProveCorReg}}), \\ recv_{\mathcal{A},b}^{\mathsf{Fing}} &= (coin'_b, m'_b, M'_b, s'_b, sig_{text,b}, recv_b^{\mathsf{Embed}}, recv_{\mathcal{A},b}^{\mathsf{ProveCorFing}}), \end{aligned}$$

and similarly for $recv_{\mathcal{A},1}^{reg}$ and $recv_{\mathcal{A},\bar{b}}^{fing}$. The meaning of the notations is as follows: $recv_{\mathcal{A},0}^{\mathsf{ProveCorReg}}$ denotes the received messages of the correctness proofs in the registration, and $recv_{\mathcal{A},0}^{\mathsf{ProveCorFing}}$ denotes the received messages of the correctness proofs in the fingerprinting. We define the latter as $(recv_{\mathcal{A},b}^{\mathsf{ProveCorDecom}}, recv_{\mathcal{A},b}^{\mathsf{ProveCorComEmb}})$ where $recv_{\mathcal{A},b}^{\mathsf{ProveCorDecom}}$ denotes the received messages for the proof of correct decomposition/representation, and $recv_{\mathcal{A},b}^{\mathsf{ProveCorComEmb}}$ denotes the received messages for the proof of correctness of the commitment C_{emb}^{DL} input to the embedding subprotocol. Further, c_0 is the only value sent in the withdrawal subprotocol, $coin'_b = (N'_b, pk_{text,b}, \tau'_b)$ is the coin, $recv_b^{\mathsf{Embed}}$ denotes the received messages in the embedding protocol (during fingerprinting). The texts to be signed may be chosen adaptively by $\mathcal{A}$ in $\mathsf{Fing}()$.

In the following scenarios, we show that certain components from the above tuples can be simulated (either by using other components or by random and independent values). For readability, we will fade out these components in the corresponding tuple. However, note that for the overall simulation to be correct these components must be considered in the corresponding tuples and in all subsequent scenarios.

S_2**:** We first derive an equivalent scenario to S_1 where we fade out some components because we can easily simulate them from the other components.

- **Simple simulations:** The triple $(sk_{\mathcal{B}}, pk_{\mathcal{B}}, sig_{coin})$ is not used in any other components; a simulator can therefore generate its own key pair, and compute sig_{coin} with it. Further, it can compute the value N' in $coin'$ as the product of m' and M', and the value M' as $g_3^{s'} pk_{text}^{s'}$.

- **Zero-knowledge proofs simulations:** The correctness proofs in the registration and fingerprinting protocols are zero-knowledge (with perfectly hidden secret inputs), and thus, they can be simulated.[38] Further, the embedding protocol is computationally zero-knowledge (under QRA assumption) where the input to this protocol is hidden in information-theoretically secure DL commitment (see also Section 5.3.3). Thus, it can also be simulated.

[38]Note that all required *auxiliary commitments* (see Section 4.7.5.1) are left out in the views since they are, due to their construction, random group elements and independent of all other values.

- **Coin simulation:** We show how to simulate the blind signature τ' and the challenge c. This does not follow trivially from unlinkability of Brands' scheme because we reuse some internal values from the withdrawal in the rest of the protocol (beyond what is used in Brands' scheme), e.g., we use the blinding exponent s in $s' \equiv sj$. We claim that c can be simulated by an independent random value, and $\tau' = (z', a', b', r')$ by a correct random signature on N', i.e., the simulator chooses $w' \in_R \mathbb{Z}_q^*$ and computes $z' \equiv N'^x$, $a' \equiv g^{w'}$, $b' \equiv N'^{w'}$, $c' \equiv hash(N', z', a', b', pk_{text})$, and $r' \equiv c'x + w'$.[39]

  We show that the real c is also random and the real τ' is a random signature on N', and both are independent of each other and all other values.

  For the randomness of c, even for a cheating $\mathcal{RC}$, we work in the random oracle model (see Section 3.3) for *hash* as done by Frankel, Tsiounis, and Yung (1996) and Frankel, Tsiounis, and Yung (1998). Then for each input tuple to *hash* that has not occurred before, the output c' is random and independent of all values chosen up to this time, in particular u. Furthermore, the inputs of the (honest) buyers contain a new random value pk_{text} each time, and thus, only repeat with negligible probability. Hence, the simulation of c is correct. It follows that even a cheating registration center $\mathcal{RC}^*$ must choose (z, a, d, r) with the correct relations, i.e., such that a value w exists with $z \equiv N^x$, $a \equiv g^w$, $d \equiv N^w$, and $r \equiv cx + w$.[40] Then r is uniquely determined by the verification equation.

  Given the correctness of (z, a, d, r), one can easily show that buyer's signature τ' has the structure claimed above where $w' := wu + v$: In the blind signing protocol we have $z' := z^s = N^{sx} = N'^x$, $a' := a^u g^v = g^{wu} g^v = g^{wu+v}$, $b' := d^{su} N'^v = N^{swu} N'^v = N'^{wu+v}$.

  It remains to show that this w' is uniformly distributed independent of the remaining values in the view. This is true because the (honest) buyer selects u and v randomly and uniformly, and the only other place where u and v are used in the real protocol is in the computation of c, and we have already shown that c can be replaced by an independent random value.

[39]Note that the simulator is given the secret key x of $\mathcal{RC}$ while $\mathcal{RC}$ is a part of the adversary. As mentioned in Section 3.4.4.1 the blind signature protocol is a proof of knowledge of x. Thus, we let our simulator run the corresponding extractor in the registration, and extract the secret key x.

[40]More precisely, if it chooses (z, a, d) differently, it will only pass the buyer's verification, and get information on the bit b with negligible probability. This is because one can easily extract such a w and x from acceptable responses to two challenges $c \neq c^*$.

S_3**:** The remaining components we have to consider now are as follows:

$$\begin{aligned} recv_{\mathcal{A},0}^{\mathsf{Reg}} &= (h_{1,0}, h_{3,0}, M_{1,0}, M_{2,0}, enc_0), \\ recv_{\mathcal{A},b}^{\mathsf{Fing}} &= (pk_{text,b}, m_b', s_b', sig_{text,b}). \end{aligned}$$

In the following scenarios, we are going to simulate certain components in the registration or fingerprinting views by replacing them with random values. Since these components are common inputs to the zero-knowledge protocols $recv_{\mathcal{A},0}^{\mathsf{ProveCorReg}}$ and $recv_{\mathcal{A},b}^{\mathsf{ProveCorFing}}$, it is not guaranteed by the definition of zero-knowledge that there exist a simulator when these inputs are not legal. Thus, we might not be able to get a correct simulation of the overall view for $\mathcal{A}$. However, we can handle this problem by a careful construction of the zero-knowledge protocols as follows: In the real protocol the prover commits to the secret inputs (exponents) using information-theoretically hiding commitments (*auxiliary commitments*), and proves knowledge of them. Then all the zero-knowledge subprotocols within this protocol are performed on these commitments for the proof of modular relations between the secret exponents, etc. Due to the perfect hiding property, these commitments can be opened to any value, and thus, for the simulation, these commitments are always legal for the considered language. Hence, all zero-knowledge subprotocols can be simulated by definition, and the simulator for the main zero-knowledge protocol can be constructed from these simulators. This is the reason why we used this approach for the construction of the protocol $\mathsf{ProveCor}()$ in Section 4.7.5.2.

Hence, each time we replace one of these components, we should keep in mind that the main (zero-knowledge) proof of language membership can be correctly simulated even though the inputs are not legal.

We start with simulating the encryption enc (in S_2) by an ElGamal encryption on a fixed value. This scenario should be indistinguishable from S_2. Otherwise, there exists a scenario distinguisher D_S that we can use to break the semantic security of ElGamal encryption.[41] Note that this is based on the Decisional Diffie-Hellman (DDH) assumption (Sections 2.2 and 2.6).

[41]It suffices to consider only passive adversaries, since the encryption is performed only once per each key, and it is never decrypted. Thus, we can play the "fictive game" with an passive adversary $\mathcal{A}_{enc}$ against the semantic security of the encryption scheme: $\mathcal{A}_{enc}$ gives its message pair (p_0, p_1) to the ElGamal encryption oracle, and receives the ElGamal encryption enc_{b^*} of one of these messages randomly chosen by the oracle according to $b^* \in_{\mathcal{R}} \{0,1\}$. Note that one of these messages (e.g., p_0) has the correct form as in the real view whereas the other one is random. Now, the adversary constructs all components of registration and fingerprinting views correctly as specified by the protocol. It then includes enc_{b^*} in $recv_{\mathcal{A},0}^{\mathsf{Reg}}$ instead of enc_0, and runs the scenario distinguisher S_D. If the output of S_D indicates the correct scenario then the adversary outputs $b^* = 0$, otherwise $b^* = 1$. The same procedure is done for enc_1.

S_4**:** The received message tuple can now be written as follows:

$$\begin{array}{rclcl} recv_{\mathcal{A},0}^{reg} & = & (h_{1,0}, h_{3,0}, M_{1,0}, M_{2,0}) & = & (g_1^{i_0}, g_3^{i_0}, g_3^{j_0}, g_4^{i'_0 j_0}), \\ recv_{\mathcal{A},b}^{fing} & = & (pk_{text,b}, m'_b, s'_b, sig_{text,b}) & = & (g_4^{i'_b}, g_1^{i_b s_b} g_2^{s_b}, s_b j_b, sig_{text,b}). \end{array}$$

Similarly we can write this for the indices 1 and $\bar{b}$. For the relation to Diffie-Hellman-type problems, where only one generator is given, we abbreviate g_1 as g (confusion with g in the withdrawal protocol should not occur) and let $g_2 = g^{\alpha}$, $g_3 = g^{\beta}$, $g_4 = g^{\gamma}$. Recall that the generators were generated (at system setup) in a way that is random (even if $\mathcal{RC}$ cheats). This is important for the following simulations, where we can use random values in the place of α, β and γ. We now rewrite the entire remaining view of the adversary. The only points where the remaining "buyer" reacts on a message from the adversary are the signatures on the purchase descriptions *text*, which we now write $sig_{text} := sig_{g',i'}(text)$ if g' is the generator and i' the secret key. We include the generators so that one remembers to simulate them, and one final global state $state_{\mathcal{A}}$ of the adversary.

$$\begin{array}{rcl} recv_{\mathcal{A},0}^{reg} & = & (g^{i_0}, g^{\beta i_0}, g^{\beta j_0}, g^{\gamma i'_0 j_0}), \\ recv_{\mathcal{A},1}^{reg} & = & (g^{i_1}, g^{\beta i_1}, g^{\beta j_1}, g^{\gamma i'_1 j_1}), \\ recv_{\mathcal{A},b}^{fing} & = & (g^{\gamma i'_b}, g^{i_b s_b + \alpha s_b}, s_b j_b, sig_{g^{\gamma}, i'_b}(text_b)), \\ recv_{\mathcal{A},\bar{b}}^{fing} & = & (g^{\gamma i'_{\bar{b}}}, g^{i_{\bar{b}} s_{\bar{b}} + \alpha s_{\bar{b}}}, s_{\bar{b}} j_{\bar{b}}, sig_{g^{\gamma}, i'_{\bar{b}}}(text_{\bar{b}})), \\ view_{glob} & = & (g, g^{\alpha}, g^{\beta}, g^{\gamma}, state_{\mathcal{A}}). \end{array}$$

S_5**:** We show that we can replace $g^{\beta i_0}$ in $recv_{\mathcal{A},0}^{reg}$ by an independent random value, and still remain indistinguishable from the previous scenario. Otherwise, the following algorithm would break the Decisional Diffie-Hellman assumption: On input (g, g^x, g^y, u), where u is either g^{xy} or an independent random value, use the unknown values x and y in the place of β and i_0, respectively. More precisely,

1. simulate the above view by choosing b, α, γ, j_0, i'_0, s_0, and i_1, j_1, i'_1, s_1 randomly,

2. compute the values where β and i_0 do not occur as usual (note that this simulator knows b, and thus, in which fingerprinting view i_0 is used),

3. simulate the remaining values, i.e., g^{β}, g^{i_0}, $g^{\beta i_0}$, $g^{\beta j_0}$, $g^{\beta i_1}$, $g^{\beta j_1}$ and $g^{i_0 s_0 + \alpha s_0}$, as g^x, g^y, u, $(g^x)^{j_0}$, $(g^x)^{i_1}$, $(g^x)^{j_1}$ and $(g^y)^{s_0} g^{\alpha s_0}$,

4. run the scenario distinguisher D_S on these views and output the bit which D_S outputs. Note that this bit corresponds to a scenario (here S_3 or S_4), and we let the DDH distinguisher guess $u = g^{xy}$ if S_D's

output corresponds to S_3, and random otherwise. Since the simulated views have the correct form as specified in the protocol, and the scenario distinguisher D_S has non-negligible success probability, then the success probability of the DDH distinguisher will also be non-negligible.

Now, we can obviously fade out this independent random value and proceed in the same way with $g^{\beta i_1}$. Thus, this leaves us with:

$$\begin{aligned}
recv^{reg}_{\mathcal{A},0} &= (g^{i_0}, g^{\beta j_0}, g^{\gamma i'_0 j_0}),\\
recv^{reg}_{\mathcal{A},1} &= (g^{i_1}, g^{\beta j_1}, g^{\gamma i'_1 j_1}),\\
recv^{fing}_{\mathcal{A},b} &= (g^{\gamma i'_b}, g^{i_b s_b + \alpha s_b}, s_b j_b, sig_{g^\gamma, i'_b}(text_b)),\\
recv^{fing}_{\mathcal{A},\bar{b}} &= (g^{\gamma i'_{\bar{b}}}, g^{i_{\bar{b}} s_{\bar{b}} + \alpha s_{\bar{b}}}, s_{\bar{b}} j_{\bar{b}}, sig_{g^\gamma, i'_{\bar{b}}}(text_{\bar{b}})),\\
view_{glob} &= (g, g^\alpha, g^\beta, g^\gamma, state_{\mathcal{A}}).
\end{aligned}$$

S_6**:** Next, we show that one can replace $g^{i_b s_b}$ in $recv^{fing}_{\mathcal{A},b}$ by an independent random value u, and still remain indistinguishable from the previous scenario. Otherwise, we can construct an algorithm which breaks *Decisional Strong Diffie-Hellman* (DSDH) Assumption (see Sections 2.2 and 2.6).

First, note that the distribution remains unchanged if we use a random value s'_b in the place of $s_b j_b$, and set $j_b = s'_b/s_b$ instead of choosing it randomly (for $b = 0, 1$). We can, therefore, rewrite the received message tuple as

$$\begin{aligned}
recv^{reg}_{\mathcal{A},0} &= (g^{i_0}, g^{\beta s'_0/s_0}, g^{\gamma i'_0 s'_0/s_0}),\\
recv^{reg}_{\mathcal{A},1} &= (g^{i_1}, g^{\beta s'_1/s_1}, g^{\gamma i'_1 s'_1/s_1}),\\
recv^{fing}_{\mathcal{A},b} &= (g^{\gamma i'_b}, g^{i_b s_b + \alpha s_b}, s'_b, sig_{g^\gamma, i'_b}(text_b))\\
recv^{fing}_{\mathcal{A},\bar{b}} &= (g^{\gamma i'_{\bar{b}}}, g^{i_{\bar{b}} s_{\bar{b}} + \alpha s_{\bar{b}}}, s'_{\bar{b}}, sig_{g^\gamma, i'_{\bar{b}}}(text_{\bar{b}}))\\
view_{glob} &= (g, g^\alpha, g^\beta, g^\gamma, state_{\mathcal{A}}).
\end{aligned}$$

and similarly for the indices 1 and $\bar{b}$.

On input of DSDH-tuples $(g, g^x, g^{1/y}, g^y, u)$ where u is either g^{xy} or a random value, we simulate a view by letting the unknown values x, y play the roles of i_b, s_b, similar as above:

1. Randomly choose b, α, β, γ, j_0, i'_0, s'_0, j_1, i'_1, s'_1 and $i_{\bar{b}}, s_{\bar{b}}$,
2. Construct all corresponding values (the given $g^{y^{-1}}$ corresponds to g^{1/s_b}). Now, as in the previous case, we run the scenario distinguisher D_S on these views, and the SDDH distinguisher outputs the bit which D_S outputs. Again, here we can interpret the bit accordingly, i.e., if D_S indicates the previous scenario then SDDH distinguisher outputs

$u = g^{xy}$, and random otherwise. Since the simulated views has the correct form as specified in the protocol, and the scenario distinguisher D_S has non-negligible success probability, then the success probability of the SDDH distinguisher will also be non-negligible.

Note that the second component in $recv^{fing}_{\mathcal{A},b}$ is now $ug^{\alpha s_b}$ which is a random value independent form all other values since u is a randomly and independently chosen value. Thus, we can fade out this value as well.

Then we go through the same procedure for $g^{i_{\bar{b}} s_{\bar{b}}}$. Thus, we have now the following views:

$$\begin{aligned} recv^{reg}_{\mathcal{A},0} &= (g^{i_0}, g^{\beta s'_0/s_0}, g^{\gamma i'_0 s'_0/s_0}),\\ recv^{reg}_{\mathcal{A},1} &= (g^{i_1}, g^{\beta s'_1/s_1}, g^{\gamma i'_1 s'_1/s_1}),\\ recv^{fing}_{\mathcal{A},b} &= (g^{\gamma i'_b}, s'_b, sig_{g^\gamma, i'_b}(text_b))\\ recv^{fing}_{\mathcal{A},\bar{b}} &= (g^{\gamma i'_{\bar{b}}}, sig_{g^\gamma, i'_{\bar{b}}}(text_{\bar{b}}))\\ view_{glob} &= (g, g^\alpha, g^\beta, g^\gamma, state_{\mathcal{A}}). \end{aligned}$$

S_7**:** Now, g^α is the only remaining component with α, and α is random. Hence, we can also fade out this value. Similarly, we can omit g^{i_b} and $g^{i_{\bar{b}}}$. This leaves us with

$$\begin{aligned} recv^{reg}_{\mathcal{A},0} &= (g^{\beta s'_0/s_0}, g^{\gamma i'_0 s'_0/s_0}),\\ recv^{reg}_{\mathcal{A},1} &= (g^{\beta s'_1/s_1}, g^{\gamma i'_1 s'_1/s_1}),\\ recv^{fing}_{\mathcal{A},b} &= (g^{\gamma i'_b}, s'_b, sig_{g^\gamma, i'_b}(text_b)),\\ recv^{fing}_{\mathcal{A},\bar{b}} &= (g^{\gamma i'_{\bar{b}}}, s'_{\bar{b}}, sig_{g^\gamma, i'_{\bar{b}}}(text_{\bar{b}})),\\ view_{glob} &= (g, g^\beta, g^\gamma, state_{\mathcal{A}}). \end{aligned}$$

Using the same technique as above, we now replace g^{β/s_0} by an independent random value u: We use the DDH assumption and let x and y play the roles of β and $1/s_0$ (Note that $1/s_0$ modulo q is also a random number.) Again, we can correctly simulate all components, and they would have correct distribution in the case $u = g^{xy}$. With random u, we get a component $u^{s'_0}$, which is random, and we can fade it out (Note that any $u \neq 1$ is a generator.) The same holds for g^{β/s_1}. Then g^β is the only remaining component with random β, and we can fade it out.

Thus, the received message tuple can be written as follows:

$$\begin{aligned} recv^{reg}_{\mathcal{A},0} &= (g^{\gamma i'_0 s'_0/s_0}), \\ recv^{reg}_{\mathcal{A},1} &= (g^{\gamma i'_1 s'_1/s_1}), \\ recv^{fing}_{\mathcal{A},b} &= (g^{\gamma i'_b}, s'_b, sig_{g^\gamma, i'_b}(text_b)), \\ recv^{fing}_{\mathcal{A},\bar{b}} &= (g^{\gamma i'_{\bar{b}}}, s'_{\bar{b}}, sig_{g^\gamma, i'_{\bar{b}}}(text_{\bar{b}})), \\ view_{glob} &= (g, g^\gamma, state_{\mathcal{A}}). \end{aligned}$$

S_8**:** Now, $s_b \in_{\mathcal{R}} \mathbb{Z}_q^*$ is a randomly and uniformly chosen value and unrelated to any of the other components. It follows that the components $g^{\gamma i'_b s'_b/s_b}$ in registration tuples are also randomly and uniformly distributed (one-time pad operation by the random value $1/s_b \in \mathbb{Z}_q^*$ in exponents) and unrelated to the fingerprinting components. Thus, we can also fade them out which leaves us with

$$\begin{aligned} recv^{fing}_{\mathcal{A},b} &= (g^{\gamma i'_b}, s'_b, sig_{g^\gamma, i'_b}(text_b)), \\ recv^{fing}_{\mathcal{A},\bar{b}} &= (g^{\gamma i'_{\bar{b}}}, s'_{\bar{b}}, sig_{g^\gamma, i'_{\bar{b}}}(text_{\bar{b}})), \\ view_{glob} &= (g, g^\gamma, state_{\mathcal{A}}), \end{aligned}$$

This means that the remaining "buyer" sends only random independent values in the registration unrelated to the fingerprintings. Her behaviour can be simulated without even choosing a value b since it is only an index now. Thus, the views are independent from the bit b, and no adversary can link them. It follows from transitivity of indistinguishability that also no successful distinguisher of the real world views exists.

■

Finally, the following theorem follows from the lemmas 4.1, 4.2, 4.3 and 4.4:

Theorem 4.3 *The construction from Section 4.7.5 is a secure anonymous fingerprinting scheme.* □

Chapter 5

Embedding for Anonymous Fingerprinting

This chapter presents in details the first explicit construction for the embedding protocol of a fingerprinting scheme which is collusion-tolerant, asymmetric, and anonymous.

5.1 Introduction

We present the construction for the embedding and extracting procedures of our fingerprinting scheme. We require the scheme to be collusion-tolerant, asymmetric and anonymous. In particular, the construction of such a scheme must guarantee that in any redistributed work, generated by a collusion of a given tolerable size, the merchant can find the information which comes from at least one traitor. In other words, after finding a redistributed work and extracting a codeword from it, the merchant is faced with the problem of error-correction with probably too many errors.

Let us take a look at the constructions we had so far, and briefly discuss whether they can offer a solution to this problem: In the symmetric collusion-tolerant scheme, described in Section 4.4, the merchant prepares a list of the codewords he has already assigned to his buyers. In identification, he uses this list and the word extracted to identify a traitor. However, the usage of such codes is restricted, since for embedding and extracting large amount of information the number of codewords, which the merchant has to keep track of, might get very large.

The next construction is the asymmetric fingerprinting scheme (with 3-party trial) described in Section 4.5. In this construction, the merchant knows half of the codeword and the commitments on the other half. He then prepares a list of these halfwords which he uses in the identification together with the extracted codeword. However, to obtain a complete codeword, which he requires for a proof in a trial, the buyer must be involved in the trial, and asked by the arbiter to open her commitments (therefore, 3-party trial). Hence, the evidences the merchant can retrieve are not sufficient for an arbiter to decide that an accused buyer is guilty. The reason is that the extracted codeword has too many errors to be reasonably decoded. In particular, this construction is not appropriate for an anonymous fingerprinting scheme since $\mathcal{M}$ does not know the identity of his buyers.

As mentioned before, we need a construction which guarantees that at least for one of the traitors *all* of his committed bits can be extracted from the redistributed work. To handle this, Pfitzmann and Waidner (1997c) propose a construction framework for a collusion-tolerant asymmetric fingerprinting scheme with 2-party trial. They propose to combine an asymmetric fingerprinting scheme, based on concatenated codes, with Error-and-Erasure-Correcting Codes (EECC) such that the extracted bits come from the *same* traitor.

The embedding protocol in Pfitzmann and Waidner (1997c) is partially explicit. In the following sections, we will focus on this construction, and give explicit protocols for its realization, and analyze its security.

5.2 Building Blocks

In this section, we consider those primitives and cryptographic protocols which will serve as building blocks for the embedding procedure of our fingerprinting protocol.

5.2.1 Reed-Solomon Code

A **Reed-Solomon** (RS) Code (Reed and Solomon 1960) is an error-correcting **block code**. A block code of length L containing N elements over the alphabet Σ is a set of N L-tuples where each L-tuple takes its components sym_i from Σ. We denote a code with $\mathcal{C}$. An element of $\mathcal{C}$ is called a codeword denoted by vector notation $\vec{word} := (sym_1, \cdots, sym_L)$.

Block codes are specified by their main parameters L, N and d^* where d^* is the **minimum distance** of the code defined as the **Hamming distance** $\mathsf{HD}(\vec{word}_1, \vec{word}_2)$ between two distinct codewords $\vec{word}_1, \vec{word}_2 \in \mathcal{C}$, i.e., $d^* := \min \mathsf{HD}(\vec{word}_1, \vec{word}_2)$. Another important parameter is the **dimension** *dim* or *information length* of a code $\mathcal{C}$. It is defined as $dim := \log_{|\Sigma|} N$ (logarithm to the base $|\Sigma|$).

Reed-Solomon codes belong to one of the most important classes of **linear codes**, namely, the **cyclic codes**. A code $\mathcal{C}$ over a field $\mathbb{F}_q$ is called *linear* if $\mathcal{C}$ is a linear subspace of $\mathbb{F}_q^L$ over $\mathbb{F}_q$, namely, for all codewords

$\vec{word}_1, \vec{word}_2 \in \mathcal{C}$ and the values $a_1, a_2 \in \mathbb{F}_q$ we have $a_1\vec{word}_1 + a_2\vec{word}_2 \in \mathcal{C}$. A linear code $\mathcal{C}$ is called cyclic code if $\mathcal{C}$ is a cyclic subspace. A subspace S of a vector space $\mathbb{F}_q^L$ over $\mathbb{F}_q$ is cyclic if whenever $(a_1, a_2, \cdots, a_L) \in S$ then $(a_L, a_1, \cdots, a_{L-1}) \in S$.

The dimension of a linear (L, N, d^*)-code $\mathcal{C}$ is the dimension of the subspace $\mathcal{C}$ of $\mathbb{F}_q^L$ over $\mathbb{F}_q$. If dim is the dimension of $\mathcal{C}$ then we say that $\mathcal{C}$ is a linear $[L, dim, d^*]$-code over $\mathbb{F}_q$. The difference $L - dim$ is called the **redundancy** of $\mathcal{C}$.

The length L of an RS-code over $\mathbb{F}_q$ is $L = q - 1$. Thus, the length is the number of nonzero elements in the ground field. The message m to be RS-encoded is represented as a polynomial $m(x) := \sum_{s=0}^{l-1} m_s x^s$ over the field $\mathbb{F}_q$ with $m_s \in \mathbb{F}_q$. We denote the RS-encoding with $\vec{h} \leftarrow \mathsf{RS}(\vec{m}, par_{\mathsf{RS}})$ where par_{RS} denotes the required parameters. Here, we use the vector notation for $m(x)$ and its RS-encoding $h(x)$, i.e., we write $\vec{m} := (m_0, \cdots, m_{l-1})$, and $\vec{h} := (h_0, \cdots, h_{L-1})$. Note that the field operations are operation on polynomials over the underlying field.

For any $T < L/2$ there exists an RS-code of minimum distance $d^* = 2T + 1$ and the dimension $dim = L - 2T$, i.e., the code can correct up to T errors. An RS-code can be constructed as follows: Let α be a generator of $\mathbb{F}_q$. The so-called **generator polynomial** for this code is $g(x) = \prod_{i=1}^{2T}(x - \alpha^i)$. The code can then be generated by multiplying $g(x)$ with message polynomials $m(x)$ of degree less than $L - 2T$ over $\mathbb{F}_q$. Basically, codes with $d^* = 2T + 1$ are used to correct up to T errors. However, one can use them to correct T_1 errors and T_2 erasures[1] where $2T_1 + T_2 + 1 \leq d^*$ holds.

For more detailed information on error-correcting codes see Vanstone and van Oorschot (1995) or van Lint (1999).

5.2.2 Proving Equality of QR-Committed Encodings

We consider a protocol $\mathsf{ProveEqEncCom}()$ in which a prover $\mathcal{P}$ proves to a verifier $\mathcal{V}$ that the contents of two given QR-commitments are different encodings of the same value. The encodings we are concerned with are binary $\mathsf{bin}()$ (here dyadic) and unary $\mathsf{un}()$ encodings (see Section 3.1).

Description: Let $(\mathsf{GenParComQR}(), \mathsf{ProtComQR}(), \mathsf{ProtOpenQR}())$ be a QR commitment scheme with the parameter $par_{com}^{\mathsf{QR}} = n$. Let C_a^{bin} and $C_{a^*}^{\mathsf{un}}$ be given commitments that are supposed to be the QR-commitments to the binary and unary encoding of the same value $a \in \Sigma := \{0, 1, \cdots, m-1\} \subset \mathbb{N}$,

[1]An erasure can be viewed as an error for which the position is known but the value (magnitude) not. The task of the decoder is here to restore or fill the erasure positions.

i.e., these commitments have the following form:

$$\begin{aligned} C_a^{\mathsf{bin}} &:= com^{\mathsf{QR}}(\mathsf{bin}(a), \vec{x}, n) = \left[\, com^{\mathsf{QR}}(a_i^{\mathsf{bin}}, x_i, n)\,\right]_{1\le i\le l} \\ C_{a^*}^{\mathsf{un}} &:= com^{\mathsf{QR}}(\mathsf{un}(a^*), \vec{y}, n) = \left[\, com^{\mathsf{QR}}(a_j^{*\mathsf{un}}, y_j, n)\,\right]_{1\le j\le l'} \end{aligned}$$

where a and a^* denote their contents, and $(a_1^{\mathsf{bin}}, a_2^{\mathsf{bin}}, \cdots, a_l^{\mathsf{bin}}) := \mathsf{bin}(a)$, $(a_1^{*\mathsf{un}}, a_2^{*\mathsf{un}} \cdots, a_{l'}^{*\mathsf{un}}) := \mathsf{un}(a^*)$ denote the binary and unary encoding with $a_i^{\mathsf{bin}} \in \{0,1\}$ for $1 \le i \le l$ and $a_j^{*\mathsf{un}} := 1_{j=a^*} := \begin{cases} 1 & : \text{ if } j = a^* \\ 0 & : \text{ else} \end{cases}$ for $1 \le j \le l'$, and $x_i, y_j \in_{\mathcal{R}} \mathbb{Z}_n^*$.

Remark 5.1. The required QR-commitment parameter $par_{com}^{\mathsf{QR}} = n$ is obtained via a trusted parameter generation. This means, depending on the requirements and the application, the required parameters can be generated either by a trusted third party or by a secure and verifiable procedure. In the second approach, one party (here the prover $\mathcal{P}$) generates the parameter par_{com}^{QR} and proves its correctness to the other (here the verifier $\mathcal{V}$). See Section 3.5.4 for details. ○

In protocol $\mathsf{ProveEqEncCom}()$, the prover $\mathcal{P}$ proves to the verifier $\mathcal{V}$ that the contents of C_a^{bin} and $C_{a^*}^{\mathsf{un}}$ are the binary and unary encodings of the same symbol, i.e., $a = a^*$.

This protocol is denoted by

$$(\mathcal{P} : -;\ \mathcal{V} : ind) \leftarrow \mathsf{ProveEqEncCom}(\mathcal{P} : state_{\mathcal{P}}^{C};\ \mathcal{V} : -;\ * : C, par_{\mathsf{com}}^{\mathsf{QR}})$$

where $state_{\mathcal{P}}^{C} := (state_{\mathcal{P}}^{C_a^{\mathsf{bin}}}, state_{\mathcal{P}}^{C_{a^*}^{\mathsf{un}}})$ is the opening information for the commitments $C := (C_a^{\mathsf{bin}}, C_{a^*}^{\mathsf{un}})$ and $ind \in \{accept, reject\}$ is the output to the verifier indicating whether it accepts or not.

The underlying language L, describing the objective to be proven, is defined as follows:

$$\begin{aligned} L := \Big\{ & (C_a^{\mathsf{bin}}, C_{a^*}^{\mathsf{un}}, n, k, l, l') | \\ & n \in [\mathsf{GenParComQR}(k)] \wedge C_a^{\mathsf{bin}} \in (\mathbb{Z}_n^{(+1)})^l \wedge C_{a^*}^{\mathsf{un}} \in (\mathbb{Z}_n^{(+1)})^{l'} \wedge \\ & \exists a, a^* \in \Sigma \wedge \exists \vec{x} \in (\mathbb{Z}_n^*)^l \wedge \exists \vec{y} \in (\mathbb{Z}_n^*)^{l'} \wedge \\ & C_a^{\mathsf{bin}} = com^{\mathsf{QR}}(\mathsf{bin}(a), \vec{x}, n) \wedge C_{a^*}^{\mathsf{un}} = com^{\mathsf{QR}}(\mathsf{un}(a^*), \vec{y}, n) \wedge \\ & a = a^* \Big\}. \end{aligned}$$

Construction: The generating algorithm for this language, denoted by $\mathsf{GenEqEncCom}()$, outputs $\big((C_a^{\mathsf{bin}}, C_{a^*}^{\mathsf{un}}, n, k, l, l'), (\vec{x}, \vec{y})\big)$ (see also 3.6.2).

The individual protocol steps are as follows:

1. $\mathcal{P}$ encodes all symbols $d_s \in \Sigma$ $(0 \leq s \leq m-1)$ using both encoding functions, commits to them, and constructs the table T shown in Table 5.1. For the sake of completeness we use different symbols d_s and d'_s for each column.

Table 5.1 Table T: Commitments to symbols of Σ in unary and binary form

d_s	$com^{\mathsf{QR}}(\mathsf{bin}(d_s))$	$com^{\mathsf{QR}}(\mathsf{un}(d'_s))$
0	$com^{\mathsf{QR}}(d^{\mathsf{bin}}_{0,1}), \cdots, com^{\mathsf{QR}}(d^{\mathsf{bin}}_{0,l})$	$com^{\mathsf{QR}}({d'}^{\mathsf{un}}_{0,1}), \cdots, com^{\mathsf{QR}}({d'}^{\mathsf{un}}_{0,l'})$
1	$com^{\mathsf{QR}}(d^{\mathsf{bin}}_{1,1}), \cdots, com^{\mathsf{QR}}(d^{\mathsf{bin}}_{1,l})$	$com^{\mathsf{QR}}({d'}^{\mathsf{un}}_{1,1}), \cdots, com^{\mathsf{QR}}({d'}^{\mathsf{un}}_{1,l'})$
$\vdots$	$\vdots$	$\vdots$
$m-1$	$com^{\mathsf{QR}}(d^{\mathsf{bin}}_{m-1,1}), \cdots, com^{\mathsf{QR}}({d'}^{\mathsf{bin}}_{m-1,l})$	$com^{\mathsf{QR}}({d'}^{\mathsf{un}}_{m-1,1}), \cdots, com^{\mathsf{QR}}({d'}^{\mathsf{un}}_{m-1,l'})$

We denote the binary and unary commitments at row s with

$$\begin{aligned} C^{\mathsf{bin}}_s &:= com^{\mathsf{QR}}\big(\mathsf{bin}(d_s), \vec{x}_s, n\big) = \big[\, com^{\mathsf{QR}}(d^{\mathsf{bin}}_{s,i}, x_{s,i}, n) \,\big]_{1\leq i\leq l} \\ C^{\mathsf{un}}_s &:= com^{\mathsf{QR}}\big(\mathsf{un}(d'_s), \vec{y}_s, n\big) = \big[\, com^{\mathsf{QR}}({d'}^{\mathsf{un}}_{s,j}, y_{s,j}, n) \,\big]_{1\leq j\leq l'} \end{aligned}$$

for $s \in [0, m-1]$. Note that, for simplicity, the randomizing components $\vec{x}_s, \vec{y}_s$ and the modulo n of the QR-commitments are omitted in the table.

2. $\mathcal{P}$ selects a random permutation σ of Σ and permutes the rows of table T by applying σ. $\mathcal{P}$ sends the resulting table T_σ to $\mathcal{V}$.

3. $\mathcal{V}$ responds with a (randomly chosen) binary challenge c.

4. If $c = 0$, then $\mathcal{P}$ sends $resp_0 := (\sigma, state^{C_s}_{\mathcal{P}})$ for $s \in [0, m-1]$ to $\mathcal{V}$ where $state^{C_s}_{\mathcal{P}} := (state^{C^{\mathsf{bin}}_s}_{\mathcal{P}}, state^{C^{\mathsf{un}}_s}_{\mathcal{P}})$, $state^{C^{\mathsf{bin}}_s}_{\mathcal{P}} := (d^{\mathsf{bin}}_s, \vec{x}_s)$, $state^{C^{\mathsf{un}}_s}_{\mathcal{P}} := ({d'}^{\mathsf{un}}_s, \vec{y}_s)$, $state^{C^{\mathsf{bin}}_s}_{\mathcal{V}} := (C^{\mathsf{bin}}_s, n)$ and $state^{C^{\mathsf{un}}_s}_{\mathcal{V}} := (C^{\mathsf{un}}_s, n)$.

   Now, $\mathcal{V}$ proceeds as follows:

   (a) Computes $T = \sigma^{-1}(T_\sigma)$,

   (b) Verifies for each $s \in [0, m-1]$ that row s of T contains the correct commitments to the binary and unary encodings of the elements $d_s, d'_s \in \Sigma$:

$$\begin{aligned} true &\overset{?}{=} \mathsf{VerOpenQR}(state^{C^{\mathsf{bin}}_s}_{\mathcal{P}}, state^{C^{\mathsf{bin}}_s}_{\mathcal{V}}) \wedge \\ true &\overset{?}{=} \mathsf{VerOpenQR}(state^{C^{\mathsf{un}}_s}_{\mathcal{P}}, state^{C^{\mathsf{un}}_s}_{\mathcal{V}}) \end{aligned}$$

(c) For each s if the opening algorithm returns *true* verifies that

$$d_s = \mathsf{bin}^{-1}(d_s^{\mathsf{bin}}) = \mathsf{un}^{-1}(d'^{\mathsf{un}}_s).$$

Note that this check is needed since the verification algorithm $\mathsf{VerOpenQR}()$ only checks whether the opening information is correct (see also 3.5.4). It does not verify that the content of the commitments are equal.

5. If $c = 1$ then $\mathcal{P}$ sends $resp_1 := (s', state_{\mathcal{P}}^{C_{s',*}})$. Here s' is the permuted row which is supposed to contain the encodings of a and a^*. Further, $state_{\mathcal{P}}^{C_{s',*}} := (state_{\mathcal{P}}^{C^{\mathsf{bin}}_{s',*}}, state_{\mathcal{P}}^{C^{\mathsf{un}}_{s',*}})$ is supposed to be the opening information of the commitments $C^{\mathsf{bin}}_{s',*}$ and $C^{\mathsf{un}}_{s',*}$ (which are supposed to be $C^{\mathsf{bin}}_{s',*} := C^{\mathsf{bin}}_{s'} C^{\mathsf{bin}}_{a}$, $C^{\mathsf{un}}_{s',*} := C^{\mathsf{un}}_{s'} C^{\mathsf{un}}_{a^*}$.)

   Now, $\mathcal{V}$ proceeds as follows:

   (a) Computes $C'^{\mathsf{bin}}_{s',*} := C^{\mathsf{bin}}_{s'} C^{\mathsf{bin}}_{a}$ and $C'^{\mathsf{un}}_{s',*} := C^{\mathsf{un}}_{s'} C^{\mathsf{un}}_{a^*}$.

   (b) Verifies that

$$\begin{array}{rcl} true & = & \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C^{\mathsf{bin}}_{s',*}}, state_{\mathcal{V}}^{C'^{\mathsf{bin}}_{s',*}}) \wedge \\ true & = & \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C^{\mathsf{un}}_{s',*}}, state_{\mathcal{V}}^{C'^{\mathsf{un}}_{s',*}}). \end{array}$$

   where $state_{\mathcal{V}}^{C'^{\mathsf{bin}}_{s',*}} := (C'^{\mathsf{bin}}_{s',*}, n)$ and $state_{\mathcal{V}}^{C'^{\mathsf{un}}_{s',*}} := (C'^{\mathsf{un}}_{s',*}, n)$.

   (c) Verifies that the quantities $C'^{\mathsf{bin}}_{s',*}$ and $C'^{\mathsf{un}}_{s',*}$ are commitments to zero by checking the corresponding opening information $(state_{\mathcal{P}}^{C^{\mathsf{bin}}_{s',*}}, state_{\mathcal{P}}^{C^{\mathsf{un}}_{s',*}})$ sent by $\mathcal{P}$.

Lemma 5.1 *The protocol* $\mathsf{ProveEqEncCom}()$ *is complete.* □

Proof. For this property, it is assumed that both parties are honest, and behave according to the specified protocol. Thus, we can assume that $\mathcal{P}$ correctly computes the table T (and T_σ). It is easy to see that $\mathcal{V}$ accepts for $c = 0$. To see this for $c = 1$, consider the following relations

$$\begin{array}{rcl} C'^{\mathsf{bin}}_{s',*} & = & C^{\mathsf{bin}}_{a} C^{\mathsf{bin}}_{s'} \\ & = & \left[\, com^{\mathsf{QR}}(a_i^{\mathsf{bin}}, x_i, n) \,\right] \left[\, com^{\mathsf{QR}}(d^{\mathsf{bin}}_{s',i}, x_{s',i}, n) \,\right]_{1 \le i \le l} \\ & = & \left[\, com^{\mathsf{QR}}(a_i^{\mathsf{bin}} \oplus d^{\mathsf{bin}}_{s',i}, x_i x_{s',i}, n) \,\right]_{1 \le i \le l} \\ & = & \left[\, com^{\mathsf{QR}}(a_i^{\mathsf{bin}} \oplus d^{\mathsf{bin}}_{s',i}, v_i, n) \,\right]_{1 \le i \le l} \end{array}$$

and

$$\begin{aligned} C'^{\mathsf{un}}_{s',*} &= C^{\mathsf{un}}_{a} C^{\mathsf{un}}_{s'} \\ &= \left[\ com^{\mathsf{QR}}(a^{\mathsf{un}}_j, y_j, n)\ \right] \left[\ com^{\mathsf{QR}}(d'^{\mathsf{un}}_{s',j}, y_{s',j}, n)\ \right]_{1 \le j \le l'} \\ &= \left[\ com^{\mathsf{QR}}(a^{\mathsf{un}}_j \oplus d'^{\mathsf{un}}_{s',j}, y_j y_{s',j}, n)\ \right]_{1 \le j \le l'} \\ &= \left[\ com^{\mathsf{QR}}(a^{\mathsf{un}}_j \oplus d'^{\mathsf{un}}_{s',j}, w_j, n)\ \right]_{1 \le j \le l'} \end{aligned}$$

where $v_i := x_i x_{s',i}$ and $w_j := y_j y_{s',j}$ (Note that $\vec{v} := \left[\ x_i x_{s',i}\ \right]_{1 \le i \le l}$ and $\vec{w} := \left[\ y_j y_{s',j}\ \right]_{1 \le j \le l'}$).

Consider now the opening of $C^{\mathsf{bin}}_{s',*}$: Since $d_{s'} = a$, and thus, $d^{\mathsf{bin}}_{s',i} = a^{\mathsf{bin}}_i$, it follows from homomorphic properties of the QR-commitment

$$\left[\ a^{\mathsf{bin}}_i \oplus d^{\mathsf{bin}}_{s',i}\ \right]_{1 \le i \le l} = \left[\ 0\ \right]_{1 \le i \le l}.$$

The same applies to $C^{\mathsf{un}}_{s',*}$, i.e., $\left[\ a^{*\mathsf{un}}_j \oplus d'^{\mathsf{un}}_{s',j}\ \right]_{1 \le i \le l'} = \left[\ 0\ \right]_{1 \le j \le l'}$. ■

Lemma 5.2 *The protocol* ProveEqEncCom() *is perfectly sound.* □

Proof. If $(C^{\mathsf{bin}}_a, C^{\mathsf{un}}_{a^*}, n, k, l, l') \notin L$, we show that even a computationally unrestricted prover cannot convince the verifier with probability better than 1/2.

Assume, we are given $(C^{\mathsf{bin}}_a, C^{\mathsf{un}}_{a^*}, n, k, l, l') \notin L$ (i.e., in particular $a \neq a^*$) and an arbitrary prover $\mathcal{P}^*$ with an arbitrary auxiliary input $aux_{\mathcal{P}^*}$. We show that $\mathcal{P}^*$ can at most pass one of the $\mathcal{V}$'s challenges.

Suppose, $\mathcal{V}$ has accepted the response $resp_0$, i.e., it verifies that each row of the table T_σ is correctly constructed. We show that $\mathcal{P}^*$ cannot reveal a response $resp_1$ which $\mathcal{V}$ would accept. Now, consider the case $c = 1$: $\mathcal{V}$'s verification of $resp_1$ would be successful if and only if $(C'^{\mathsf{bin}}_{s',*}, C'^{\mathsf{un}}_{s',*}, n, k, l, l') \in L$, and $C'^{\mathsf{bin}}_{s',*}, C'^{\mathsf{un}}_{s',*}$ are commitments to zero. A successful verification implies that there exist $d_{s'}, d'_{s'} \in \Sigma$ such that $d_{s'} = a$ and $d'_{s'} = a^*$. Since we assumed that $d_{s'} = d'_{s'}$ holds, it follows that $a = a^*$ which is a contradiction. Thus, $\mathcal{P}^*$ cannot correctly open the resulting commitments, and consequently $\mathcal{V}$ will not accept the response in this case, unless $\mathcal{P}^*$ is capable of opening the QR-commitments $C^{\mathsf{bin}}_{s'}, C^{\mathsf{un}}_{s'}$ in two different ways. This is, however, not possible even for all powerful $\mathcal{P}^*$ since QR commitment scheme is information-theoretically binding.

Conversely, consider the case where $\mathcal{V}$ has accepted the response $resp_1$. Thus, there exist $d_{s'}, d'_{s'} \in \Sigma$ such that $d_{s'} = a$ and $d'_{s'} = a^*$. Since we assumed that $a \neq a^*$ it follows that $d_{s'} \neq d'_{s'}$ and thus, row s' in the table T_σ is not correctly constructed[2]. However, this would mean that $\mathcal{V}$ would

[2]According to the protocol, each row s of T must contain the commitments to a *different* element d_s of Σ (which $\mathcal{V}$ additionally checks) However, for the soundness property it suffices that the table contains, at least in one row, the (binary and unary) commitments to a. The specified form of the table T is necessary for the zero-knowledge property.

not accept $resp_0$ unless $\mathcal{P}^*$ is capable of opening QR-commitments $C_{s'}^{\mathsf{bin}}, C_{s'}^{\mathsf{un}}$ in two different ways. Again, this is not possible even for all powerful $\mathcal{P}^*$ since QR commitment scheme is information-theoretically binding.

Hence, after sending the table T_σ, $\mathcal{P}^*$ can correctly answer to at most one of the two possible challenges, and since $\mathcal{P}^*$ does not know the challenge in advance, its success probability is at most 1/2.

■

Lemma 5.3 *The protocol* ProveEqEncCom() *is perfect zero-knowledge.* □

Proof. We construct a simulator $\mathcal{S}^{\mathcal{V}^*}$ with black-box access to $\mathcal{V}^*$ as given in Algorithm 5 that we will explain in the following. The inputs to $\mathcal{S}^{\mathcal{V}^*}$ are $par_{com}^{\mathsf{QR}} = n$, C_a^{bin} and $C_{a^*}^{\mathsf{un}}$.

Our goal is to show that $\mathcal{S}^{\mathcal{V}^*}$ generates a view with identical distribution as $\mathcal{V}^*$'s view in the real protocol. Thus, we must verify that all quantities computed by the simulator are of correct form:

In Algorithm 5, $\mathcal{S}^{\mathcal{V}^*}$ computes all quantities according to the real protocol except for the construction of the table T in case $c = 1$ (see Step 3). In this case, $\mathcal{S}^{\mathcal{V}^*}$ proceeds as follows:

First, it computes the unary commitments to $a^*+1 \bmod m$, $a^*+2 \bmod m$ and so on. It is not difficult to see that this can be done by rotating the unary commitments $C_{a^*}^{\mathsf{un}}$ once at a time.

Secondly, $\mathcal{S}^{\mathcal{V}^*}$ computes the binary commitments C_s^{bin} by using the unary commitments C_s^{un}. This is done by applying algorithm DeriveEncCom() from Section 5.2.5 (where the corresponding encoding function Encode() is the binary encoding bin().)

Thirdly, the table T is constructed in Step 3c by randomly entering $(C_s^{\mathsf{bin}}, C_s^{\mathsf{un}})$ to any $m-1$ rows of T and $(C_a^{\mathsf{bin}})^{-1}, (C_{a^*}^{\mathsf{un}})^{-1}$ in the remaining row. The reason for entering the inverse commitments in that row is that the simulator does not know the opening values for the commitments $C_a^{\mathsf{bin}}, C_{a^*}^{\mathsf{un}}$, in particular, the randomizing parts $\vec{x}, \vec{y}$. Thus, it will not be able to response with the opening information for the product commitments $C_{s',*}^{\mathsf{bin}}$ and $C_{s',*}^{\mathsf{un}}$ computed by the verifier.

Finally, all rows are blinded in Step 3d to obtain independent commitments which follows from Lemma 3.2 stating that the result of multiplication of a given QR-commitment with a random QR-commitment to zero is again a random commitment to the same value.

Remark 5.2. A trivial solution for constructing the table T is to fill all rows with random values except for one row where the commitments $(C_a^{\mathsf{bin}})^{-1}, (C_{a^*}^{\mathsf{un}})^{-1}$ are entered. However, the resulting protocol will not be perfectly zero-knowledge (that we actually prefer) since the simulated values will not have the same probability distribution as the values in the real protocol (see also Footnote 2.) ○

Algorithm 5 The Simulator $\mathcal{S}^{\mathcal{V}^*}$ for protocol $\mathsf{ProveEqEncCom}()$

Input: $C_a^{\mathsf{bin}}, C_{a^*}^{\mathsf{un}}, n = par_{com}^{\mathsf{QR}}$

1. Choose $c \in_{\mathcal{R}} \{0,1\}$, and a random permutation σ of Σ.

2. If $c = 0$, construct the table T_σ^0 by computing the commitments $C_s^{\mathsf{bin}} := com^{\mathsf{QR}}\big(\mathsf{bin}(d_s), \vec{x}_s, n\big)$ and $C_s^{\mathsf{un}} := com^{\mathsf{QR}}\big(\mathsf{un}(d'_s), \vec{y}_s, n\big)$ where $s \in [0, m-1]$, $\vec{x}_s \in_{\mathcal{R}} (\mathbb{Z}_n^*)^l$ and $\vec{y}_s \in_{\mathcal{R}} (\mathbb{Z}_n^*)^{l'}$.

3. If $c = 1$, compute the table T_σ^1 as follows:

   (a) Use the unary commitment $C_{a^*}^{\mathsf{un}}$ to compute the unary commitments $C_s^{\mathsf{un}} := com^{\mathsf{QR}}\big(\mathsf{un}(d'_s), \vec{y}_s, n\big)$ to $d'_s := a^* + s \bmod m$ with $s \in [0, m-1]$. To add $1 \bmod m$ each time, shift the individual commitments in the commitment tuple $C_{a^*}^{\mathsf{un}} := \big(com^{\mathsf{QR}}(a_1^{*\,\mathsf{un}}, y_1, n), \cdots, com^{\mathsf{QR}}(a_{l'}^{*\,\mathsf{un}}, y_{l'}, n)\big)$ one position to the right, and once for all remaining $m-1$ values (rows).

   (b) Use C_s^{un} to compute the corresponding binary commitments C_s^{bin} for each $d_s \neq a$. For this, use the algorithm $\mathsf{DeriveEncCom}()$ from Section 5.2.5 where the encoding function $\mathsf{Encode}()$ is the binary encoding $\mathsf{bin}()$. According to this algorithm and Lemma 5.10, the commitment to each bit $d_{s,i}^{\mathsf{bin}}$ of d_s can be computed as follows: $com^{\mathsf{QR}}(d_{s,i}^{\mathsf{bin}}, z_i, n) := \prod_{j : d_{j,i}^{\mathsf{bin}} \neq 0^{\mathsf{bin}}} com^{\mathsf{QR}}\big({d'}_{s,j}^{\mathsf{un}}, y_{s,j}, n\big)$ where for each i ($1 \leq i \leq l$) the set $\{j : d_{j,i}^{\mathsf{bin}} \neq 0^{\mathsf{bin}}\}$ indicates all elements $d_j \in \Sigma$ whose binary encodings contain a 1 at their i-th bit position.

   (c) Construct the table T as follows: Enter randomly $(C_s^{\mathsf{bin}}, C_s^{\mathsf{un}})$ to any $m-1$ rows of T. In the remaining row, denoted by s', enter the inverse commitments $(C_a^{\mathsf{bin}})^{-1}, (C_{a^*}^{\mathsf{un}})^{-1}$.

   (d) Blind the binary and unary commitments $(C_s^{\mathsf{bin}}, C_s^{\mathsf{un}})$ at each row s of the table T by multiplying them with commitments to zero, i.e., compute $C_s^{\mathsf{bin}} C_{s,0}^{\mathsf{bin}}$ and $C_s^{\mathsf{un}} C_{s,0}^{\mathsf{un}}$ where $C_{s,0}^{\mathsf{bin}} := com^{\mathsf{QR}}(\mathsf{bin}(0), \vec{x'_s}, n)$, $C_{s,0}^{\mathsf{un}} := com^{\mathsf{QR}}(\mathsf{un}(0), \vec{y'_s}, n)$, $\vec{x'_s} \in_{\mathcal{R}} (\mathbb{Z}_n^*)^l$, and $\vec{y'_s} \in_{\mathcal{R}} (\mathbb{Z}_n^*)^{l'}$.

4. Start the verifier $\mathcal{V}^*$, send T_σ^c to it, and wait until $\mathcal{V}^*$ sends the challenge c^*.

5. IF $c^* = c$, THEN

   (a) send $resp_c$ accordingly: $resp_0$ as in the real protocol and $resp_1 := (s', state_{\mathcal{P}}^{C_{s',0}^{\mathsf{bin}}}, state_{\mathcal{P}}^{C_{s',0}^{\mathsf{un}}})$ with $state_{\mathcal{P}}^{C_{s',0}^{\mathsf{bin}}} := (0^{\mathsf{bin}}, \vec{x'_{s'}}, n)$ and $state_{\mathcal{P}}^{C_{s',0}^{\mathsf{un}}} := (0^{\mathsf{un}}, \vec{y'_{s'}}, n)$.

   (b) wait until $\mathcal{V}^*$ outputs $ind_{\mathcal{V}^*}$. Then output the view $view_{\mathcal{S}^{\mathcal{V}^*}} = (T_\sigma^c, c, resp_c, ind_{\mathcal{V}^*})$.

   ELSE output the symbol $\perp$.

We consider the simulated view $(T_\sigma^c, c, \mathit{resp}_c, \mathit{ind}_{\mathcal{V}^*})$, and compare it to the real view.

- T_σ^c: is a table consisting of (independent and random) unary and binary QR-commitments to all symbols of Σ:

  For $c = 0$, the table T_σ^0 is constructed exactly as in the real protocol.

  For $c = 1$, the table T_σ^1 is by construction a permutation of a table T under a random permutation σ' of Σ (the table is filled randomly) Further, T consists of binary and unary QR-commitments to all symbols in Σ which is conform with the real protocol. For the row with the entry $((C_a^{\mathsf{bin}})^{-1}, (C_{a^*}^{\mathsf{un}})^{-1})$ it follows from Lemma 3.1 that

$$\begin{aligned} C'^{\mathsf{bin}}_a &:= (C_a^{\mathsf{bin}})^{-1} \\ &= \mathit{com}^{\mathsf{QR}}(\mathsf{bin}(a), \vec{u}, n) = \left[\; \mathit{com}^{\mathsf{QR}}(a_i^{\mathsf{bin}}, u_i, n) \;\right]_{1 \le i \le l} \\ C'^{\mathsf{un}}_{a^*} &:= (C_{a^*}^{\mathsf{un}})^{-1} \\ &= \mathit{com}^{\mathsf{QR}}(\mathsf{un}(a^*), \vec{u'}, n) = \left[\; \mathit{com}^{\mathsf{QR}}(a_j^{*\mathsf{un}}, u'_j, n) \;\right]_{1 \le j \le l'} \end{aligned}$$

  where $u_i :\equiv x_i^{-1}, u'_j :\equiv y_j^{-1}$ and $x_i, y_j \in_{\mathcal{R}} \mathbb{Z}_n^*$. Thus, C'^{bin}_a and $C'^{\mathsf{un}}_{a^*}$ are again binary and unary QR-commitments to a and a^* respectively (see also Section 3.5.4).

  The commitments in the table T^1 are random and independent due to the blinding step. This follows from Lemma 3.2 stating that the result of multiplication of a given QR-commitment with a random QR-commitment to zero, is again a random commitment to the same value. More precisely, the randomizing parts are determined as follows: For the binary commitments we have $x'_{s,i} \prod_{\Delta_i} y_{s,j}$ if $s \neq a$, and $x'_{a,i} u_i$ if $s = a$. For the unary commitments $y_{s,j} y'_{s,j}$ if $s \neq a^*$, and $y'_{s,j} u_{s,j}$ if $s = a^*$.

- c^*: The challenge c^*, sent by $\mathcal{V}^*$ has the correct distribution since $\mathcal{S}^{\mathcal{V}^*}$ maintains the distribution of c^* by verifying whether $c^* = c$ and acting accordingly.

- resp_c: By construction resp_c has the same distribution as in the real protocol: In the case $c = 0$ all components of resp_0 have the correct form because $\mathit{state}_{\mathcal{P}}^{C_s^{\mathsf{bin}}} = (d_s, \vec{x}_s)$, $\mathit{state}_{\mathcal{P}}^{C_s^{\mathsf{un}}} = (d'_s, \vec{y}_s)$, σ is a random permutation of Σ, $d_s, d'_s \in \Sigma$ for all s, $n \in [\mathsf{GenParComQR}(k)]$, $\vec{x}_s \in_{\mathcal{R}} (\mathbb{Z}_n^*)^l$, and $\vec{y}_s \in_{\mathcal{R}} (\mathbb{Z}_n^*)^l$. Thus, the view has exactly the same distribution as in the real protocol.

  Also in the case $c = 1$, all components of resp_1 have the correct form: s' is the result of a random permutation, and the quantities $\mathit{state}_{\mathcal{P}}^{C_{s',0}^{\mathsf{bin}}} = (\mathsf{bin}(0), \vec{x'_{s'}})$ and $\mathit{state}_{\mathcal{P}}^{C_{s',0}^{\mathsf{un}}} = (\mathsf{un}(0), \vec{y'_{s'}})$, with $\vec{x'}_{s'} \in_{\mathcal{R}}$

$(\mathbb{Z}_n^*)^l, \vec{y'}_{s'} \in_{\mathcal{R}} (\mathbb{Z}_n^*)^l$ and $n \in [\mathsf{GenParComQR}(k)]$, are the correct opening information for the commitments $C'^{\mathsf{bin}}_{s',*}$ and $C'^{\mathsf{un}}_{s',*}$, as in the real protocol. The reason for this is that the quantities $C'^{\mathsf{bin}}_{s',*}$, $C'^{\mathsf{un}}_{s',*}$ are random commitments to zero. We can see this by considering the verifications of the verifier $\mathcal{V}^*$:

$$\begin{aligned} C'^{\mathsf{bin}}_{s',*} &:= C^{\mathsf{bin}}_a C^{\mathsf{bin}}_{s'} = C^{\mathsf{bin}}_a C^{\mathsf{bin}}_{s',0} (C^{\mathsf{bin}}_a)^{-1} = C^{\mathsf{bin}}_{s',0} \\ C'^{\mathsf{un}}_{s',*} &:= C^{\mathsf{un}}_{a^*} C^{\mathsf{un}}_{s'} = C^{\mathsf{un}}_{a^*} C^{\mathsf{un}}_{s',0} (C^{\mathsf{un}}_{a^*})^{-1} = C^{\mathsf{un}}_{s',0}. \end{aligned}$$

Finally, the probability that the simulator fails is at most 1/2 since the simulated view is taken only if $c^* = c$. The running time of the simulator is polynomial since every step of Algorithm 5 can be performed in polynomial time.[3] ■

From the previous lemmas we conclude the following theorem:

Theorem 5.1 *The protocol* ProveEqEncCom() *is perfect zero-knowledge proof of language membership.* □

Remark 5.3. The construction for the protocol ProveEqEncCom() can also be used for arbitrary encodings (with binary symbols). However, in general, the protocol appears to be computationally zero-knowledge (not perfect zero-knowledge) unless there exists an efficient procedure for deriving the commitments to one encoding from the commitments to the other encoding as we have seen for the unary and binary encoding in the simulator algorithm (see Algorithm 5). Assume now, we are given two arbitrary encodings e_1 and e_2 which encode $z \in G$ to binary strings of length l and l'. $\mathcal{S}$ constructs the table T. Since $\mathcal{S}$ does not know the content a it chooses an arbitrary row s' of T and writes the entry $\left((C^{e_1}_a)^{-1}, (C^{e_2}_{a^*})^{-1}\right)$ in this row. Now, with high probability two rows in T contain commitments to the same symbol a (and a^* respectively). This means that the simulated T has not the same probability distribution as the table in the real protocol. In case of QR-commitments, one may show that if a distinguisher D distinguishes both tables then D can be used to construct a distinguisher distinguishing quadratic residues from non-residues, and this is considered to be hard under *Quadratic Residuosity Assumption* (QRA) (see Section 3.5.4 for QRA). ○

5.2.2.1 Efficiency

We consider the main costs from different perspectives, namely, the number of required commitments, the number of main operations for the involved

[3] All required operations, i.e., shifting commitments, multiplication, computing inverse elements, random selection of elements and membership test in $\mathbb{Z}_n^*$, permutation of the finite set Σ and computing the inverse of the encodings bin() and un() can be performed in polynomial time.

parties, and the number of communicated bits. Note that for different challenges c, we have different costs for different types of operations. We take the challenge with higher costs.

- *Number of commitments*: $m(l+l') = m(\log n + m)$ for computing the table T
- *Number of operations for $\mathcal{P}$*:
  - Multiplication: $l + l' = \log n + m$ multiplications modulo n (in case $c = 1$)
  - Squaring: $m(l+l') = m(\log n + m)$ squarings modulo n (in case $c = 1$)
- *Number of operations for $\mathcal{V}$*:
  - Multiplication: $l + l' = (\log n + m)$ multiplications modulo n (in case $c = 1$)
  - Squaring: $m(l+l') = m(\log n + m)$ squarings modulo n (in case $c = 0$)
- *Communicated bits*: $2m(l+l')\log(n) = 2m\big(\log(n) + m\big)\log(n)$

Note that we have $m(l+l') = O(\mathsf{poly}(k))$ where k is the security parameter.

5.2.3 Proving Correctness of Committed Reed-Solomon Encoding

We consider a protocol $\mathsf{ProveEncRSCom}()$ in which a prover $\mathcal{P}$ proves to a verifier $\mathcal{V}$ that given two QR-commitments, the content of one them is the Reed-Solomon (RS) encoding of the content of the other one over the finite field $\mathbb{F}_{2^e}$.

Description: Let $z \in G$ be the value to be RS-encoded, and a be the binary encoding of z, i.e., $a = \mathsf{bin} \circ \mathsf{int}(z)$. The value z and its RS-encoding are represented as polynomials $\sum_{s=0}^{l'-1} c_s x^s$, and $\sum_{s'=0}^{l''-1} c'_{s'} x^{s'}$ over $\mathbb{F}_{2^e}$ where $c_s, c'_{s'} \in \mathbb{F}_{2^e}$ (see also Section 5.2.1)

Let $f_s := \mathsf{bin} \circ \mathsf{int}(c'_s)$ and $h_{s'} := \mathsf{bin} \circ \mathsf{int}(c'_{s'})$ for $s \in [0, l'-1]$ and $s' \in [0, l''-1]$ where we have $\mathit{length}_2(f_s) = \mathit{length}_2(h_{s'}) = e$. Recall that we use vector convention to denote the above polynomials with their binary encoded coefficients f_s and $h_{s'}$, i.e., we write $\vec{f} := (f_{l'-1}, \cdots, f_0)$ and $\vec{h} := (h_{l''-1}, \cdots, h_0)$ where $\vec{h}$ represents the RS-encoding of $\vec{f}$, i.e., $\vec{h} := \mathsf{RS}(\vec{f})$. Note that since the underlying field is $\mathbb{F}_{2^e}$, we can set $a := \vec{f}$.

Now, let $(\mathsf{GenParComQR}(), \mathsf{ProtComQR}(), \mathsf{ProtOpenQR}())$ be a QR commitment scheme with $\mathit{par}^{\mathsf{QR}}_{\mathit{com}} = n$ (see Remark 5.1 for the generation of n). Further, let $C^{\mathbb{F}}_f$ and $C^{\mathbb{F}}_h$ be given commitments that are supposed to be the commitments to $\vec{f}$ and its RS-encoding $\vec{h}$. They are denoted in general as follows

$$\begin{aligned} C_f^{\mathbb{F}} &:= com^{\mathsf{QR}}(\vec{f}, \vec{x}, n) = [\ com^{\mathsf{QR}}(f_{s,j}, x_{s,j}, n)\ ]_{0 \le s \le l'-1, 1 \le j \le e} \\ C_h^{\mathbb{F}} &:= com^{\mathsf{QR}}(\vec{h}, \vec{y}, n) = [\ com^{\mathsf{QR}}(h_{s',j}, y_{s',j}, n)\ ]_{0 \le s' \le l''-1, 1 \le j \le e} \end{aligned}$$

where $x_{s,j} \in_{\mathcal{R}} \mathbb{Z}_n^*$ and $y_{s',j} \in_{\mathcal{R}} \mathbb{Z}_n^*$.

In the protocol $\mathsf{ProveEncRSCom}()$ the prover proves to $\mathcal{V}$ that the contents of $C_h^{\mathbb{F}}$ is the RS-encoding of the content of $C_f^{\mathbb{F}}$ over $\mathbb{F}_{2^e}$. The protocol is denoted by

$$(\mathcal{P} : -;\ \mathcal{V} : ind) \leftarrow \mathsf{ProveEncRSCom}(\mathcal{P} : state_{\mathcal{P}}^{C};\ \mathcal{V} : -;\ * : C, par_{\mathsf{com}}^{\mathsf{QR}})$$

where $state_{\mathcal{P}}^{C} := (state_{\mathcal{P}}^{C_f^{\mathbb{F}}}, state_{\mathcal{P}}^{C_h^{\mathbb{F}}})$ denotes the opening information for the commitments $C := (C_f^{\mathbb{F}}, C_h^{\mathbb{F}})$ and $ind \in \{accept, reject\}$ is the output to the verifier indicating whether it accepts or not.

The underlying language L describing the objective to be proven is defined as follows:

$$\begin{aligned} L := \{ & C_f^{\mathbb{F}}, C_h^{\mathbb{F}}, n, l', l'', k, e) | \\ & n \in [\mathsf{GenParComQR}(k)] \wedge C_f^{\mathbb{F}} \in (\mathbb{Z}_n^*)^{l'e} \wedge C_h^{\mathbb{F}} \in (\mathbb{Z}_n^*)^{l''e} \wedge \\ & \exists \vec{f} \in \mathbb{F}_{2^e}^{l'} \wedge \exists \vec{h} \in \mathbb{F}_{2^e}^{l''} \wedge \exists \vec{x} \in (\mathbb{Z}_n^*)^{l'e} \wedge \exists \vec{y} \in (\mathbb{Z}_n^*)^{l''e} \wedge \\ & C_f^{\mathbb{F}} = com^{\mathsf{QR}}(\vec{f}, \vec{x}, n) \wedge C_h^{\mathbb{F}} = com^{\mathsf{QR}}(\vec{h}, \vec{y}, n) \wedge \\ & \vec{h} = \mathsf{RS}(\vec{f}) \}. \end{aligned}$$

Construction: The generation algorithm for this language, denoted by $\mathsf{GenEncRSCom}()$, outputs $\big((C_f^{\mathbb{F}}, C_h^{\mathbb{F}}, n, l', l'', k, e), (\vec{x}, \vec{y})\big)$. The individual protocol steps are as follows:

1. $\mathcal{P}$ chooses $z' \in_{\mathcal{R}} G$ (a random version of z). Let a' be the binary encoding of z' and $\vec{f'}$ be the encoding of z' over $\mathbb{F}_{2^e}$ (as mentioned above for $\vec{f}$). As for a we can set $a' := \vec{f'}$. Let $\vec{h'}$ be the RS-encoding of z' over $\mathbb{F}_{2^e}$. Again, as for f and h the coefficients of f' and h' are binary encoded. $\mathcal{P}$ computes the QR-commitments to these encodings
$$\begin{aligned} C_{f'}^{\mathbb{F}} &:= com^{\mathsf{QR}}(\vec{f'}, \vec{x'}, n) = [\ com^{\mathsf{QR}}({f'}_{s,j}, x'_{s,j}, n)\ ]_{0 \le s \le l'-1, 1 \le j \le e} \\ C_{h'}^{\mathbb{F}} &:= com^{\mathsf{QR}}(\vec{h'}, \vec{y'}, n) = [\ com^{\mathsf{QR}}(h'_{s',j}, y'_{s',j}, n)\ ]_{0 \le s' \le l''-1, 1 \le j \le e} \end{aligned}$$
where $x'_{s,j} \in_{\mathcal{R}} \mathbb{Z}_n^*, y'_{s',j} \in_{\mathcal{R}} \mathbb{Z}_n^*$. $\mathcal{P}$ sends these commitments to $\mathcal{V}$.
2. $\mathcal{V}$ sends a binary (and randomly selected) challenge $c \in_R \{0, 1\}$ to $\mathcal{P}$.

3. If $c = 0$, then $\mathcal{P}$ sends $resp_0 := (state_{\mathcal{P}}^{C_{f'}^{\mathbb{F}}}, state_{\mathcal{P}}^{C_{h'}^{\mathbb{F}}})$ to $\mathcal{V}$.

   $\mathcal{V}$ proceeds as follows:

   (a) Verifies that

   $$\begin{aligned} true &\overset{?}{=} \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C_{f'}^{\mathbb{F}}}, state_{\mathcal{V}}^{C_{f'}^{\mathbb{F}}}) \wedge \\ true &\overset{?}{=} \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C_{h'}^{\mathbb{F}}}, state_{\mathcal{V}}^{C_{h'}^{\mathbb{F}}}) \end{aligned}$$

   where $state_{\mathcal{V}}^{C_{f'}^{\mathbb{F}}} := (C_{f'}^{\mathbb{F}}, n)$ and $state_{\mathcal{V}}^{C_{h'}^{\mathbb{F}}} := (C_{h'}^{\mathbb{F}}, n)$.

   (b) Checks whether the content of $C_{h'}^{\mathbb{F}}$ is the RS-encoding of the content of $C_{f'}^{\mathbb{F}}$ over $\mathbb{F}_{2^e}$.

4. If $c = 1$, then $\mathcal{P}$ sends the response $resp_1 := (state_{\mathcal{P}}^{C_{f''}^{\mathbb{F}}}, state_{\mathcal{P}}^{C_{h''}^{\mathbb{F}}})$ where $C_{f''}^{\mathbb{F}}$ and $C_{h''}^{\mathbb{F}}$ are supposed to be the commitments computed as $C_{f''}^{\mathbb{F}} := C_{f'}^{\mathbb{F}} C_f^{\mathbb{F}}$, $C_{h''}^{\mathbb{F}} := C_{h'}^{\mathbb{F}} C_h^{\mathbb{F}}$, and $state_{\mathcal{P}}^{C_{f''}^{\mathbb{F}}} := (\vec{f''}, v, n)$, $state_{\mathcal{P}}^{C_{h''}^{\mathbb{F}}} := (\vec{h''}, w, n)$ are supposed to be the corresponding opening information where $\vec{f''} \in \mathbb{F}_{2^e}^{l'}$, $\vec{h''} \in \mathbb{F}_{2^e}^{l''}$, $\vec{v} \in (\mathbb{Z}_n^*)^{l'e}$ and $w \in (\mathbb{Z}_n^*)^{l''e}$. (Note that $\vec{f''}$ and $\vec{h''}$ denote the contents of $C_{f''}^{\mathbb{F}}$ and $C_{h''}^{\mathbb{F}}$, and for the components of v and w the following should hold $v_{s,j} := x'_{s,j} x_{s',j}$ and $w_{s',j} := y'_{s,j} y_{s',j}$.)

   Now, $\mathcal{V}$ proceeds as follows:

   (a) Computes $C'^{\mathbb{F}}_f := C_{f'}^{\mathbb{F}} C_f^{\mathbb{F}}$ and $C'^{\mathbb{F}}_{h''} := C_{h'}^{\mathbb{F}} C_h^{\mathbb{F}}$.

   (b) Verifies that

   $$\begin{aligned} true &\overset{?}{=} \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C_{f''}^{\mathbb{F}}}, state_{\mathcal{V}}^{C'^{\mathbb{F}}_{f''}}) \\ true &\overset{?}{=} \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C_{h''}^{\mathbb{F}}}, state_{\mathcal{V}}^{C'^{\mathbb{F}}_{h''}}). \end{aligned}$$

   where $state_{\mathcal{V}}^{C'^{\mathbb{F}}_{f''}} := (C'^{\mathbb{F}}_{f''}, n)$ and $state_{\mathcal{V}}^{C'^{\mathbb{F}}_{h''}} := (C'^{\mathbb{F}}_{h''}, n)$.

   (c) Verifies that $\vec{h''} = \mathsf{RS}(\vec{f''})$ holds, i.e., it checks whether the content of $C_{h''}^{\mathbb{F}}$ is the RS-encoding of the content of $C_{f''}^{\mathbb{F}}$.

Lemma 5.4 *The protocol* $\mathsf{ProveEncRSCom}()$ *is complete.* □

Proof. For this property, it is assumed that both parties are honest and behave according to the specified protocol. For $c = 0$, one can easily see that $\mathcal{V}$ accepts. To see this for $c = 1$, consider the commitments $C'^{\mathbb{F}}_{f''}$ and

${C'}^{\mathbb{F}}_{h''}$. From homomorphic properties of the QR-commitments follows

$$\begin{aligned}
{C'}^{\mathbb{F}}_{f''} &= C^{\mathbb{F}}_{f'} C^{\mathbb{F}}_{f} \\
&= com^{\mathsf{QR}}(\vec{f'}, \vec{x'}, n) com^{\mathsf{QR}}(\vec{f}, \vec{x}, n) \\
&= com^{\mathsf{QR}}(\vec{f'} \oplus \vec{f}, \vec{x'} \bullet \vec{x}, n) \\
&= \left[\ com^{\mathsf{QR}}(f'_{s,j} \oplus f_{s,j}, x'_{s,j} x_{s,j}, n)\ \right]_{0 \le s \le l'-1, 1 \le j \le e} \\
&= \left[\ com^{\mathsf{QR}}(f''_{s,j}, v_{s,j}, n)\ \right]_{0 \le s \le l'-1, 1 \le j \le e}
\end{aligned}$$

where $f''_{s,j} := f'_{s,j} \oplus f_{s,j}$, and $v_{s,j} := x'_{s,j} x_{s,j}$, and

$$\begin{aligned}
{C'}^{\mathbb{F}}_{h''} &= C^{\mathbb{F}}_{h'} C^{\mathbb{F}}_{h} \\
&= com^{\mathsf{QR}}(\vec{h'}, \vec{y'}, n) com^{\mathsf{QR}}(\vec{h}, \vec{y}, n) \\
&= com^{\mathsf{QR}}(\vec{h'} \oplus \vec{h}, \vec{y'} \bullet \vec{y}, n) \\
&= \left[\ com^{\mathsf{QR}}(h'_{s'} \oplus h_{s'}, \vec{y'}_{s'} \vec{y}_{s'}, n)\ \right]_{0 \le s' \le l''-1} \\
&= \left[\ com^{\mathsf{QR}}(h'_{s',j} \oplus h_{s',j}, y'_{s',j} y_{s',j}, n)\ \right]_{0 \le s' \le l''-1, 1 \le j \le e} \\
&= \left[\ com^{\mathsf{QR}}(h''_{s',j}, w_{s',j}, n)\ \right]_{0 \le s' \le l''-1, 1 \le j \le e}
\end{aligned}$$

where $h''_{s',j} := h'_{s',j} \oplus h_{s',j}$ (i.e., $h'' := h \oplus h'$), and $w_{s,j} := y'_{s',j} y_{s',j}$.

Note that for two polynomials $\vec{f}, \vec{f'}$ defined over $\mathbb{F}_{2^e}$ it holds

$$\vec{f} + \vec{f'} := \vec{f} \oplus \vec{f'} := (f_{l'-1} \oplus f_{l'-1}, \cdots, f_0 \oplus f_0)$$

where $f_s \oplus f'_s = (f_{s,e-1} \oplus f'_{s,e-1}, \cdots, f_{s,0} \oplus, f'_{s,0})$ and $s \in [0, l'-1]$.

As specified by the protocol, h and h' are RS-encodings of f and f', i.e., $h := \mathsf{RS}(f)$ and $h' = \mathsf{RS}(f')$. It follows from the linearity of the RS-encoding

$$h'' = \vec{h} \oplus \vec{h'} = \mathsf{RS}(\vec{f}) \oplus \mathsf{RS}(\vec{f'}) = \mathsf{RS}(\vec{f} \oplus \vec{f'}) = \mathsf{RS}(\vec{f''})$$

where $\vec{f''} := \vec{f} \oplus \vec{f'}$. ■

Lemma 5.5 *The protocol* ProveEncRSCom() *is perfectly sound.* □

Proof. If $(C^{\mathbb{F}}_{f}, C^{\mathbb{F}}_{h}, n, l', l'', k, e) \notin L$ we show that the success probability of even a computationally unrestricted prover is at most 1/2.

Assume, we are given $(C^{\mathbb{F}}_{f}, C^{\mathbb{F}}_{h}, n, l', l'', k, e) \notin L$ (i.e., in particular $h \neq \mathsf{RS}(f)$), and an arbitrary prover $\mathcal{P}^*$ with an arbitrary auxiliary input $aux_{\mathcal{P}^*}$. We show that $\mathcal{P}^*$ can at most pass one of the $\mathcal{V}$'s challenges.

Suppose, $\mathcal{V}$ has accepted the response $resp_0$, i.e., it has verified that $(C^{\mathbb{F}}_{f'}, C^{\mathbb{F}}_{h'}, n, l', l'', k, e) \in L$ holds, in particular, $\vec{h'} = \mathsf{RS}(\vec{f'})$. Now, consider the case $c = 1$: $\mathcal{V}$'s verification of $resp_1$ would fail because of the following reason: The verification is successful if and only if $(C^{\mathbb{F}}_{f''}, C^{\mathbb{F}}_{h''}, n, l', l'', k, e) \in L$, i.e., there exist f'' and h'' with $\vec{h''} = \mathsf{RS}(\vec{f''})$ where $\vec{f''} = \vec{f'} \oplus \vec{f}$, $\vec{h''} =$

$\vec{h'} \oplus \vec{h}$ should hold due to the homomorphic property of the QR-commitment. However, since we assumed $h' = \mathsf{RS}(f')$, it follows from the linearity of RS-code that $\vec{h''} \neq \mathsf{RS}(f'')$ unless $\mathcal{P}^*$ is capable of opening the QR-commitments in two different ways. This is, however, not possible even for all powerful $\mathcal{P}^*$ since QR commitment scheme is information-theoretically binding.

Conversely, suppose, $\mathcal{V}$ has accepted the response $resp_1$, i.e., there exist f'' and h'' with $\vec{h''} = \mathsf{RS}(\vec{f''})$ where $\vec{f''} = \vec{f'} \oplus \vec{f}$, $\vec{h''} = \vec{h'} \oplus \vec{h}$ should hold due to the homomorphic property of the QR-commitment. Since we assumed $h \neq \mathsf{RS}(f)$, it follows from the linearity of RS-code that $\vec{h'} \neq \mathsf{RS}(f')$ where $\vec{f'} = \vec{f''} \oplus \vec{f}$, $\vec{h'} = \vec{h''} \oplus \vec{h}$. However, this would mean that $\mathcal{V}$ would not accept $resp_0$ unless $\mathcal{P}^*$ is capable of opening QR-commitment in two different ways. This is not possible even for all powerful $\mathcal{P}^*$ since QR commitment scheme is information-theoretically binding.

Hence, after sending $C^{\mathbb{F}}_{f'}, C^{\mathbb{F}}_{h'}$, $\mathcal{P}^*$ can correctly answer to at most one of the two possible challenges, and since $\mathcal{P}^*$ does not know the challenge in advance, its success probability is at most 1/2. ∎

Lemma 5.6 *The protocol* ProveEncRSCom() *is perfect zero-knowledge.* □

Proof. We construct a simulator with black-box access to $\mathcal{V}^*$ that can generate a view with the same distribution as $\mathcal{V}^*$'s view in the real protocol. The simulator $\mathcal{S}^{\mathcal{V}^*}$ is given in Algorithm 6.

Algorithm 6 The Simulator $\mathcal{S}^{\mathcal{V}^*}$ for protocol $\mathsf{ProveEncRSCom}()$

```
1. Choose $c \in_{\mathcal{R}} \{0,1\}$.
2. Choose $z' \in_{\mathcal{R}} G$.
   (a) Determine its encoding $\vec{f'}$ and its RS-encoding $\vec{h'} = \mathsf{RS}(\vec{f'})$ over $\mathbb{F}_{2^e}$.
   (b) Compute the following quantities: $C^{\mathbb{F}}_{f'} := com^{\mathsf{QR}}(\vec{f'}, \vec{x'}, n)$, $C^{\mathbb{F}}_{h'} := com^{\mathsf{QR}}(\vec{h'}, \vec{y'}, n)$, $C'^{\mathbb{F}}_{c} := C^{\mathbb{F}}_{f'}/(C^{\mathbb{F}}_{f})^c$ and $C''^{\mathbb{F}}_{c} := C^{\mathbb{F}}_{h'}/(C^{\mathbb{F}}_{h})^c$.
3. Start the verifier $\mathcal{V}^*$, send $C'^{\mathbb{F}}_{c}, C''^{\mathbb{F}}_{c}$, and wait until $\mathcal{V}^*$ sends challenge $c^*$.
4. IF $c^* = c$, THEN
   (a) send $resp_c$ accordingly, i.e., $resp_0 := (state_{\mathcal{P}}^{C^{\mathbb{F}}_{f'}}, state_{\mathcal{P}}^{C^{\mathbb{F}}_{h'}})$ and $resp_1 := (state_{\mathcal{P}}^{C^{\mathbb{F}}_{f'}}, state_{\mathcal{P}}^{C^{\mathbb{F}}_{h'}})$.
   (b) wait until $\mathcal{V}^*$ outputs $ind_{\mathcal{V}^*}$. Then output the view $view_{\mathcal{S}^{\mathcal{V}^*}} = (C'^{\mathbb{F}}_{c}, C''^{\mathbb{F}}_{c}, c, resp_c, ind_{\mathcal{V}^*})$.
   ELSE output $\perp$.
```

We consider the simulated view $(C^{\mathbb{F}}_{f'}, C^{\mathbb{F}}_{h'}, c, resp_c, ind_{\mathcal{V}^*})$, and compare it to the real view.

- $C'^{\mathbb{F}}_{c}, C''^{\mathbb{F}}_{c}$: For $c = 0$ we have QR-commitments $C'^{\mathbb{F}}_{0} = C^{\mathbb{F}}_{f'}, C''^{\mathbb{F}}_{0} = C^{\mathbb{F}}_{h'}$, i.e., these quantities are constructed in exactly the same way as in the real protocol. For $c = 1$ the quantity $C'^{\mathbb{F}}_{1} := C^{\mathbb{F}}_{f'}/C^{\mathbb{F}}_{f}$ is a QR-commitment to the polynomial $\vec{f''} := \vec{f} \oplus \vec{f'}$. This simply follows from the homomorphic properties of QR-commitments (see Section 3.5.4). $\vec{f''}$ is a random polynomial over $\mathbb{F}_{2^e}$ since it is a one-time pad of $\vec{f}$ with the randomly and uniformly chosen polynomial $\vec{f'}$ over $\mathbb{F}_{2^e}$. Similarly, the quantity $C''^{\mathbb{F}}_{1} := C^{\mathbb{F}}_{h'}/C^{\mathbb{F}}_{h}$ is a QR-commitment to a random polynomial $\vec{h''} := \vec{h'} \oplus \vec{h}$. Further, we have $\vec{h''} = \mathsf{RS}(\vec{f''})$ since RS-encoding is a linear operation. Thus, $C'^{\mathbb{F}}_{1}$ and $C''^{\mathbb{F}}_{1}$ have the correct form, and the same distribution as $C^{\mathbb{F}}_{f''}$ and $C^{\mathbb{F}}_{h''}$ in the real protocol.
- c^*: The challenge c^*, sent by $\mathcal{V}^*$ has the correct distribution, since $\mathcal{S}^{\mathcal{V}^*}$ maintains the distribution of c^* by verifying that $c^* = c$ and acting accordingly.
- $resp_c$: By construction the response $resp_c$ has the same distribution as in the real protocol: In the case $c = 0$, we have

$resp_0 := (state_{\mathcal{P}}^{C^{\mathbb{F}}_{f'}}, state_{\mathcal{P}}^{C^{\mathbb{F}}_{h'}})$. This is the opening information of QR-commitments to a random polynomial $\vec{f'}$ and its RS-encoding over $\mathbb{F}_{2^e}$ as in the real protocol. In the case $c = 1$, we have $resp_1 := (state_{\mathcal{P}}^{C^{\mathbb{F}}_{f'}}, state_{\mathcal{P}}^{C^{\mathbb{F}}_{h'}})$. This is again the opening information of the QR-commitments $C^{\mathbb{F}}_f C'^{\mathbb{F}}_1 = C^{\mathbb{F}}_{f'}$ and $C^{\mathbb{F}}_h C''^{\mathbb{F}}_1 = C^{\mathbb{F}}_{h'}$ to a random polynomial f' and its RS-encoding h' as in the real protocol.

The probability that the simulator fails is at most 1/2 since the simulated view is taken only if $c^* = c$. The running time of the simulator is polynomial since every step in Algorithm 6 can be performed in polynomial time. ■

From the previous lemmas we can conclude the following theorem:

Theorem 5.2 *The protocol* ProveEncRSCom() *is a perfect zero-knowledge proof of language membership.* □

Remark 5.4. One may use protocol ProveEncRSCom() for any linear code over $\mathbb{F}_{2^e}$. In general, the encoding of $\vec{f}$ is represented as $\vec{h} := \vec{f}G$ where G is a $dim \times L$ generating matrix. ○

5.2.3.1 Efficiency

We consider the main costs from different perspectives, namely, the number of required commitments, the number of main operations for the involved parties, and the number of communicated bits. Note that for different challenges c, we have different costs for different types of operations. We take the challenge with higher costs.

In the following, we can exploit that $l' = \log(a)$ since the encoding is over $\mathbb{F}_{2^e}$, and that $l'' = |\mathbb{F}_{2^e}| - 1 = 2^e - 1$ since l'' is the length of the RS-code.

- *Number of commitments*: $e(l' + l'') = \log(a) + e(2^e - 1)$
- *Number of operations for $\mathcal{P}$*:
  - Multiplication: $e(l+l') = \log(a)+e(2^e-1)$ multiplications modulo n (in case $c = 1$)
  - Squaring: $e(l + l') = \log(a) + e(2^e - 1)$ squarings modulo n (in case $c = 0$)
- *Number of operations for $\mathcal{V}$*:
  - Multiplication: $e(l' + l'') = \log(a) + e(2^e - 1)$ multiplications modulo n (in case $c = 1$)
  - Squaring: $e(l' + l'') = \log(a) + e(2^e - 1)$ squarings modulo n (in case $c = 0$)

- *Communicated bits*: $2e(l' + l'')\log(n) = 2\big(\log(a) + e(2^e - 1)\big)\log(n)$

Note that we have $(l' + l'')\log(n) = O(\mathsf{poly}(k))$ where k is the security parameter.

5.2.4 Proving Equality of Committed Numbers

We consider a protocol $\mathsf{ProveEqCom}()$ in which a prover $\mathcal{P}$ proves to a verifier $\mathcal{V}$ that the contents of two different commitments (here QR and DL commitments) are equal.

Description: Let the tuple $(\mathsf{GenParComQR}(), \mathsf{ProtComQR}(), \mathsf{ProtOpenQR}())$ be the QR-commitment scheme with $par^{\mathsf{QR}}_{\mathsf{com}} = n$, and $(\mathsf{GenParComDL}(), \mathsf{ProtComDL}(), \mathsf{ProtOpenDL}())$ be the DL commitment scheme with $par^{\mathsf{DL}}_{\mathsf{com}} = (p, q, g, h)$, as defined in Sections 3.5.4 and 3.5.5. See also these sections for the choice of the corresponding parameters.

Let C^{QR}_a and $C^{\mathsf{DL}}_{a^*}$ be given commitments that are supposed to be the QR-commitment and DL-commitment, i.e., they have the following form:

$$\begin{aligned} C^{\mathsf{QR}}_a &:= com^{\mathsf{QR}}(a, \vec{x}, n) = \big[\ com^{\mathsf{QR}}(a_i, x_i, n)\ \big]_{1 \le i \le l} \\ C^{\mathsf{DL}}_{a^*} &:= com^{\mathsf{DL}}(a^*, \vec{y}, n) = \big[\ com^{\mathsf{DL}}(a^*_i, y_i, n)\ \big]_{1 \le i \le l} \end{aligned}$$

where a and a^* denote the corresponding contents, and supposed to be equal, $x_i \in_{\mathcal{R}} \mathbb{Z}^*_n$ and $y_i \in_{\mathcal{R}} \mathbb{Z}_q$ and $length_2(a) = length_2(a^*) = l$. In the protocol $\mathsf{ProveEqCom}()$, the prover $\mathcal{P}$ proves to $\mathcal{V}$ that the contents of C^{QR}_a and $C^{\mathsf{DL}}_{a^*}$ are equal, i.e., $a = a^*$, and that it knows the opening information for these commitments.

The protocol is denoted as follows

$$(\mathcal{P} : -;\ \mathcal{V} : ind) \leftarrow \mathsf{ProveEqCom}(\mathcal{P} : state^C_{\mathcal{P}};\ \mathcal{V} : -;\ * : C, par^{\mathsf{QR}}_{\mathsf{com}}, par^{\mathsf{DL}}_{\mathsf{com}})$$

where $state^C_{\mathcal{P}} := (state^{C^{\mathsf{QR}}_a}_{\mathcal{P}}, state^{C^{\mathsf{DL}}_{a^*}}_{\mathcal{P}})$ denotes the opening information for the commitments $C := (C^{\mathsf{QR}}_a, C^{\mathsf{DL}}_{a^*})$ and $ind \in \{accept, reject\}$ is the output to the verifier indicating whether it accepts or not.

As usual, we first consider the language describing the objective to be proven. Since DL-commitments are information-theoretically (perfectly) hiding, the prover must also prove knowledge of their contents. For this, we have to define the corresponding relation R. R and the language L are defined as follows:[4]

$$\begin{aligned}
R := \big\{ &\big((a,\vec{x}),(a^*,\vec{y})),(C_a^{\mathsf{QR}},C_{a^*}^{\mathsf{DL}},n,(p,q,g,h),(k,l))\big) \mid \\
&n \in [\mathsf{GenParComQR}(k)] \wedge (p,q,g,h) \in [\mathsf{GenParComDL}(k)] \wedge \\
&a,a^* \in \{0,1\}^l, \vec{x} \in (\mathbb{Z}_n^*)^l, \vec{y} \in \mathbb{Z}_q^l : \\
&C_a^{\mathsf{QR}} = com^{\mathsf{QR}}(a,\vec{x},n) \wedge C_a^{\mathsf{DL}} = com^{\mathsf{DL}}(a,\vec{y},p,q,g,h)\big\}, \\
L := \big\{ &(C_a^{\mathsf{QR}},C_{a^*}^{\mathsf{DL}},n,(p,q,g,h),(k,l)) \mid \\
&\exists a,a^* \in \{0,1\}^l \wedge \exists \vec{x} \in (\mathbb{Z}_n^*)^l, \wedge \exists \vec{y} \in \mathbb{Z}_q^l \\
&: \big(((a,\vec{x}),(a^*,\vec{y})),(C_a^{\mathsf{QR}},C_{a^*}^{\mathsf{DL}},n,(p,q,g,h),(k,l))\big) \in R \wedge (a=a^*)\big\}.
\end{aligned}$$

Construction: The generation algorithm for this language, denoted by GenEqCom(), outputs $((C_a^{\mathsf{QR}},C_{a^*}^{\mathsf{DL}},n,(p,q,g,h),(k,l)),(\vec{x},\vec{y}))$. The individual protocol steps are as follows:

1. For $1 \le i \le l$, $\mathcal{P}$ prepares the strings $b_1 := (b_{1,1},\cdots,b_{1,l})$, and $b_2 := (b_{2,1},\cdots,b_{2,l})$ where $b_{1,i} \in_{\mathcal{R}} \{0,1\}$ and $b_{2,i} := \bar{b}_{1,i}$ is the binary complement of $b_{1,i}$.

2. $\mathcal{P}$ computes the following sets of auxiliary commitments

$$\begin{aligned}
C_1^{\mathsf{QR}} &:= com^{\mathsf{QR}}(b_1,\vec{x'}_1,n) = \big[\, com^{\mathsf{QR}}(b_{1,i},x'_{1,i},n)\,\big]_{1\le i\le l}, \\
C_2^{\mathsf{QR}} &:= com^{\mathsf{QR}}(b_2,\vec{x'}_2,n) = \big[\, com^{\mathsf{QR}}(b_{2,i},x'_{2,i},n\,\big]_{1\le i\le l}, \\
C_1^{\mathsf{DL}} &:= com^{\mathsf{DL}}(b_1,\vec{y'}_1,p,q,g,h) = \big[\, com^{\mathsf{DL}}(b_{1,i},y'_{1,i},p,q,g,h)\,\big]_{1\le i\le l}, \\
C_2^{\mathsf{DL}} &:= com^{\mathsf{DL}}(b_2,\vec{y'}_2,p,q,g,h) = \big[\, com^{\mathsf{DL}}(b_{2,i},y'_{2,i},p,q,g,h)\,\big]_{1\le i\le l}
\end{aligned}$$

   where $\vec{x'}_1,\vec{x'}_2 \in_{\mathcal{R}} (\mathbb{Z}_n^*)^l$ and $\vec{y'}_1,\vec{y'}_2 \in_{\mathcal{R}} (\mathbb{Z}_q)^l$.

3. $\mathcal{P}$ sends these commitments to $\mathcal{V}$.

4. $\mathcal{V}$ responds with a random challenge $c \in_R \{0,1\}$.

5. If $c = 0$, then $\mathcal{P}$ opens the auxiliary commitments by sending the response $resp_0 := (state_{\mathcal{P}}^{C_1^{\mathsf{QR}}}, state_{\mathcal{P}}^{C_2^{\mathsf{QR}}}, state_{\mathcal{P}}^{C_1^{\mathsf{DL}}}, state_{\mathcal{P}}^{C_2^{\mathsf{DL}}})$ to $\mathcal{V}$.

   $\mathcal{V}$ verifies first

$$\begin{aligned}
true &\overset{?}{=} \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C_1^{\mathsf{QR}}}, state_{\mathcal{V}}^{C_1^{\mathsf{QR}}}) \\
true &\overset{?}{=} \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C_2^{\mathsf{QR}}}, state_{\mathcal{V}}^{C_2^{\mathsf{QR}}}) \\
true &\overset{?}{=} \mathsf{VerOpenDL}(state_{\mathcal{P}}^{C_1^{\mathsf{DL}}}, state_{\mathcal{V}}^{C_1^{\mathsf{DL}}}) \\
true &\overset{?}{=} \mathsf{VerOpenDL}(state_{\mathcal{P}}^{C_2^{\mathsf{DL}}}, state_{\mathcal{V}}^{C_2^{\mathsf{DL}}})
\end{aligned}$$

[4]Note that L is a subset of the language L_R defined by the relation R.

where $state_{\mathcal{V}}^{C_j^{\mathsf{QR}}} := (C_j^{\mathsf{QR}}, n)$ and $state_{\mathcal{V}}^{C_j^{\mathsf{DL}}} := (C_j^{\mathsf{DL}}, p, q, g, h)$ for $j \in \{1, 2\}$.

Next, $\mathcal{V}$ verifies that $b_{2,i} = \bar{b}_{1,i}$, and that the content of C_1^{QR} is the same as the content of C_1^{DL}, and the content of C_2^{QR} is the same as the content of C_2^{DL}.

6. If $c = 1$, then $\mathcal{P}$ sends $resp_1 := (I, state_{\mathcal{P}}^{C_{*,I}^{\mathsf{QR}}}, state_{\mathcal{P}}^{C_{*,I}^{\mathsf{DL}}})$. Here, I is the index set $I = \{j | b_{j,i} = a_i\}$ where each $j \in \{1, 2\}$ points to a commitment in C_j^{QR} and C_j^{DL} whose content is supposed to be equal to a_i and a_i^* respectively.[5] Further, $C_{*,I}^{\mathsf{QR}}$ and $C_{*,I}^{\mathsf{DL}}$ are supposed to be the commitments computed as

$$\begin{aligned} C_{*,I}^{\mathsf{QR}} &:= C_a^{\mathsf{QR}} \left[\, com^{\mathsf{QR}}(b_{j,i}, x'_{j,i}, n) \, \right]_{1 \le i \le l}, \\ C_{*,I}^{\mathsf{DL}} &:= C_a^{\mathsf{DL}} / \left[\, com^{\mathsf{DL}}(b_{j,i}, y'_{j,i}, p, q, g, h) \, \right]_{1 \le i \le l} \end{aligned}$$

where $j \in I$, and $state_{\mathcal{P}}^{C_{*,I}^{\mathsf{QR}}} := (\mathsf{bin}(0), \vec{v})$, $state_{\mathcal{P}}^{C_{*,I}^{\mathsf{DL}}} := (\mathsf{bin}(0), \vec{w})$ are supposed to the corresponding opening information for these commitments with $\vec{v} \in (\mathbb{Z}_n^*)^l$ and $\vec{w} \in \mathbb{Z}_q^l$ (Note that the components of the vectors $\vec{v}, \vec{w}$ will then be $v_i := x_i x'_{j,i} \bmod n$ and $w_i := y_i - y'_{j,i} \bmod q$.)

$\mathcal{V}$ proceeds as follows:

(a) Computes

$$\begin{aligned} C'^{\mathsf{QR}}_{*,I} &:= C_a^{\mathsf{QR}} \left[\, com^{\mathsf{QR}}(b_{j,i}, x'_{j,i}, n) \, \right]_{1 \le i \le l}, \\ C'^{\mathsf{DL}}_{*,I} &:= C_{a^*}^{\mathsf{DL}} / \left[\, com^{\mathsf{DL}}(b_{j,i}, y'_{j,i}, n) \, \right]_{1 \le i \le l} \end{aligned}$$

where $j \in I$.

(b) Verifies that

$$\begin{aligned} true &\overset{?}{=} \mathsf{VerOpenQR}(state_{\mathcal{P}}^{C_{*,I}^{\mathsf{QR}}}, state_{\mathcal{V}}^{C'^{\mathsf{QR}}_{*,I}}) \\ true &\overset{?}{=} \mathsf{VerOpenDL}(state_{\mathcal{P}}^{C_{*,I}^{\mathsf{DL}}}, state_{\mathcal{V}}^{C'^{\mathsf{DL}}_{*,I}}). \end{aligned}$$

(c) Verifies that $C'^{\mathsf{QR}}_{*,I}$ and $C'^{\mathsf{DL}}_{*,I}$ are commitments to zero using the opening information sent by $\mathcal{P}$.

Lemma 5.7 *The protocol* $\mathsf{ProveEqCom}()$ *is complete.* □

Proof. The completeness of the protocol can easily be verified by applying the homomorphic properties of the QR and DL commitments similar to that of the previous protocols. ■

[5] To be in general case, we may let the prover send also the index set $I^* = \{j | b_{j,i} = a_i^*\}$. However, for the objective to be proven here, $I^* = I$ must hold. Thus, we use only I.

Lemma 5.8 *The protocol* $\mathsf{ProveEqCom}()$ *is perfectly sound.* □

Proof. We show that a cheating prover $\mathcal{P}^*$, even with unrestricted computing power, cannot convince the verifier except with the probability of at most 1/2.

This is done in two steps: First we prove weak soundness and then strong soundness. The reason for proving weak soundness is that the language membership for equality of QR and DL commitments always holds as DL-commitment is perfectly hiding, and thus, there always exists a value to which a commitment can be opened.

For weak soundness we prove that the prover indeed knows the contents of C_a^{QR} and $C_{a^*}^{\mathsf{DL}}$. For this, we construct a knowledge extractor $\mathcal{E}$ which, on the common input $Y := (C_a^{\mathsf{QR}}, C_{a^*}^{\mathsf{DL}}, n, (p,q,g,h), (k,l)) \in L$, can extract $X := \big((a,\vec{x}), (a^*, \vec{y})\big)$ with $(X,Y) \in R$ from any successful prover $\mathcal{P}^*$. The extractor is given in Algorithm 7.

Algorithm 7 The Extractor $\mathcal{E}$ for relation R

1. Start $\mathcal{P}^*$ from its initial state $state^0_{\mathcal{P}^*}$, and wait until it sends $C_1^{\mathsf{QR}}, C_2^{\mathsf{QR}}, C_1^{\mathsf{DL}}, C_2^{\mathsf{DL}}$. The contents of these commitments are denoted with b_1, b_2, b_1^*, b_2^*. Store the resulting state $state^1_{\mathcal{P}^*}$ of $\mathcal{P}^*$ at this stage.
2. Send the challenge $c = 0$ to $\mathcal{P}^*$, and wait until $\mathcal{P}^*$ returns $resp_0 = (b_1, \vec{x'}_1, b_2, \vec{x'}_2, b_1^*, \vec{y'}_1, b_2^*, \vec{y'}_2)$.
3. Start $\mathcal{P}^*$ at state $state^1_{\mathcal{P}^*}$, send the challenge $c = 1$, and wait until $\mathcal{P}^*$ returns $resp_1 = (I, (\mathsf{bin}(0), \vec{v}), (\mathsf{bin}(0), \vec{w}))$.
4. If both $resp_0$ and $resp_1$ pass the corresponding verifications, then output the following quantities: $a = (a_1, \cdots, a_l)$ with $a_i = b_{j,i}$, $\vec{x} = (x_1, \cdots, x_l)$ with $x_i \equiv v_i (x'_{j,i})^{-1} \bmod n$, $a^* = (a_l^*, \cdots, a_1^*)$ with $a_i^* = b_{j,i}^*$, and $\vec{y} = (y_1, \cdots, y_l)$ with $y_i \equiv w_i + y'_{j,i} \bmod q$ where $j \in I$.
5. Otherwise, repeat (go back to the first step).

If $\mathcal{E}$ halts, it will output $\big((a,\vec{x}), (a^*, \vec{y})\big)$ such that the relation R holds: From response $resp_0$, $\mathcal{E}$ has all $b_{j,i}$ values and all randomizing components, i.e., $\vec{x'}_j = (x'_{j,1}, \cdots x'_{j,l})$ and $\vec{y'}_j = (y'_{j,1}, \cdots, y'_{j,l})$ for $j = 1, 2$. From $resp_1$, it knows which of $b_{j,i}$ and $b_{j,i}^*$ are equal to a_i and a_i^*. Thus, it can determine (extract) the contents a and a^* of C_a^{QR} and $C_{a^*}^{\mathsf{DL}}$.

To extract the randomizing components of the commitments, we consider the product commitments $C^{\mathsf{QR}}_{*,I}$ and $C^{\mathsf{DL}}_{*,I}$. For the QR commitment we have

$$\begin{aligned} C^{\mathsf{QR}}_{*,I} &= [\ com^{\mathsf{QR}}(a_i, x_i, n)\ ]_{1\leq i\leq l}[\ com^{\mathsf{QR}}(b_{j,i}, x'_{j,i}, n)\ ]_{1\leq i\leq l} \\ &= [\ com^{\mathsf{QR}}(a_i \oplus b_{j,i}, x_i x'_{j,i}, n)\ ]_{1\leq i\leq l} \\ &= [\ com^{\mathsf{QR}}(a_i \oplus b_{j,i}, v_i, n)\ ]_{1\leq i\leq l} \end{aligned}$$

where $v_i := x_i x'_{j,i}$. This implies that $x_i \equiv v_i({x'}_{j,i})^{-1} \bmod n$. Similarly, for the DL-commitments we can write

$$\begin{aligned} C^{\mathsf{DL}}_{*,I} &= [\ com^{\mathsf{DL}}(a_i, x_i, p, q, g, h)\ ]_{1\leq i\leq l}\ /\ [\ com^{\mathsf{DL}}(b_{j,i}, x'_{j,i}, p, q, g, h)\ ]_{1\leq i\leq l} \\ &= [\ com^{\mathsf{QR}}(a_i - b_{j,i}, y_i - y'_{j,i}, p, q, g, h)\ ]_{1\leq i\leq l} \\ &= [\ com^{\mathsf{QR}}(a_i - b_{j,i}, w_i, p, q, g, h)\ ]_{1\leq i\leq l} \end{aligned}$$

where $w_i := y_i - y'_{j,i} \bmod q$. This implies that $y_i \equiv w_i + y'_{j,i} \bmod q$.

Next, we consider the success probability of the extractor: Let C denote the common input and A_ϵ denote the set of all possible random tapes $rand_{\mathcal{P}^*}$ for the prover $\mathcal{P}^*$. A_ϵ is the sample space underlying the interaction between $\mathcal{E}$ and $\mathcal{P}^*$.[6] The set $A_\nu := A_\epsilon \times \{0,1\}$ is the sample space underlying the interaction of $\mathcal{P}^*$ and $\mathcal{V}$, i.e., $A_\nu := \big\{(rand, chall) | rand \in_{\mathcal{R}} A_\epsilon \wedge chall \in_{\mathcal{R}} \{0,1\}\big\}$ where *chall* is the random tape of the verifier consisting of only one bit. Further, let Θ be the random value representing the commitments $T := (C^{\mathsf{QR}}_1, C^{\mathsf{QR}}_2, C^{\mathsf{DL}}_1, C^{\mathsf{DL}}_2)$, and $RESP_0$, $RESP_1$ be the random variables representing the responses of $\mathcal{P}^*$ to the challenges $chall = 0$ and $chall = 1$ of the verifier $\mathcal{V}$. Consider now the following events

$$\begin{aligned} E^0_T &:= \big\{rand \in A_\epsilon | \Theta(rand) = T \wedge true = \mathsf{Verify}(C, T, RESP_0(rand))\big\} \\ E^1_T &:= \big\{rand \in A_\epsilon | \Theta(rand) = T \wedge true = \mathsf{Verify}(C, T, RESP_1(rand))\big\} \end{aligned}$$

where $\mathsf{Verify}()$ denotes all $\mathcal{V}$'s verifications after receiving $resp_c = RESP_c(rand)$.

For $rand \in E^0_T \cap E^1_T$ the extractor outputs the tuple $X = \big((a, \vec{x}, a^*, \vec{y})\big)$ such that $(X, Y) \in R$ where $Y = \big((C^{\mathsf{QR}}_a, C^{\mathsf{DL}}_{a^*}), n, (p, q, g, h), (k, l)\big)$. We determine the probability for this event. For fixed $\Theta(rand) = T$, we consider the accepting event

$$E_T := \{(rand, c) \in A_\nu | \Theta(rand) = T \wedge true = \mathsf{Verify}(C, T, RESP_c(rand))\}.$$

[6]The length of the random tape $rand_{\mathcal{P}^*}$ of an interactive algorithm (function) $\mathcal{P}^*$ is a function $f_{\mathcal{P}^*}(\cdot)$ in the length of the common input $|C|$. Here, we can set $A_\epsilon := \{0,1\}^{f_{\mathcal{P}^*}(|C|)}$ because the extractor $\mathcal{E}$ only initializes $\mathcal{P}^*$ and makes no other random choices. The random tape $rand_{\mathcal{P}^*}$ is assigned a random string $rand \in_{\mathcal{R}} A_\epsilon$. After this no other random sources are used.

Now, we claim that for any T the following holds

$$\mathbf{Prob}[E_T^0] + \mathbf{Prob}[E_T^1] = 2\mathbf{Prob}[E_T]$$

where $\mathbf{Prob}[E_T] := P_{\mathcal{P}^*}(C, aux)$ denotes the success probability of $\mathcal{P}^*$. To see this, let $CHALL$ be the random variable denoting $\mathcal{V}$'s challenge. Then we can write

$$\begin{aligned}
\mathbf{Prob}[E_T] &= \mathbf{Prob}[E_T|CHALL = 0]\mathbf{Prob}[CHALL = 0] \\
&\qquad +\mathbf{Prob}[E_T|CHALL = 1]\mathbf{Prob}[CHALL = 1] \\
&= \frac{1}{2}\mathbf{Prob}[E_T|CHALL = 0] + \frac{1}{2}\mathbf{Prob}[E_T|CHALL = 1] \\
&= \frac{1}{2}\mathbf{Prob}[E_T^0] + \frac{1}{2}\mathbf{Prob}[E_T^1] \\
&= \frac{1}{2}(\mathbf{Prob}[E_T^0] + \frac{1}{2}\mathbf{Prob}[E_T^1]).
\end{aligned}$$

For the success probability $P_{\mathcal{E}}(C, aux)$ of the extractor $\mathcal{E}$ we can write

$$\begin{aligned}
P_{\mathcal{E}} &\geq \sum_T \mathbf{Prob}[E_T^0 \cap E_T^1] \\
&= \sum_T \big(\mathbf{Prob}[E_T^0] + \mathbf{Prob}[E_T^1] - \mathbf{Prob}[E_T^0 \cup E_T^1]\big) \\
&\geq \sum_T \big(\mathbf{Prob}[E_T^0] + \mathbf{Prob}[E_T^1] - \mathbf{Prob}[\Theta = T]\big) \\
&= \sum_T \big(\mathbf{Prob}[E_T^0] + \mathbf{Prob}[E_T^1]\big) - 1 \\
&= \sum_T \big(2\mathbf{Prob}[E_T]\big) - 1 \\
&= 2P_{\mathcal{P}^*} - 1
\end{aligned}$$

The expected number of iterations is $\frac{1}{P_{\mathcal{E}}} \leq \frac{1}{2P_{\mathcal{P}^*}-1}$, i.e., inverse polynomial proportional to the success probability of $\mathcal{P}^*$ (see Section 3.6.3). Further, each iteration of Algorithm 7 consists of steps which can be performed in polynomial time.

Next, we prove the strong soundness: If $\big(C_a^{\mathsf{QR}}, C_{a^*}^{\mathsf{DL}}, n, (p, q, g, h), (k, l)\big) \notin L$ then an arbitrary prover $\mathcal{P}^*$, who can even open the commitments $(C_a^{\mathsf{QR}}, C_{a^*}^{\mathsf{DL}})$, cannot convince the verifier $\mathcal{V}$ with probability better than 1/2. The proof for this part is similar to that of 5.2.2 since both constructions are based on the same cut and choose principle. Also here, we can conclude that the prover is only successful if it can open QR-commitments in two different ways, which is not possible since the QR-commitment scheme is information-theoretically binding. ■

Lemma 5.9 *The protocol* $\mathsf{ProveEqCom}()$ *is perfect zero-knowledge.* □

Proof. We construct a simulator with black-box access to $\mathcal{V}^*$ that can generate a view with the same distribution as $\mathcal{V}^*$ has in the real protocol. The simulator $\mathcal{S}^{\mathcal{V}^*}$ is given in Algorithm 8.

Algorithm 8 The Simulator $\mathcal{S}^{\mathcal{V}^*}$ for protocol $\mathsf{ProveEqCom}()$

```
Inputs: $C_a^{\mathsf{QR}}, C_{a^*}^{\mathsf{DL}}, n, (p, q, g, h)$
  1. Choose $c \in_{\mathcal{R}} \{0, 1\}$.
  2. Prepare $C_1^{\mathsf{QR}}, C_2^{\mathsf{QR}}, C_1^{\mathsf{DL}}, C_2^{\mathsf{DL}}$ as specified in the real protocol.
  3. Compute the commitments $C'^{\mathsf{QR}}_{1,c} := C_1^{\mathsf{QR}}/(C_a^{\mathsf{QR}})^c$, $C'^{\mathsf{QR}}_{2,c} := C_2^{\mathsf{QR}}/(C_a^{\mathsf{QR}})^c$, $C'^{\mathsf{DL}}_{1,c} := C_1^{\mathsf{DL}}(C_{a^*}^{\mathsf{DL}})^c$ and $C'^{\mathsf{DL}}_{2,c} := C_2^{\mathsf{DL}}(C_{a^*}^{\mathsf{DL}})^c$.
  4. Start the verifier $\mathcal{V}^*$, send $C'^{\mathsf{QR}}_{1,c}, C'^{\mathsf{QR}}_{2,c}, C'^{\mathsf{DL}}_{1,c}, C'^{\mathsf{DL}}_{2,c}$ and wait until $\mathcal{V}^*$ sends challenge $c^*$.
  5. IF $c^* = c$, THEN
     (a) send the response $resp_c$ according to the challenge $c$: Here, the simulator sets $resp_0 := (state_{\mathcal{P}}^{C_1^{\mathsf{QR}}}, state_{\mathcal{P}}^{C_2^{\mathsf{QR}}}, state_{\mathcal{P}}^{C_1^{\mathsf{DL}}}, state_{\mathcal{P}}^{C_2^{\mathsf{DL}}})$ and $resp_1 := (I, state_{\mathcal{P}}^{C_I^{\mathsf{QR}}}, state_{\mathcal{P}}^{C_I^{\mathsf{DL}}})$ where $I = \{j : b_{j,i} = 0\}$.
     (b) wait until $\mathcal{V}^*$ outputs $ind_{\mathcal{V}^*}$. Then output the view $view_{\mathcal{S}^{\mathcal{V}^*}} := (C'^{\mathsf{QR}}_{1,c}, C'^{\mathsf{QR}}_{2,c}, C'^{\mathsf{DL}}_{1,c}, C'^{\mathsf{DL}}_{2,c}, c, resp_c, ind_{\mathcal{V}^*})$.
     ELSE output $\perp$.
```

In the following, we consider the distribution of the simulated view $(C^{\mathsf{QR}}_{1,c}, C^{\mathsf{QR}}_{2,c}, C^{\mathsf{DL}}_{1,c}, C^{\mathsf{DL}}_{2,c}, c, resp_c, ind_{\mathcal{V}^*})$.

- $C'^{\mathsf{QR}}_{1,c}, C'^{\mathsf{QR}}_{2,c}, C'^{\mathsf{DL}}_{1,c}, C'^{\mathsf{DL}}_{2,c}$: For $c = 0$ these commitments are generated in exactly the same way as in the real protocol.

  For $c = 1$, the content of the commitments $C'^{\mathsf{QR}}_{1,c}, C'^{\mathsf{DL}}_{1,c}$ is $d_{1,i} := a_i \oplus b_{1,i}$, and that of the commitments $C'^{\mathsf{QR}}_{2,c}, C'^{\mathsf{DL}}_{2,c}$ is $d_{2,i} := a_i^* \oplus b_{2,i}$. Here, $d_{1,i}$ is random since it is a one-time pad of a fixed a_i with random $b_{1,i}$. Further, $d_{2,i} = \bar{d}_{1,i}$ because $b_{2,i} = \bar{b}_{1,i}$ and $a_i = a_i^*$. Thus, these commitments have the same distribution as $C_1^{\mathsf{QR}}, C_2^{\mathsf{QR}}, C_1^{\mathsf{DL}}, C_2^{\mathsf{DL}}$ in the real protocol.

- c^*: The challenge c^*, sent by $\mathcal{V}^*$ has the correct distribution, since the simulator $\mathcal{S}^{\mathcal{V}^*}$ maintains the distribution of c^* by verifying whether $c^* = c$ and acting accordingly.

- $resp_c$: By construction $resp_c$ has the same distribution as in the real protocol. For $c = 0$ this is clear. For $c = 1$, the index set I points to those commitments in $C'^{\mathsf{QR}}_1, C'^{\mathsf{QR}}_2, C'^{\mathsf{DL}}_1$ and C'^{DL}_2 whose contents make up a binary string equal to a. This is because for any $j \in I$ we have $b_{j,i} = 0$, and thus, the content of the corresponding i-th commitment in $C'^{\mathsf{QR}}_j, C'^{\mathsf{DL}}_j$ is $d_{j,i} := a_i \oplus b_{j,i} = a_i$. Hence, the QR and DL commitments restricted to I have the same content as C^{QR}_a and C^{DL}_a which correspond to the correct distribution as in the real protocol. Further, $state_{\mathcal{P}}^{C^{\mathsf{QR}}_I}, state_{\mathcal{P}}^{C^{\mathsf{DL}}_I}$ contained in $resp_1$ are the opening information for the QR and DL commitments to zero ($b_{j,i} = 0$) as in the real protocol.

The probability that the simulator fails is at most 1/2 since the simulated view is taken only if $c^* = c$. The running time of the simulator is polynomial since every step in Algorithm 8 can be performed in polynomial time. ■

From the lemmas above we conclude the following theorem.

Theorem 5.3 *The Protocol* ProveEqCom() *is perfect zero-knowledge proof of knowledge.* □

5.2.4.1 Efficiency

We consider the main costs from different perspectives, namely, the number of required commitments, the number of main operations for the involved parties, and the number of communicated bits. Note that for different challenges c, we have different costs for different types of operations. We take the challenge with higher costs.

In the following we have $l = \log(a)$.

- *Number of commitments*: $4l$
- *Number of operations for $\mathcal{P}$*:
  - Multiplication: l multiplications modulo n, and l multiplications modulo p (in case $c = 1$)
  - Exponentiation: $2l$ squarings modulo n, and $2l$ exponentiation modulo p (in case $c = 0$)
- *Number of operations for $\mathcal{V}$*:
  - Multiplication: l multiplications modulo p, and l multiplications modulo n (in case $c = 1$)
  - Exponentiation: $2l$ squarings modulo n, and $2l$ exponentiation modulo p (in case $c = 0$)
- *Communicated bits*: $2l(\log(n) + \log(p))$

Note that we have $l(\log n + \log p) = O(\mathsf{poly}(k))$ where k is the security parameter.

5.2.5 Committed Encoding from Unary Commitments

In this section, we describe an algorithm which allows us to compute the QR-commitment to a well defined encoding of a value, given the commitment to the unary encoding of that value. The algorithm is from Pfitzmann and Waidner (1997b).

Description: Let $\Sigma := \{0, 1, \cdots, m\} \subset \mathbb{N}$ be a finite alphabet and $\mathsf{Encode}() : \Sigma \mapsto \{0,1\}^l$ be a computable and invertible encoding. Let $\mathsf{Encode}(\Sigma) := W := \{w_0, \cdots w_m\}$. We assume, w.l.o.g., $w_a := \mathsf{Encode}(a)$, for $a \in \Sigma$.

Suppose, we are given the QR-commitment to the unary encoding of $a \in \Sigma$. We denote this commitment with

$$C_a^{\mathsf{un}} := com^{\mathsf{QR}}(\mathsf{un}(a), \vec{y}, n) = \left[\ com^{\mathsf{QR}}(a_s^{\mathsf{un}}, y_s, n)\ \right]_{1 \le s \le l'} .$$

We want to compute the commitment

$$C_{w_a} := com(w_a, \vec{z}, n) = \left[\ com^{\mathsf{QR}}(w_{a,j}, z_j, n)\ \right]_{1 \le j \le l}$$

to the corresponding encoding $w_a := \mathsf{Encode}(a)$ (Note the content a of C_a^{un} is not known). The algorithm for deriving this commitment is denoted by

$$C_{w_a} \leftarrow \mathsf{DeriveEncCom}(C_a^{\mathsf{un}}, par_{drv})$$

where par_{drv} denote the required parameters, e.g., Σ, the mapping $\mathsf{Encode}()$, etc.

Construction: The algorithm $\mathsf{DeriveEncCom}()$ is given in Algorithm 9.

Algorithm 9 Computing with unary commitments $\mathsf{DeriveEncCom}()$

```
Inputs: $C_a^{\mathsf{un}}, par_{drv}$

  1. Compute the QR-Commitments to the encoding of zero symbol of $\Sigma$,
     i.e., to $w_0 := \mathsf{Encode}(0)$:
         $C_{w_0} = com^{\mathsf{QR}}(w_0, \vec{x}_0, n) = [\ com^{\mathsf{QR}}(w_{0,j}, x_{0,j}, n)\ ]_{1 \le j \le l}$
     where $\vec{x}_0 \in_{\mathcal{R}} (\mathbb{Z}_n^*)^l$.
  2. Determine the sets $\Delta_j := \{s : w_{s,j} \neq w_{0,j}\}$ of those symbols $s$ in $\Sigma$
     whose encoding $w_s$ differs from $w_0$ at the $j$-th bit position.
  3. Compute the commitment to the $j$-th bit of $w_a$ as follows
         $C_{w_{a,j}} := com^{\mathsf{QR}}(w_{a,j}, z_j, n) = com^{\mathsf{QR}}(w_{0,j}, x_{0,j}, n)P$
     where $z_j = x_{0,j} \prod_{s:w_{s,j} \neq w_{0,j}} y_s$ and $P := \prod_{s:w_{s,j} \neq w_{0,j}} com^{\mathsf{QR}}(a_s^{\mathsf{un}}, y_s, n)$.
  4. Determine the commitment to $w_a$ as
         $C_{w_a} := com(w_a, \vec{z}, n) = [\ com^{\mathsf{QR}}(w_{a,j}, z_j, n)\ ]_{1 \le j \le l}$ .
```

Lemma 5.10 *Algorithm* $\mathsf{DeriveEncCom}()$ *is correct.* □

Proof. Due to the homomorphic properties of QR-commitment, the quantity $C_{w_{a,j}}$ is a QR-commitment for whose content c the following holds:

$$\begin{aligned} c &= w_{0,j} + \sum_{s:w_{s,j} \neq w_{0,j}} a_s^{\mathsf{un}} \\ &= w_{0,j} + 1_{w_{s,j} \neq w_{0,j}} \\ &= w_{0,j} + 1_{w_{a,j} \neq w_{0,j}} \\ &= w_{a,j}. \end{aligned}$$

■

As an example, assume that the encoding $\mathsf{Encode}()$ is binary encoding with $w_0 = 0$. Given the unary QR-commitment to a, compute the commitment to j-th bit of the binary encoding of a, i.e., $w_{a,i}^{\mathsf{bin}}$, as follows

$$com^{\mathsf{QR}}(w_{a,j}^{\mathsf{bin}}, z_j, n) = com^{\mathsf{QR}}(0, x_{0,j}, n) \prod_{s:w_{s,j} \neq w_{0,j}} com^{\mathsf{QR}}\big(a_s^{\mathsf{un}}, y_s, n\big).$$

Since $w_0^{\mathsf{bin}} := 0$, $\varDelta_j := \{s | w_{s,j}^{\mathsf{bin}} \neq w_{0,j}^{\mathsf{bin}}\}$ is the set of all s where $w_{s,j}^{\mathsf{bin}} = 1$. Now, if $a = s$ for a $s \in \varDelta_j$, then we have $a_s^{\mathsf{un}} = 1$, and $a_s^{\mathsf{un}} = 0$ for $s \neq a$. In this case, we can write $P_j := \prod_{\Delta_j} com^{\mathsf{QR}}\big(a^{\mathsf{un}}_{s,}, y_s, n\big) = com^{\mathsf{QR}}\big(1, z_i, n\big)$ where $z_i \in \mathbb{Z}^*_n$. This means, P_j is a commitment to 1 since only one of its terms is a commitment to 1, and the others are commitments to 0. It follows that $w^{\mathsf{bin}}_{a,j} = 1$. Similarly, P_j will be a commitment to zero, i.e., $w^{\mathsf{bin}}_{a,j} = 0$, if there is no $s \in \varDelta_j$ such that $s = a$.

5.2.5.1 Efficiency

- *Number of commitments*: The number of required commitments is $l + |\Sigma|$, i.e., l commitments for w_0 and $|\Sigma|$ commitments of the input C^{un}_a.
- *Number of operations*:
  - Multiplication: For each $j \in [1, l]$, Δ_j multiplications modulo n are required to compute $C_{w_{a,j}}$. In total, we have $\sum_{j=1}^{l} |\Delta_j|$.
  - Squaring: l squarings modulo n are required for computing C_{w_0}.

  Note that $l, |\Sigma| \in \mathsf{poly}(k)$ where k is the security parameter.

5.3 Construction for Embedding and Extracting

The idea is as follows: Each party, merchant and buyer, learns its own halfword, but this time, these halfwords are linked together. Each symbol consists of two halfsymbols, one known to the buyer and the other known to the merchant. In the identification, after extracting a word from the redistributed work, the symbols which do not match on the known halfsymbols are directly excluded. What remains is a word with many erasures, to which the Error-and-Erasure-Correcting Code (EECC) decoder is applied.

For completeness, we give a high-level description of the embedding protocol. The main structure is the same as defined for embedding in Section 4.5.4.3, except that some input/output variables have now slightly different names.

Key generation: $\mathcal{M}$ runs the key generation algorithm $key_{emb} \leftarrow \mathsf{GenKeyEmbed}(par_{emb})$ where $key_{emb} =: (key^{s}_{emb}, key^{p}_{emb})$ consists of a secret part key^{s}_{emb} and a public part key^{p}_{emb}.

Encoding: The protocol for encoding the embedding value $\vec{emb}$ in form of a collusion-tolerant codeword $word$ is denoted by

$$\begin{aligned}&(\mathcal{B}: -;\ \mathcal{M}: C_{word})\\ &\quad \leftarrow \mathsf{GenWord}(\mathcal{B}: state^{C_{emb}}_{\mathcal{B}};\ \mathcal{M}: half\vec{word}\_trace;\ *: C_{emb}, par^{p}_{\mathsf{CT}}, par_{\mathsf{EC}})\end{aligned}$$

where $C_{word} := com(word)$ denotes the commitment to $word$, and $par^{p}_{\mathsf{CT}}, par_{\mathsf{EC}} \in key^{p}_{emb}$ denote the parameters for the collusion-tolerant encoding and for the EECC.

Let the codeword to be embedded consist of L symbols from the alphabet Σ (Note that $L, \Sigma \in par^{p}_{\mathsf{CT}}$). In the following protocol steps $\mathcal{M}$'s tracing information and $\mathcal{B}$'s embedding information are appropriately encoded and then combined to make up the final codeword to be embedded in the underlying work. The individual steps of the protocol are as follows:

1. *Generating tracing halfword*: $\mathcal{M}$ secretly generates L halfsymbols.

$$half\vec{word}\_trace \leftarrow \mathsf{GenTraceWord}(par_{trace\_inf})$$

where par_{trace_inf} denotes the corresponding (security) parameters, and

$$half\vec{word}\_trace =: (halfsym\_trace_{1}, \cdots, halfsym\_trace_{L}).$$

2. *EECC encoding of embedding halfword*:

(a) $\mathcal{B}$ encodes $\vec{emb}$ using an appropriate Error-and-Erasure-Correcting Code (EECC) into L halfsymbols $halfsym\_emb_s$. This algorithm is denoted by

$$halfw\vec{ord}\_emb \leftarrow \mathsf{EECC}(\vec{emb}, par_{\mathsf{EC}})$$

where par_{EC} denotes the required parameters for EECC and

$$halfw\vec{ord}\_emb =: (halfsym\_emb_1, \cdots, halfsym\_emb_L).$$

(b) $\mathcal{B}$ commits to this halfword

$$\begin{aligned} C_{hw\_emb} &:= com(halfw\vec{ord}\_emb) \\ &:= [com(halfsym\_emb_1), \cdots, com(halfsym\_emb_L)], \end{aligned}$$

and sends the result C_{hw_emb} to $\mathcal{M}$.
The embedding halfsymbols will be combined with the tracing halfsymbols to form the symbols of the final codeword (see Step 4).

3. *Proof of correct EECC encoding*: $\mathcal{B}$ proves (in zero-knowledge) to $\mathcal{M}$ that the content of the commitment C_{hw_emb} is the correct EECC encoding of the content of C_{emb}. The protocol is denoted by

$$\begin{aligned} &(\mathcal{B} : -;\ \mathcal{M} : ind) \\ &\quad \leftarrow \mathsf{ProveEncECCom}(\mathcal{B} : state_{\mathcal{B}}^{C_{emb}}, state_{\mathcal{B}}^{C_{hw\_emb}};\ \mathcal{M} : -;\ * : A) \end{aligned}$$

where $A := (C_{emb}, C_{hw\_emb}, par_{\mathsf{EC}})$ denotes the common input, $state_{\mathcal{B}}^{C_{hw\_emb}}$ denotes the opening information for the commitments C_{hw_emb}, and $ind \in \{accept, reject\}$ indicates whether $\mathcal{M}$ accepts or rejects.

4. *Constructing collusion tolerant codeword:* $\mathcal{M}$ and $\mathcal{B}$ compute the collusion-tolerant codeword *word* to be embedded. The protocol is denoted by

$$\begin{aligned} &(\mathcal{B} : -;\ \mathcal{M} : C_{word}) \\ &\quad \leftarrow \mathsf{CollTolWord}(\mathcal{B} : state_{\mathcal{B}}^{C_{hw\_emb}};\ \mathcal{M} : par^{s}_{\mathsf{CT}}, halfw\vec{ord}\_trace;\ * : B) \end{aligned}$$

where $B := (C_{hw\_emb}, par^{p}_{\mathsf{CT}})$ denotes the common input, and par^{s}_{CT} denotes the secret parameters for the collusion-tolerant encoding. The protocol steps are as follows:

(a) *Combining halfwords*: The halfsymbols $halfsym\_trace_s$ and $halfsym\_emb_s$ are mixed into symbols sym_s. This is done by applying the secret random permutations $subst_s \in par_{\mathsf{CT}}$ of the underlying code alphabet Σ. We denote this by

$$sym_s := subst_s(halfsym\_trace_s \| halfsym\_emb_s)$$

where $sym_s \in \Sigma$, $s \in [1, L]$, and $\|$ denotes the concatenation.[7] The codeword associated to $\mathcal{B}$ is

$$\vec{word}' := (sym_1, \cdots, sym_L).$$

(b) *Collusion-tolerant encoding*: The codeword $\vec{word}'$ is encoded to a collusion-tolerant codeword *word*.

Embedding codeword: The codeword *word* is embedded into *work*. The corresponding protocol is denoted by

$$(\mathcal{B} : work^{fing};\ \mathcal{M} : -) \leftarrow \mathsf{EmbedWord}(\mathcal{B} : key_{\mathsf{com}};\ \mathcal{M} : work, C_{word}, key_{emb})$$

where key_{com} denotes the secret key constructed such that it allows the buyer $\mathcal{B}$ to open any commitment computed with the underlying commitment scheme, and $work^{fing}$ denotes the fingerprinted work.

Next, we will give a detailed instantiation of the components of the embedding protocol.

5.3.1 Instantiation of Embedding

In the following, we present explicit constructions for the components of the protocols $\mathsf{GenWord}()$ and $\mathsf{EmbedWord}$ $()$.

Collusion-tolerant code Γ: We instantiate the collusion-tolerant code by the outer code $\Gamma(L, N, n, d)$ defined over the alphabet $\Sigma := \mathbb{Z}_n$ (see Section 4.3.6). Recall that the meaning of the parameters are as follows: n is the size of the alphabet over which the outer code Γ is defined, L is the length of the outer code and is as function of the maximum collusion size *collsize*, $d = 2n^2 \log(2nL/\epsilon)$ is the block length of the basic codewords, and N is the number of symbols to be assigned by codewords. We assume $n := n_1 n_2$, and that n_1 and n_2 are small powers of 2 where $n_1 = 2^{\kappa_1}$ and $n_2 = 2^{\kappa_2}$. Thus, each symbol of the outer code is the concatenation of two strings of lengths κ_1 and κ_2 bits.[8] Note that we have $\Sigma_1 = \mathbb{Z}_{n_1}$ and $\Sigma_2 = \mathbb{Z}_{n_2}$.

For the embedding parameters we have

[7]The use of the permutations $subst_s$ is necessary for merchant's security (Pfitzmann and Waidner 1997a).

[8]The parameters (L, d, n_1, n_2) are chosen as polynomial functions of the given security parameters k, γ, and the maximum size of the collusion *collsize*. Note that the tracing error probability ϵ is usually defined as a function of a security parameter γ (here, $\epsilon{:=}2^{-\gamma}$).

- $par_{\mathsf{CT}}^{p}{:=}\{L, N, n, d, n_1, n_2, n, \mathit{collsize}, \epsilon, \Sigma_1, \Sigma_2, \Sigma\}$, $par_{\mathsf{CT}}^{s}{:=}\{\vec{subst}, \Pi_0\}$ are the public and secret parameters for the outer code of Boneh and Shaw where
  - $\vec{subst} := (subst_1, \cdots, subst_L)$ are L random permutations of the (finite) code alphabet $\Sigma \subset \mathbb{N}_0$ and
  - $\Pi_0 := \{\pi_1, \cdots, \pi_L\}$ is the set of permutations for the bits of basis codewords as described in Section 4.3.6.4.
- $key_{emb}^{p} := \{par_{\mathsf{RS}}, par_{\mathsf{CT}}^{p}\}$, $key_{emb}^{s} := \{sk^{\mathsf{WM}}, par_{\mathsf{CT}}^{s}\}$ are the public and secret embedding keys where
  - par_{RS} denotes the parameters for RS-code (e.g., $L, \mathbb{F}_{n_2}$ and the generating polynomial $g(x)$, see Section 5.2.1), and
  - sk^{WM} is the secret embedding and extracting key of the underlying watermarking scheme (see Section 4.3).

Bit-commitment scheme $com()$: We can instantiate this primitive with QR-commitment scheme or OU-commitment scheme (see Section 3.5). We take the QR-commitments where $par_{\mathsf{com}}^{\mathsf{QR}} = n'$ and $key_{\mathsf{com}}^{\mathsf{QR}} := (p', q')$ is the factorization of n'. In the embedding protocol n' is generated by the buyer $\mathcal{B}$ and then proven to be of correct form in a verifiable and secure procedure (see Sections 3.5.4 and 4.7.5.1 for more details on generation of these parameters). In the following, we omit writing $par_{\mathsf{com}}^{\mathsf{QR}}$ and only write n'.

Error and erasure correcting code $\mathsf{RS}()$: We instantiate the EECC with the Reed-Solomon code (see Section 5.2.1) defined over the finite field $\mathbb{F}_{2^{\kappa_2}}$ and denoted with $\mathsf{RS}()$. The code length is L.

Remark 5.5. Pfitzmann and Waidner (1997c) show that it is sufficient to use an EECC-code which tolerates $3n_1L$ errors and $L - L/\mathit{collsize}$ erasures to guarantee the security for the merchant. When applying RS-code, they show that $L \geq 2L^*/\log(L^*)$ is a sufficient condition where $L^* := \big(length_2(\vec{emb})\big)\mathit{collsize}$, and it is required that $n_1 \geq 24\mathit{collsize}$. ○

Generating tracing halfword $\vec{halfsym\_trace}$**:** $\mathcal{M}$ secretly generates the tracing halfsymbols by randomly choosing $halfsym\_trace_s \in_{\mathcal{R}} \Sigma_1$ for $s \in [1, L]$. The halfword is then $\vec{halfword\_trace} =: (halfsym\_trace_1, \cdots, halfsym\_trace_L)$.

Reed-Solomon encoding of embedding halfword: The RS-encoding of $\vec{emb}$ is denoted by $\vec{halfword\_emb} := (halfsym\_emb_1, \cdots, halfsym\_emb_L)$ where $halfsym\_emb_s \in \mathbb{F}_{2^{\kappa_2}}$ for $s \in [1, L]$.

Proof of equality of committed numbers $\mathsf{ProveEqCom}()$**:** The common input C_{emb} to the embedding protocol $\mathsf{EmbedProt}()$ is supposed to be a commitment to $\vec{emb}$. Since we use QR-commitment within the embedding protocol, we would like C_{emb} to be QR-commitment. However, in the protocols previous to the embedding protocol, the commitments to $\vec{emb}$ are based on DL-commitments which we denoted with C^{DL}_{emb} (see Section 4.7.5.3).

To solve this problem, we need a way to transform the DL-commitment to the corresponding QR-commitment. More concretely, we need a proof that the content of both commitments are equal. We implement this with the protocol $\mathsf{ProveEqCom}()$ explained in Section 5.2.4.[9]

Thus, in the following C_{emb} represents QR-commitments to $\vec{emb}$.

Proof of correct RS encoding $\mathsf{ProveEncRSCom}()$**:** Since we instantiate EECC with RS-code, we can instantiate protocol $\mathsf{ProveEncECCom}()$ with the protocol $\mathsf{ProveEncRSCom}()$ from Section 5.2.3. In this protocol, $\mathcal{B}$ proves to $\mathcal{M}$ that the content of a commitment $C^{\mathsf{bin}}_{hw\_emb}$ is the RS-encoding of the content of the commitment C_{emb}. Here, $C^{\mathsf{bin}}_{hw\_emb}$ denotes the QR-commitment to the binary encoding of $\vec{halfword\_emb}$. We denote this binary encoding with

$$\vec{halfword\_emb}^{\mathsf{bin}} := (halfsym\_emb^{\mathsf{bin}}_1, \cdots, halfsym\_emb^{\mathsf{bin}}_L)$$

where

$$halfsym\_emb^{\mathsf{bin}}_s := (halfsym\_emb^{\mathsf{bin}}_{s,0}, \cdots, halfsym\_emb^{\mathsf{bin}}_{s,\kappa_2-1})$$

for $s \in [1, L]$. The QR-commitment to the binary encoding is:

$$\begin{array}{rcl} C^{\mathsf{bin}}_{hw\_emb} & := & com^{\mathsf{QR}}(\vec{halfword\_emb}^{\mathsf{bin}}, \vec{x}, n') \\ & := & \left[\ com^{\mathsf{QR}}(halfsym\_emb^{\mathsf{bin}}_s, \vec{x}_s, n')\ \right]_{1 \le s \le L} \\ & := & \left[\ com^{\mathsf{QR}}(halfsym\_emb^{\mathsf{bin}}_{s,j}, x_{s,j}, n')\ \right]_{1 \le s \le L,\, 0 \le j \le \kappa_2 - 1} \end{array}$$

where $x_{s,j} \in_{\mathcal{R}} \mathbb{Z}^*_{n'}$.

Constructing collusion tolerant codeword $\mathsf{CollTolWord}()$**:** To implement protocol $\mathsf{CollTolWord}()$ we slightly modify its procedure as outlined in Section 5.3:

1. $\mathcal{B}$ encodes $\vec{halfword\_emb}$ in unary. We denote this with

$$\vec{halfword\_emb}^{\mathsf{un}} := (halfsym\_emb^{\mathsf{un}}_1, \cdots, halfsym\_emb^{\mathsf{un}}_L)$$

where $halfsym\_emb^{\mathsf{un}}_s := (halfsym\_emb^{\mathsf{un}}_{s,0}, \cdots, halfsym\_emb^{\mathsf{un}}_{s,n_2-1})$.

[9]Note that we consider this protocol as a part of embedding protocol, but, one may also consider it as a protocol previous to the embedding protocol, and use its output as common input to the embedding protocol.

2. $\mathcal{B}$ computes the QR-commitments to $\vec{halfword\_emb}^{\mathsf{un}}$

$$\begin{aligned} C^{\mathsf{un}}_{hw\_emb} &:= com^{\mathsf{QR}}(\vec{halfword\_emb}^{\mathsf{un}}, \vec{y}, n') \\ &:= \left[\, com^{\mathsf{QR}}(halfsym\_emb^{\mathsf{un}}_s, \vec{y}_s, n') \,\right]_{1\le s\le L} \\ &:= \left[\, com^{\mathsf{QR}}(halfsym\_emb^{\mathsf{un}}_{s,j}, y_{s,j}, n') \,\right]_{1\le s\le L,\, 0\le j\le n_2-1} . \end{aligned}$$

We denote the commitments to the individual halfsymbols $halfsym\_emb^{\mathsf{un}}_s$ as follows:

$$\begin{aligned} C^{\mathsf{un}}_{hs\_emb_s} &:= com^{\mathsf{QR}}(halfsym\_emb^{\mathsf{un}}_s, \vec{y}_s, n') \\ &:= \left[\, com^{\mathsf{QR}}(halfsym\_emb^{\mathsf{un}}_{s,j}, y_{s,j}, n') \,\right]_{0\le j\le n_2-1} . \end{aligned}$$

3. For each $halfsym\_emb_s$, $s \in [1, L]$, $\mathcal{B}$ proves to $\mathcal{M}$ that the commitments $C^{\mathsf{bin}}_{hs\_emb_s}$ and $C^{\mathsf{un}}_{hs\_emb_s}$ have the same content. This is done with the protocol $\mathsf{ProveEqEncCom}()$ described in Section 5.2.2.
4. For each $halfsym\_trace_s$, $\mathcal{M}$ uses the commitments $C^{\mathsf{un}}_{hs\_emb_s}$ to compute the commitments $C_{w_{sym_s}}$ to the basic code encoding $w_{sym_s} := \Gamma_0(sym_s)$ of the symbol sym_s where $sym_s := subst_s(halfsym\_emb_s || halfsym\_trace_s)$. To see how this is done, we illustrate it by using the Tables T_s shown in Table 5.2.

Table 5.2 The codeword table T_s for the symbol sym_s

$halfsym\_trace_s$	$halfsym\_emb_s$	sym'_s	sym_s $:=subst_s(sym'_s)$	w_{sym_s} $:=\Gamma_0(sym_s)$
0	0	0	5	$0001\cdots111$
	1	1	3	$1111\cdots111$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	n_2-2	n_2-2		$0011\cdots111$
	n_2-1	n_2-1		$0000\cdots111$
1	0	n_2		$0111\cdots111$
	1	n_2+1		$0000\cdots111$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	n_2-2	$2n_2-2$		$0000\cdots111$
	n_2-1	$2n_2-1$		$0000\cdots111$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n_1-1	0	$n-n_2$		$0000\cdots000$
	1	$n-n_2+1$		$0000\cdots111$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	n_2-2	$n-2$		$0000\cdots001$
	n_2-1	$n-1$		$0000\cdots111$

The first column of Table 5.2 contains all possible search halfsymbols $\mathit{halfsym\_trace}_s \in \Sigma_1$ with $\mathit{length}_2(\mathit{halfsym\_trace}_s) = \kappa_1$. For each of these search halfsymbols there are n_2 possible embedding halfsymbols $\mathit{halfsym\_emb}_s \in \Sigma_2$ with $\mathit{length}_2(\mathit{halfsym\_emb}_s) = \kappa_2$. The third column contains the symbols $\mathit{sym}'_s \in \Sigma$ determined by $\mathit{halfsym\_trace}_s \| \mathit{halfsym\_emb}_s$, and the fourth column contains the permutations of symbols sym'_s determined by $\mathit{subst}_s(\mathit{sym}'_s)$.

The last column of the table contains the encoding of each symbol sym_s with a basic codeword $w_{\mathit{sym}_s} := \Gamma_0(\mathit{sym}_s)$ from Boneh and Shaw where $l_w := \mathit{length}_2(w_{\mathit{sym}_s}) = (n-1)d$. For readability, the basic codewords in the table are shown without repeating, i.e., we set $d = 1$ (see also Section 4.3.6.4). Note that the figures in the last two columns of the table are only examples.

The procedure for computing $C_{w_{\mathit{sym}_s}}$ is as follows:[10] Let r be the value of $\mathit{halfsym\_trace}_s$ in the Table 5.2, and consider the corresponding row r in the table.

(a) $\mathcal{M}$ first prepares the commitments to the unary encoding of symbols sym'_s. For this, he arranges a tuple of n commitments, where at the positions, rn_2 to $rn_2 + n_2 - 1$ he places the commitments $C^{\mathsf{un}}_{\mathit{hs\_emb}_s}$, and at the remaining $n - n_2$ positions he places random commitments to zero. We denote this tuple of commitments with $C^{\mathsf{un}}_{\mathit{sym}'_s}$.

(b) $\mathcal{M}$ permutes the commitments in $C^{\mathsf{un}}_{\mathit{sym}'_s}$ by applying the permutation subst_s. The resulting commitment is the unary commitment $C^{\mathsf{un}}_{\mathit{sym}_s} := \mathit{subst}_s(C^{\mathsf{un}}_{\mathit{sym}'_s})$ to the symbol sym_s.

(c) On input $C^{\mathsf{un}}_{\mathit{sym}_s}$ and the parameters $\mathit{par}_{\mathit{drv}}$ required for the underlying encoding, $\mathcal{M}$ runs the algorithm $\mathsf{DeriveEncCom}()$ from Section 5.2.5 to compute the commitment $C_{w_{\mathit{sym}_s}}$, i.e.,

$$C_{w_{\mathit{sym}_s}} \leftarrow \mathsf{DeriveEncCom}(C^{\mathsf{un}}_{\mathit{sym}_s}, \mathit{par}_{\mathit{drv}})$$

for $s \in [1, L]$. The encoding of the symbols sym'_s is done according to the Boneh and Shaw basic code, i.e., $\Gamma_0(\mathit{sym}'_s)$ as mentioned above.

(d) The commitment C_{word} to the word word is denoted by

$$C_{\mathit{word}} := (C_{w_{\mathit{sym}_1}}, C_{w_{\mathit{sym}_2}}, \cdots, C_{w_{\mathit{sym}_L}}).$$

(e) The individual commitments $C_{w_{\mathit{sym}_s}}$ in C_{word} are permuted before actual embedding, i.e.,

$$C_{\mathit{word}^\pi} := \big(\pi_1(C_{w_{\mathit{sym}_1}}), \pi_2(C_{w_{\mathit{sym}_2}}), \cdots, \pi_L(C_{w_{\mathit{sym}_L}})\big).$$

[10]This commitment should have the form $C_{w_{\mathit{sym}_s}} := \mathit{com}^{\mathsf{QR}}(w_{\mathit{sym}_s}, \vec{z}, n')$ for some $\vec{z} \in (\mathbb{Z}^*_{n'})^{l_w}$.

For simplicity, we will use the notation *word* for the codeword to be embedded in the following.

Embedding word EmbedWord(): Let *word* be the codeword to be embedded into *work* with $length_2(word) := l$. For embedding this codeword, $\mathcal{M}$ exploits the homomorphic property of the underlying QR commitment scheme. Note that in the following expressions, for simplicity, we omit writing the parameters of the QR-commitment i.e., the random values chosen from $\mathbb{Z}_{n'}$ and the modulo number n'.

$\mathcal{M}$ proceeds as follows (see also Section 4.3.5):

1. Transform *work* to an appropriate embedding domain (e.g., frequency coefficients). We assume that the components of *work* in the embedding domain are represented as bit strings. We denote this representation by
$$\vec{work} := (work_1, work_2, \ldots, work_{l'}, work_\delta).$$
2. Select the components which should carry the marks, i.e., the marking positions. For this, $\mathcal{M}$ applies the embedding key $sk^{\mathsf{WM}} := (i_1, \cdots, i_l)$. Let $(work_{i_1}, work_{i_2}, \ldots, work_{i_l})$ be the selected components. We assume that the individual marks are disjoint, meaning that the modified positions in the bit strings are disjoint. For each mark, there are two possible versions of the underlying work, one marked with a binary 0 and one with a binary 1 (see also Remark 4.5). We denote the two versions of each component $work_{i_j}$ with $work^0_{i_j}$ and $work^1_{i_j}$. With $\Delta_{i_j} := \{u : work^0_{i_j,u} \neq work^1_{i_j,u}\}$ we denote the set of those bit positions in $work_{i_j}$ where both versions are distinct. Recall that $work_{i_j,u}$ denotes the u-th bit in the string $work_{i_j}$.
Without loss of generality, we write
$$\vec{work} := (work_{i_1}, work_{i_2}, \ldots, work_{i_l}, work'_\delta)$$
Note that we arrange the components of $\vec{work}$ in this way for readability reasons, and should keep in mind their correct order for the purpose of secure embedding.
3. $\mathcal{M}$ marks each of the selected components with zero, i.e., he replaces $work_{i_j}$, $i_j \in sk^{\mathsf{WM}}$ with $work^0_{i_j}$. We denote the resulting *zero codeword* with $word^0$, and the marked version with
$$\vec{work}^{\,0} := (work^0_{i_1}, work^0_{i_2}, \ldots, work^0_{i_l}, work'_\delta).$$
4. $\mathcal{M}$ commits to $work^0_{i_j}$, and for each i_j, he multiplies $com^{\mathsf{QR}}(work^0_{i_j})$ with commitment $com^{\mathsf{QR}}(word_j)$ at bit positions Δ_{i_j} where the two

(marked) versions of this component are distinct. We denote this computation as follows:

$$\begin{aligned} C^*{}_{work^{fing}_{i_j}} &:= com^{\mathsf{QR}}(word_j)\ \big[\ com^{\mathsf{QR}}(work^0_{i_j,u})\ \big]_{u\in\Delta_{i_j}} \\ &:= \big[\ com^{\mathsf{QR}}(word_j \oplus work^0_{i_j,u})\ \big]_{u\in\Delta_{i_j}} \\ &:= \big[\ com^{\mathsf{QR}}(work^{fing}_{i_j,u})\ \big]_{u\in\Delta_{i_j}} \end{aligned}$$

where $word_j$ denotes the j-th bit in the word $word$, and $work^{fing}_{i_j,u} := work^0_{i_j,u} \oplus word_j$ denotes the modified version of $work_{i_j,u}$.
Now, if $word_j = 1$ then those bits at positions Δ_{i_j} are flipped through exoring with 1, and $\mathcal{M}$ obtains the commitment $com^{\mathsf{QR}}(work^1_{i_j})$ to the component $work^1_{i_j}$. Otherwise ($word_j = 0$) the corresponding component remains unchanged. We denote the final result with

$$C^*{}_{work^{fing}} := [C_{work_{i_1}}, \cdots, C_{work_{i_l}}].$$

5. $\mathcal{M}$ sends $C_{work^{fing}} := [C^*{}_{work^{fing}}, C_{work'_\delta}]$ with the reversed ordering to $\mathcal{B}$ where $C_{work'_\delta} := com^{\mathsf{QR}}(work'_\delta)$. Note that $\mathcal{M}$ may not need to commit to $work_\delta$, and may send it as plaintext.
6. $\mathcal{B}$ decrypts the commitments $C_{work^{fing}}$ using $key^{\mathsf{QR}}_{\mathsf{com}}$, i.e.,

$$\vec{work}^{fing} \leftarrow \mathsf{OpenComQR}(key^{\mathsf{QR}}_{\mathsf{com}}, C_{work^{fing}})$$

7. $\mathcal{B}$ re-transforms the components $\vec{work}^{fing}$ to the original domain to obtain $work^{fing}$.

As one can see, if both parties behave according to the procedure explained above, it follows from the homomorphic property of the QR-commitment and the correctness of the underlying watermarking scheme that $\mathcal{B}$ obtains a fingerprinted work in which the constructed codeword is correctly embedded. Thus, we can conclude the following lemma:

Lemma 5.11 *The protocol* EmbedWord() *is complete.* □

Remark 5.6. Reducing data expansion: In the embedding procedure described before, we used commitments to individual bits. This leads to a large data expansion. The number of bits is at most $O\big(l_{work}\log(n')\big)$ where $l_{work} := length_2(\vec{work})$ is the bit length of the underlying work in embedding domain, and n' is the public parameter of the QR-commitment (In case the security requirements are not violated, one may reduce the number of bits by sending $work_\delta$ in plaintext.)

To further reduce the number of commitments, we can use a commitment scheme which enables us to commit to many bits at the same time. For this, Pfitzmann and Schunter (1996) propose a procedure in which they apply an encryption scheme based on **higher degree residue** as the underlying commitment scheme (see Benaloh (1994) and Naccache and Stern (1998)). However, the decryption algorithm of this scheme is very inefficient. Instead, we can apply the OU encryption scheme (Okamoto and Uchiyama 1998) which has a very efficient decryption (see Sections 3.4.2.3 and 3.5.6). In the following we explain how this can be done.

We compute commitments $C_{work'_\mu}$ to consecutive strings $work'_\mu$ of the representation of the underlying work where $length_2(work'_\mu) = k'$ and $2^{k'} < p$ must hold for OU-scheme (see Section 3.5.6)

If a bit $work'_{\mu,\nu}$ belongs to $work_{i_j}$, we XOR $word_j$ at that position. Since $\mathcal{M}$ knows the value $work'_{\mu,\nu}$, he can perform the XOR operation by adding/subtracting $2^\nu word_j$ to/from the value of the bit-string $work'_\mu$. To do this, $\mathcal{M}$ shifts the corresponding bit $word_j$ in $word$ to the ν-th position by computing $\big(com^{\mathsf{OU}}(word_j)\big)^{2^\nu}$. More concretely, if $work'_{\mu,\nu} = 0$ he computes

$$com^{\mathsf{OU}}(work'_\mu)\big(com^{\mathsf{OU}}(word_j)\big)^{2^\nu} = com^{\mathsf{OU}}(work'_\mu + 2^\nu word_j)$$

otherwise ($work'_{\mu,\nu} = 1$)

$$com^{\mathsf{OU}}(work'_\mu)/\big(com^{\mathsf{OU}}(word_j)\big)^{2^\nu} = com^{\mathsf{OU}}(work'_\mu - 2^\nu word_j).$$

In this way, we can strongly reduce the number of bits to at most $O\big(\frac{l_{work}}{k'}\log(n')\big)$ where $n' = p'^2 q'$ is now the corresponding modulo parameter of the OU-scheme. The additional cost are local multiplications for the shifting operations. For instance, by using square and multiply exponentiation scheme, this cost is maximal in the order of $O(\log(\varphi(n'))$, where $\varphi(n')$ is the group order in OU-scheme. ○

We summarize the instantiation of the embedding protocol in Algorithm 10. For brevity, we consider the correct execution of protocols where $ind = accept$, otherwise the protocols are aborted.

Algorithm 10 Details of embedding protocol $\mathsf{EmbedProt}()$

```
$(\mathcal{B} : -;\ \mathcal{M} : ind) \leftarrow \mathsf{ProveEqCom}(\mathcal{B} : state_{\mathcal{B}}^{C_{emb}}, state_{\mathcal{B}}^{C_{emb}^{\mathsf{DL}}};\ \mathcal{M} : -;\ * : C_{emb}, C_{emb}^{\mathsf{DL}})$;
$(\mathcal{M} : \vec{halfword}\_trace \in_{\mathcal{R}} \Sigma_1^L)$;
$(\mathcal{B} : \vec{halfword}\_emb) \leftarrow \mathsf{RS}(\mathcal{B} : \vec{emb}, par_{\mathsf{RS}})$;
$(\mathcal{B} : -;\ \mathcal{M} : ind) \leftarrow \mathsf{ProveEncRSCom}(\mathcal{B} : state_{\mathcal{B}}^{C_{emb}}, state_{\mathcal{B}}^{C_{hw\_emb}};\ \mathcal{M} : -;\ * : C_{emb}, C_{hw\_emb}, par_{\mathsf{RS}})$;
for $s = 1$ to $L$ do
    $(\mathcal{B} : -;\ \mathcal{M} : ind) \leftarrow \mathsf{ProveEqEncCom}(\mathcal{B} : state_{\mathcal{B}}^{C_{hs\_emb_s}^{\mathsf{bin}}}, state_{\mathcal{B}}^{C_{hs\_emb_s}^{\mathsf{un}}};\ \mathcal{M} : -;\ * : C_{hs\_emb_s}^{\mathsf{bin}}, C_{hs\_emb_s}^{\mathsf{un}})$;
end for;
for $s = 1$ to $L$ do
    $(\mathcal{M}$ : Derive $C_{sym'_s}^{\mathsf{un}}$ from $C_{hs\_emb_s}^{\mathsf{un}}$ );
    $(\mathcal{M} : C_{sym_s}^{\mathsf{un}} \leftarrow subst_s(C_{sym'_s}^{\mathsf{un}}))$;
    $(\mathcal{M} : C_{w_{sym_s}}) \leftarrow \mathsf{DeriveEncCom}(\mathcal{M} : C_{sym_s}^{\mathsf{un}}, par_{drv})$;
end for;
$\big(\mathcal{M} : C_{word} \leftarrow \pi_1(C_{w_{sym_1}}), \cdots, \pi_i(C_{w_{sym_L}})\big)$;
$(\mathcal{M} : work^0) \leftarrow \mathsf{EmbedMark}(\mathcal{M} : work, word^0, sk^{\mathsf{WM}})$;
$(\mathcal{M} : C_{work^{fing}} \leftarrow C_{word} C_{work^0})$; {multiplication should be understood as defined in Section 5.3.1}
$(\mathcal{B} : \vec{work}^{fing}) \leftarrow \mathsf{OpenComQR}(\mathcal{B} : key_{\mathsf{com}}^{\mathsf{QR}}; C_{\vec{work}^{fing}})$;
$(\mathcal{B}$ : Obtain $work^{fing}$ by re-transforming $\vec{work}^{fing}$ to the original domain )
```

5.3.2 Construction for Extracting Procedure

After finding a redistributed copy $work^{red}$ for which $true \leftarrow \mathsf{simtest}(work, work^{red})$ holds, the merchant $\mathcal{M}$ applies the extraction algorithm denoted by

$$\vec{emb} \leftarrow \mathsf{ExtractProt}(work^{red}, key_{emb}, REC_{\mathcal{M}}^{work}).$$

The extraction procedure for the embedding procedure in Algorithm 10 is presented by Algorithm 11.

Algorithm 11 Details of extraction algorithm $\mathsf{ExtractProt}()$

1. Extract an outer codeword from $work^{red}$ using the extracting algorithm of the underlying watermarking scheme.

$$word^{red} \leftarrow \mathsf{ExtractMark}(work^{red}, work, sk^{\mathsf{WM}}).$$

2. For each of the L positions of the outer code
   (a) apply the tracing algorithm $\mathsf{ExtractBSBC}(sym_i^{red}, \pi_i)$ of the basis code Γ_0 to identify symbols sym_i^{ext}, $i \in [1, L]$ (see Section 4.3.6.4),
   (b) decrypt each of these symbols using $subst_i^{-1}$, and
   (c) separate each symbol sym_i^{ext} into two halves of length κ_1 and κ_2 bits. The resulting outer codeword is denoted by $\vec{word}^{ext}$, and the halfword consisting of the first halfsymbols is denoted by $\vec{halfword\_trace}^{ext}$.
3. Search among purchase records $REC_{\mathcal{M}}^{work}$ of the underlying work $work$ for the one where the corresponding $\vec{halfword\_trace}$ has at least $L/collsize$ halfsymbols in common with $halfword\_trace^{ext}$.
4. Reconstruct the value $\vec{emb}$ from the second halfsymbols of $word^{ext}$ as follows:
   (a) Exclude all those symbols sym_i^{ext} which cannot belong to this traitor because their first halfsymbols are different from those in $halfword\_trace$. The remaining second halfsymbols represent a word $\vec{halfword\_emb}_i^{ext}$ with many erasures.
   (b) Run the decoding of EECC (here RS-decoding) to obtain $\vec{emb}$.

5.3.3 Security Analysis

In this section, we analyze the security of the embedding/extracting procedure according to the requirements defined in Section 4.5.4.2. We assume that we are given a robust watermarking scheme $(\mathsf{GenKeyWM}(), \mathsf{EmbedMark}(), \mathsf{ExtractMark}())$ which also satisfies the conditions of the *marking assumption*. Further, we use the QR commitment scheme $(\mathsf{GenParComQR}(), \mathsf{ProtComQR}(), \mathsf{ProtOpenQR}())$ and the DL-commitment scheme $(\mathsf{GenParComDL}(), \mathsf{ProtComDL}(), \mathsf{ProtOpenDL}())$ as defined in Section 3.5.

The embedding procedure consists of two main parts: A part where the merchant $\mathcal{M}$ and the buyer $\mathcal{B}$ are involved in the proof protocols $\mathsf{ProveEqCom}()$, $\mathsf{ProveEncRSCom}()$, $\mathsf{ProveEqEncCom}()$, and a part where the merchant alone performs local computations, in particular on the commitments to certain encodings of the embedding value $\vec{emb}$ (see Algorithm 10).

5.3.3.1 Security for Merchant

The security requirements for the merchant are soundness and secrecy as defined in Section 4.5.4.2. Soundness guarantees that if the buyer obtains a fingerprinted work after the completion of the embedding protocol then the desired information is correctly embedded into the underlying work. Secrecy guarantees that the embedding process reveals no additional information on merchant's secret inputs beyond the fingerprinted work. In the following, we show that the scheme fulfils these requirements.

- *Soundness*: First, we consider the proof subprotocols $\mathsf{ProveEqCom}()$, $\mathsf{ProveEncRSCom}()$, $\mathsf{ProveEqEncCom}()$ within the embedding procedure. It follows from Lemmas 5.2, 5.5 and 5.8 that each of these protocols is a perfectly sound proof system. Further, due to the logic of the main protocol the inputs of these subprotocols match with the outputs of the preceding subprotocols:

  In the subprotocol $\mathsf{ProveEqCom}()$, $\mathcal{B}$ proves knowledge of the contents of the commitments C_{emb} and C^{DL}_{emb}, and that these contents are equal (Theorem 5.3). Here, a proof of knowledge and a proof of language membership is required contrary to a mere proof of language membership.[11] In the subprotocol $\mathsf{ProveEncRSCom}()$, $\mathcal{B}$ proves that the content of C_{hw_emb} is the RS-encoding of the content of C_{emb}.

  Finally, in the subprotocol $\mathsf{ProveEqEncCom}()$, she proves that the content of $C^{\mathsf{un}}_{hw\_emb}$ is the unary encoding of the content of C_{hw_emb}. In both cases, a proof of language membership is sufficient since QR-commitments are information-theoretically binding. It follows that the contents of commitments C_{emb}, C_{hw_emb} and $C^{\mathsf{un}}_{hw\_emb}$ are the correct encodings of the content of C^{DL}_{emb} as specified by the protocol.

  Next, we consider $\mathcal{M}$'s own computations regarding the construction of the desired codeword. These are the computation of the commitments $C^{\mathsf{un}}_{sym_s}$, $C_{w_{sym_s}}$ and finally embedding the content of C_{word} with the algorithm $\mathsf{EmbedWord}()$. Since $C^{\mathsf{un}}_{hw\_emb}$ has the correct form as mentioned above, it follows from Step 4 of the algorithm $\mathsf{CollToIWord}()$ (Section 5.3.1) that also the content of $C^{\mathsf{un}}_{sym_s}$ has the correct form. The correctness of the content of $C_{w_{sym_s}}$ follows from the correctness of the algorithm $\mathsf{DeriveEncCom}()$ (see Section 5.2.5). Thus, the content of C_{word} has the correct encoding as specified by the protocol. The

[11] The reason for proof of knowledge is that the applied DL-commitment is information-theoretically hiding, and thus, there always exist a content of C^{DL}_{emb} in the corresponding language, and so, the language membership is trivial.

correctness of $\mathsf{EmbedWord}()$ follows from the homomorphic property of the QR-commitments, and from the correctness of the underlying watermarking scheme (Lemma 5.11). It follows that the content of C_{word} will be correctly embedded into the underlying work. Thus, if $\mathcal{B}$ obtains a fingerprinted work $work^{fing}$ so the specified encoding of $\vec{emb}$ is embedded in $work^{fing}$.

By definition of the embedding and extracting procedures, if a value $\vec{emb}$ is correctly embedded into a work $work$, then the extraction algorithm $\mathsf{ExtractProt}()$ extracts this value from a version of $work^{red}$ for which $true \leftarrow \mathsf{simtest}(work, work^{red})$ holds.

- *Secrecy*: Due to the construction, $\mathcal{M}$ generates the quantities $key^{s}_{emb} := \{\Pi_0, \vec{subst}, sk^{\mathsf{WM}}\}$, $\vec{halfword\_trace}$, C_{word} secretly and locally. Further, he keeps the components of the original work $\vec{work}$ secret, and the only information $\mathcal{B}$ obtains is the fingerprinted work $work^{fing}$.

Note that the collusion-tolerance is a security requirement of the fingerprinting protocol. However, we remark that according to the construction, the embedding protocol already embeds a collusion-tolerant encoding of $\vec{emb}$ into the underlying work, and the properties of the construction (*collsize*-secure outer code, and RS-encoding, etc.) guarantee that as long as the maximum tolerable collusion *collsize* is not exceeded, the extracted value is the embedding value $\vec{emb}$ of a traitor (Pfitzmann and Waidner (1997a)).

Thus, we can conclude:

Lemma 5.12 *The embedding protocol described in Section 5.3.1 is secure for the merchant.* □

5.3.3.2 Security for Buyer

The security requirement for the buyer is secrecy as defined in Section 4.5.4.2. It guarantees that the embedding process reveals no information on buyer's secret inputs. In the following, we show that the embedding protocol is zero-knowledge.

Secrecy: Each of the subprotocols involving the secret inputs of the buyer, i.e., $\mathsf{ProveEqCom}()$, $\mathsf{ProveEncRSCom}()$, $\mathsf{ProveEqEncCom}()$ is a perfect zero-knowledge proof. This follows from Theorems 5.1, 5.2 and 5.3. However, due to the construction of the embedding protocol, the (sequential) composition of these subprotocols is not perfect zero-knowledge since in the subsequent subprotocols, $\mathcal{B}$ (the prover) computes new QR-commitments to the secret inputs and sends them to $\mathcal{M}$ (the verifier). For the overall simulation, one can simulate these commitments by QR-commitments to fixed values, and if a distinguisher exists distinguishing the real view from the completely simulated one, then one can use it to break the semantic security of the QR commitment which is based on QRA.[12] Thus, it follows from the *general composition theorem* for sequential composition of auxiliary input zero-knowledge protocols (see, e.g., Goldreich and Oren (1994) and Definition 3.8 in Section 3.6.4):

Lemma 5.13 *The embedding protocol described in Section 5.3.1 is computationally zero-knowledge (for the merchant) under Quadratic Residuosity Assumption (QRA).* □

Thus, under QRA, the embedding protocol leaks no additional information on $\vec{emb}$ to $\mathcal{M}$ except the information $\mathcal{M}$ obtains on $\vec{emb}$ from the QR-commitment C_{emb} which is again secure under the QRA assumption.[13]

5.3.4 Efficiency of Embedding Protocol

In this section, we consider the efficiency of the main components of the embedding protocol. The costs we consider for the efficiency are the number of required commitments, the number of main operations for the involved parties, and the number of communicated bits.

Before going into details, we recall in the following the main convention we make use of: We denote the binary length of a value a with $length_2(a)$, and denote the length of a word *word* defined over an alphabet A with $length_A(word)$. Thus, we have $\kappa_2{:=}length_2(n_2)$ and $length_{\mathbb{F}_{n_2}}(\mathsf{RS}(\vec{emb})){:=}L$. We further set $l_{emb} := length_2(\vec{emb})$. Since we apply RS-code, we have $L = n_2 - 1$. Further, since the encodings of $\vec{emb}$ is over $\mathbb{F}_{2^{\kappa_2}}$, we can write $l_{emb} = \kappa_2 length_{\mathbb{F}_{n_2}}(emb)$. The length of the basic code $\Gamma_0(n, d)$ used to encode the symbols of $\Sigma := \{0, 1, 2, \cdots, n-1\}$ is given by $length_2(w \in \Gamma_0(n, d)) = (n-1)d$. The main parameter for the QR-commitment is $par^{\mathsf{QR}}_{\mathsf{com}} = n'$ (see Section 3.5.4), and those for the DL-commitment are the primes p and q. Note that we have $par^{\mathsf{DL}}_{\mathsf{com}} = (p, q, g, h)$ (see Section 3.5.5).

For the zero-knowledge protocols we consider the efficiency only for one iteration. The total efficiency of each of these protocols will then be γ_S times the efficiency of one iteration where γ_S is the number of the iterations. Further, note that in zero-knowledge protocols, we always consider the challenge with larger costs.

Proving equality of committed numbers ProveEqCom()

- *Number of commitments*: There are $3l_{emb}$ QR-commitments and $3l_{emb}$ DL-commitments required.

[12] Note that we obtain an overall perfect zero-knowledge protocol, if we input all these commitments together with other common inputs as a "global common input" to the main protocol. However, we will then lose the desired modularity.

[13] Note that semantic security is sufficient since the commitments are never opened.

- *Number of operations for $\mathcal{M}$*:
  - Multiplication: There are $2l_{emb}$ modulo n' and $2l_{emb}$ modulo p multiplications.
  - Exponentiation: There are $2l_{emb}$ squarings modulo n' and $2l_{emb}$ exponentiations mod p
- *Number of operations for $\mathcal{B}$*:
  - Multiplication: There are $2l_{emb}$ modulo n' and $2l_{emb}$ modulo p multiplications.
  - Exponentiation: There are $2l_{emb}$ squarings modulo n' and $2l_{emb}$ exponentiation mod p.
- *Communicated bits*: There are $3(\log(n') + \log(p))l_{emb}$ bit transmitted.

Proof of correct EECC encoding $\mathsf{ProveEncRSCom}()$

- *Number of commitments*: There are $2length_2(n_2)\big(length_{\mathbb{F}_{n_2}}(\vec{emb}) + length_{\mathbb{F}_{n_2}}(\mathsf{RS}(\vec{emb}))\big)$ QR-commitments. Using the above relations, we have $2(l_{emb} + L\kappa_2) = 2(l_{emb} + \kappa_2(n_2 - 1))$.
- *Number of operations for $\mathcal{M}$*:
  - Multiplication: There are $length_2(n_2)\big(length_{\mathbb{F}_{n_2}}(\vec{emb}) + length_{\mathbb{F}_{n_2}}(\mathsf{RS}(\vec{emb}))\big)$ multiplications modulo n'. Using the above relations, we have $l_{emb} + (n_2 - 1)\kappa_2$ multiplications modulo n'.
  - Exponentiation: There are $length_2(n_2)\big(length_{\mathbb{F}_{n_2}}(\vec{emb}) + length_{\mathbb{F}_{n_2}}(\mathsf{RS}(\vec{emb}))\big)$ squarings modulo n'. Using the above relations, we have $l_{emb} + (n_2 - 1)\kappa_2$ squarings modulo n'.
- *Number of operations for $\mathcal{B}$*:
  - Multiplication: Identical to that for $\mathcal{M}$.
  - Exponentiation: Identical to that for $\mathcal{M}$.
- *Communicated bits*: There are $2length_2(n_2)\big(length_{\mathbb{F}_{n_2}}(\vec{emb}) + length_{\mathbb{F}_{n_2}}(\mathsf{RS}(\vec{emb}))\big)length_2(n')$ bits transmitted. Using the relations above, we have $2\big(l_{emb} + L\kappa_2\big)\log(n') = 2\big(l_{emb} + \kappa_2(n_2 - 1)\big)\log(n')$.

Proof of Equality of Binary and Unary Committed Numbers $\mathsf{ProveEqEncCom}()$

- *Number of commitments*: There are $n_2\big(length_2(halfsym\_emb_s) + n_2\big) = n_2\big(\kappa_2 + n_2\big)$ QR-commitments.

- *Number of operations for $\mathcal{M}$*:
  - Multiplication: There are $2n_2\big(length_2(halfsym\_emb_s) + n_2\big) = 2n_2\big(\kappa_2 + n_2\big)$ multiplications modulo n'.
  - Exponentiation: There are $n_2\big(length_2(halfsym\_emb_s) + n_2\big) = n_2\big(\kappa_2 + n_2\big)$ squarings modulo n'.
- *Number of operations for $\mathcal{B}$*:
  - Multiplication: Identical to that for $\mathcal{M}$.
  - Exponentiation: Identical to that for $\mathcal{M}$.
- *Communicated bits*: There are $2Ln_2\big(length_2(halfsym\_emb_s) + n_2\big) = 2Ln_2\big(\kappa_2 + n_2\big)$ bits transmitted.

Deriving codeword DeriveEncCom()**:**

- *Number of commitments*: For each $C^{\mathsf{bin}}_{hs\_emb_s}$, $s \in [1, L]$, the output of this algorithm is the QR-commitment $C_{w_{sym_s}}$ to the basic codeword w_{sym_s} of the symbol sym_s. For computing these commitments, $\mathcal{M}$ requires the commitment C_{sym_s}, and the commitment C_{w_0} to the basic codeword w_0 for the symbol "0" in $\mathbb{Z}_n$. The former one consists of n QR-commitments. For the latter one, $\mathcal{M}$ actually needs to compute $(n-1)$ QR-commitments (Note that these commitments are used for the computation only, and we do not need to repeat them d times, i.e., we can set $d = 1$). Thus, effectively $L(n-1) + Ln = L(2n-1)$ commitments are required.
- *Number of operations for $\mathcal{M}$*:
  - Multiplication: Only those commitments in C_{sym_s} are multiplied which are at the positions where $w_{sym_s,j}$ differs from $w_{0,j}$. Due to the structure of the basic codewords (see Section 5.2.5), we have $|\Delta_j| = j$, and the number of multiplications required for each sym_s is $\sum_{j=1}^{n-1} |\Delta_j| = n(n-1)/2$. For L symbols, we have $Ln(n-1)/2$.

Embedding EmbedWord()

Let $l_{work} := length_2(\vec{work})$ be the bit length of the underlying work. Assume, we require at most λ bits of the underlying work to embed one mark.

- *Number of commitments*:
  Since each bit in the codeword *word* is repeated d times, $\mathcal{M}$ has to commit to at least $d\lambda length_2(w_{sym_s}) = d\lambda(n-1)$ bits of the underlying work for each symbol sym_s. Thus, the number of required QR-commitments is at least $\lambda Ld(n-1)$. However, this number is at most l_{work}.

- *Number of operations for $\mathcal{M}$*:
  - Multiplication: Commitments are multiplied only at the positions where marks has to be embedded, i.e., $d\lambda L length_2(w_{sym_s}) = \lambda Ld(n-1)$ multiplications modulo n' are required (for computing commitments to the remaining bits of the work only squaring modulo n' are required.)
  - Exponentiation: We require at most l_{work} squarings modulo n'.
- *Number of operations for $\mathcal{B}$*: At most l_{work} square roots modulo n' must be computed which we may interpret as the cost for maximal $O(l_{work} \log(\varphi(n')))$ multiplications modulo n'.[14]
- *Number of communicated bits*: There are at most $l_{work} \log(n')$ bits sent to the buyer.

5.3.4.1 Discussion on Efficiency of Embedding Protocol

In the following, we consider the maximal cost for the buyer and the merchant.

- *Main costs for merchant*: Among the zero-knowledge protocols, ProveEqEncCom() is the most costly one: The computational cost is $O(n_2^2) = O(n^2) = O(\mathsf{poly}(k))$ multiplications, and the number of QR-commitments is in the same order, i.e., $O(n^2)$.

  Regarding other parts of the embedding protocol, the most costly ones are the algorithms DeriveEncCom() and EmbedWord(). The former algorithm requires $O(Ln) = O(n^2) = O(\mathsf{poly}(k))$ commitments and $O(Ln^2) = O(n^3)$ multiplications. The latter requires at least $O(Ldn) = O(n^4 \log n)$ and at most l_{work} QR-commitments that, obviously, determines the number of the required operations and the communicated bits respectively.

  Thus, as one can see, the computational and communicational costs are strongly dominated by the costs of EmbedWord().

- *Main costs for buyer*: Among the zero-knowledge protocols, ProveEqEncCom() is the most costly one: The computational cost is $O(n^2) = O(\mathsf{poly}(k))$ multiplications, and the number of QR-commitments is in the same order, i.e., $O(n^2)$.

  Regarding other parts of the embedding protocol, the dominant costs for $\mathcal{B}$ lies in EmbedWord() where in worst case, she may have to compute l_{work} square roots modulo n' which requires maximal $O(l_{work} \log(\varphi(n')))$ multiplications modulo n'.

[14]Let $n' = p'q'$ where p', q' are known Blum integers. Then one requires the expected running time of $O\big((\log p')^3\big)$ bit operations to compute a square root modulo n'.

Thus, the major part of the computational and communicational cost is caused by the embedding EmbedWord() where it may be required to commit to all bits of the underlying work. In this context, the type of the used commitment has a great impact on the computational, storage and communicational costs, in particular for the buyer $\mathcal{B}$. As shown in Remark 5.6, we can strongly reduce these costs by using a commitment scheme which allows us to commit to many bits at the same time.

5.4 Conclusion

In Section 4.6 and this chapter, we presented the first explicit and reasonably efficient construction for an anonymous fingerprinting scheme, and proved its security. The construction is modular and offers several advanced security properties such as collusion-tolerance, asymmetry and anonymity. A further property of our construction is that it provides direct non-repudiation, i.e., in case of illegal redistribution, the rightful copyright holder *alone* can obtain enough evidence against the cheating user(s) and convince any honest third party. In particular, we give the first explicit construction for the core module of the fingerprinting scheme, namely, the embedding protocol. In this protocol the identifying information is securely and verifiably embedded into the underlying digital content.

Implementing the anonymity property requires a more enhanced and complex protocol design. However, as we have shown, the complexity of the whole fingerprinting protocol is dominated by the complexity of the embedding procedure, so that the anonymity is currently not the bottleneck for implementation. The bottleneck for real life applications is rather the collusion-tolerance. This is because the best known collusion-tolerant codes are still too long to be practical. The construction is more practical for applications which do not require this property.

The most known proposals for collusion-tolerant encoding use completely different approaches that cannot be easily compared with each other (see also Section 4.3.6). However, some important issues are as follows: The construction in Boneh and Shaw (1995) and Boneh and Shaw (1998) is data-independent collusion-tolerant encoding and abstracts from the underlying watermarking schemes whereas the constructions in Kilian et al. (1998) and Ergun, Kilian, and Kumar (1999) are already collusion-tolerant watermarking schemes, and thus, data-dependent. Further, the latter constructions use real numbers and do not consider error-analysis for discrete value implementation. Moreover, while Boneh and Shaw (1998) prove a lower bound for their codes and a large gap still remains, Ergun, Kilian, and Kumar (1999) give an almost tight lower bound for their construction.

In this thesis, we used the codes from Boneh and Shaw (Boneh and Shaw (1995), Boneh and Shaw (1998)) as the underlying collusion-tolerant encoding, in particular, because its data-independency makes a modular design of asymmetric fingerprinting schemes easier. The design of shorter collusion-tolerant codes is an ongoing research. Since our fingerprinting schemes are modular, new codes can efficiently be incorporated into our protocols. Another issue which may be of interest is to examine whether the class of watermarking schemes introduced in Kilian et al. (1998) (and Ergun, Kilian, and Kumar (1999)) can be efficiently made asymmetric.

Bibliography

Adelsbach, André, Birgit Pfitzmann, and Ahmad-Reza Sadeghi. 2000, October. "Proving Ownership of Digital Content." Edited by Andreas Pfitzmann, *Information Hiding—3rd International Workshop, IH'99*, Volume 1768 of *Lecture Notes in Computer Science*. Dresden, Germany: Springer-Verlag, Berlin Germany, 126–141.

Adelsbach, André, and Ahmad-Reza Sadeghi. 2002. "Secure Management of Digital Rights." Technical report, Department of Computer Science, Saarland University.

Ajtai, Miklós, and Cynthia Dwork. 1997, May. "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence." *Proceedings of the 29th Annual Symposium on Theory Of Computing (STOC)*. El Paso, TX, USA: ACM Press, 284–293.

Alon, Noga, and Joel H. Spencer. 1992. *The Probabilistic Method, with an appendix on open problems by Paul Erdös*. John Wiley & Sons.

Anderson, Ross J. 2001. *Security Engineering — A Guide to Building Dependable Distributed Systems*. John Wiley & Sons.

Anderson, Ross J., and Markus Kuhn. 1996, November. "Tamper Resistance – a Cautionary Note." *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*. USENIX, Oakland, California, 1–11.

Babai, Laszlo, and Endre Szemerédi. 1984. "On the complexity of matrix group problems." *Proceedings of the 25th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 229–240.

Bach, Eric, and Jeffrey Shallit. 1996. *Algorithmic Number Theory — Efficient Algorithms*. Volume I. Cambridge, USA: MIT Press. ISBN: 0-262-02405-5.

Barak, Boaz, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. 2001. "On the (Im)possibility of Obfuscating Programs." Edited by Joe Kilian, *Advances in Cryptology – CRYPTO '2001*, Volume 2139 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 1–18.

Bellare, Mihir. 1998. “Practice-Oriented Provable Security.” In *Lectures on data security: modern cryptology in theory and practise*, edited by Ivan Damgård, Volume 1561 of *Lecture Notes in Computer Science*, 1–15. Springer-Verlag, Berlin Germany.

Bellare, Mihir, and Oded Goldreich. 1993. “On Defining Proofs of Knowledge.” Edited by E.F. Brickell, *Advances in Cryptology – CRYPTO ’92*, Volume 740 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 390–420.

Bellare, Mihir, and Shafi Goldwasser. 2001. Lecture Notes on Cryptography. `http://www-cse.ucsd.edu/users/mihir/papers/gb.html`. Course notes of summer course Cryptography and Computer Security at MIT, 1996–2001.

Bellare, Mihir, and Phillip Rogaway. 1993, November. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols.” Edited by Victoria Ashby, *Proceedings of the 1st ACM Conference on Computer and Communications Security*. Fairfax, Virginia: ACM Press, 62–73.

———. 1995. “Optimal Asymmetric Encryption — How to encrypt with RSA.” Edited by A. De Santis, *Advances in Cryptology – EUROCRYPT ’94*, Volume 950 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 92–111. Final (revised) version appeared November 19, 1995. Available from `http://www-cse.ucsd.edu/users/mihir/papers/oaep.html`.

Benaloh, Josh. 1994, May. “Dense probabilistic encryption.” *First Annual Workshop on Selected Areas in Cryptography*. Kingston, ON, 120–128.

Biehl, Ingrid, and Bernd Meyer. 1997. “Protocols for Collusion-Secure Asymmetric Fingerprinting.” *14th Symposium on Theoretical Aspects of Computer Science (STACS’97)*, Volume 1281 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 399–412.

Biham, Eli, Dan Boneh, and Omer Reingold. 1999. “Breaking Generalized Diffie-Hellman modulo a composite is no easier than factoring.” *Information Processing Letters* 70:83–87. Also appeared in Theory of Cryptography Library, Record 97-14, 1997.

Blakley, G. R., C. Meadows, and G. B. Purdy. 1986. “Fingerprinting long forgiving Messages.” Edited by Hugh C. Williams, *Advances in Cryptology – CRYPTO ’85*, Volume 218 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 180.

Blum, Manuel, and Silvio Micali. 1984. “How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits." *SIAM Journal on Computing* 13 (4): 850–864 (November).

Bonatti, Piero, Sabrina de Capitani di Vimercati, and Pierangela Samarati. 2002. "An Algebra for Composing Access Control Policies." *ACM Transactions on Information and System Security* 5 (1): 1–35 (February).

Boneh, Dan. 1998. "The Decision Diffie-Hellman problem." *Third International Algorithmic Number Theory Symposium (ANTS-III)*, Volume 1423 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 48–63.

———. 2000, October. Personal Communication.

———. 2001. "Simplified OAEP for the RSA and Rabin functions." Edited by Joe Kilian, *Advances in Cryptology – CRYPTO '2001*, Volume 2139 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 275–291.

Boneh, Dan, and Matthew Franklin. 1999. "An Efficient Public Key Traitor Scheme (Extended Abstract)." Edited by Michael Wiener, *Advances in Cryptology – CRYPTO '99*, Volume 1666 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 338–353.

Boneh, Dan, and Richard J. Lipton. 1996. "Algorithms for black box fields and their application to cryptography." Edited by Neal Koblitz, *Advances in Cryptology – CRYPTO '96*, Volume 1109 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 283–297.

Boneh, Dan, and James Shaw. 1994. "Collusion-Secure Fingerprinting for Digital Data." Technical Report TR-468-94, DCS, Princeton University, NJ 08544. Submitted for patent application.

———. 1995. "Collusion-Secure Fingerprinting for Digital Data." Edited by Don Coppersmith, *Advances in Cryptology – CRYPTO '95*, Volume 963 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 452–465.

———. 1998. "Collusion-Secure Fingerprinting for Digital Data." *IEEE Transactions on Information Theory* 44 (5): 1897–1905.

Boudot, Fabrice. 2000. "Efficient Proofs that a Committed Number Lies in an Interval." Edited by Bart Preneel, *Advances in Cryptology – EUROCRYPT '2000*, Volume 1807 of *Lecture Notes in Computer Science*. Brugge, Belgium: Springer-Verlag, Berlin Germany, 431–444.

Boyar, J., K. Friedl, and C. Lund. 1991. "Practical zero-knowledge proofs: Giving hints and using deficiencies." *Journal of Cryptology* 4 (3): 185–206.

Boyar, Joan F., Stuart A. Kurtz, and Mark W. Krentel. 1990. "A Discrete Logarithm Implementation of Perfect Zero-Knowledge Blobs." *Journal of Cryptology* 2 (2): 63–76.

Brands, Stefan. 1993, March. "An Efficient Off-line Electronic Cash System Based On The Representation Problem." Technical Report CS-R9323, Centrum voor Wiskunde en Informatica.

———. 1994. "Untraceable Off-line Cash in Wallet with Observers." Edited by Douglas R. Stinson, *Advances in Cryptology – CRYPTO '93*, Volume 773 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 302–318.

Brassard, Gilles, David Chaum, and Claude Crépeau. 1988. "Minimum Disclosure Proofs of Knowledge." *Journal of Computer and System Sciences* 37 (2): 156–189 (October).

Brickell, Ernie, Peter Gemmell, and David Kravitz. 1995, January. "Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change." *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95).* Sandia National Labs, 457–466.

Cachin, Christian, Klaus Kursawe, and Victor Shoup. 2000, July. "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography." *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing.* ACM, Portland, Oregon. Full version appeared as Cryptology ePrint Archive Report 2000/034 (2000/7/7).

Camenisch, Jan. 1998. "Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem." Ph.D. diss., ETH Zürich. Dissertation ETH No. 12520, Vol. 2 of ETH-Series in Information Security an Cryptography, ISBN 3-89649-286-1, Hartung-Gorre Verlag, Konstanz.

———. 2000. "Efficient Anonymous Fingerprinting with Group Signatures." Edited by T. Okamoto, *Advances in Cryptology – ASIACRYPT '2000*, Volume 1976 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research, Kyoto, Japan: Springer-Verlag, Berlin Germany, 415–428.

Camenisch, Jan, Ueli Maurer, and Markus Stadler. 1996, September. "Digital Payment Systems with Passive Anonymity-Revoking Trustees."

Edited by E. Bertino, H. Kurth, G. Martella, and E. Montolivo, *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS)*, Volume 1146 of *Lecture Notes in Computer Science.* Rome, Italy: Springer-Verlag, Berlin Germany, 33–43.

Camenisch, Jan, and Markus Michels. 1999a. “Proving in Zero-Knowledge that a Number is the Product of Two Safe Primes.” Edited by Jacques Stern, *Advances in Cryptology – EUROCRYPT ’99*, Volume 1599 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 107–122.

———. 1999b. “Separability and Efficiency for Generic Group Signature Schemes.” Edited by Michael Wiener, *Advances in Cryptology – CRYPTO ’99*, Volume 1666 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 413–430.

Camenisch, Jan, and Markus Stadler. 1997, March. “Proof Systems for General Statements about Discrete Logarithms.” Technical report TR 260, Department of Computer Science, ETH Zürich.

Canetti, Ran. 1997. “Towards realizing random oracles: Hash functions that hide all partial information.” Edited by Burton S. Kaliski, Jr., *Advances in Cryptology – CRYPTO ’97*, Volume 1294 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 455–469.

Canetti, Ran, Oded Goldreich, and Shai Halevi. 1998, May. “The Random Oracle Methodology, Revisited.” *Proceedings of the 30th Annual Symposium on Theory Of Computing (STOC).* Dallas, TX, USA: ACM Press, 209–218.

Chaum, D., J.-H. Evertse, and J. van de Graaf. 1988. “An improved protocol for demonstrating possession of discrete logarithms and some generalizations.” Edited by David Chaum and Wyn L. Price, *Advances in Cryptology – EUROCRYPT ’87*, Volume 304 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 127–141.

Chaum, David. 1983. Blind Signature Systems. Abstact/Plenum.

———. 1989. “Privacy Protected Payments: Unconditional Payer and/or Payee Untraceability.” Edited by D. Chaum and I. Schaumueller-Bichl, *Smartcard 2000.* Elsevier Science Publisher North Holland, 69–93.

Chaum, David, A. Fiat, and M. Naor. 1990. “Untraceable Electronic Cash.” Edited by Shafi Goldwasser, *Advances in Cryptology – CRYPTO ’88*, Volume 403 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research, Santa Barbara, CA, USA: Springer-Verlag, Berlin Germany, 319–327.

Chaum, David, and Torben Pryds Pedersen. 1993. "Wallet Databases with Observers." Edited by E.F. Brickell, *Advances in Cryptology – CRYPTO '92*, Volume 740 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 89–105.

Chaum, David, and Hans van Antwerpen. 1990. "Undeniable Signatures." Edited by Giles Brassard, *Advances in Cryptology – CRYPTO '89*, Volume 435 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research, Santa Barbara, CA, USA: Springer-Verlag, Berlin Germany.

Chaum, David L. 1985. "Security without identification: transaction systems to make big brother obsolete." *Communications of the ACM* 28 (10): 1030–1044 (October).

Chor, Benny, Amos Fiat, and Moni Naor. 1994. "Tracing Traitors." Edited by Yvo G. Desmedt, *Advances in Cryptology – CRYPTO '94*, Volume 839 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 257–270.

Cox, Brad. 1996. *Superdistribution, Objects as Property on the Electronic Frontier.* Addison-Wesley.

Cox, Ingemar, Joe Kilian, Tom Leighton, and Talal Shamoon. 1996, May/June. "A Secure, Robust Watermark for Multimedia." Edited by Ross Anderson, *Information Hiding—First International Workshop, IH'96*, Volume 1174 of *Lecture Notes in Computer Science.* Cambridge, U.K.: Springer-Verlag, Berlin Germany, 175–190.

———. 1997. "Secure Spread Spectrum Watermarking for Multimedia." *IEEE Transactions on Image Processing* 6 (12): 1673–1687.

Cox, Ingemar, Matthew L. Miller, and Jefferey A. Bloom. 2002. *Digital Watermarking.* Morgan Kaufmann Publisher.

Cox, Ingemar J., Matthew L. Miller, and Andrew L. McKellips. 1999. "Watermarking as Communications With Side Information." *Proceedings of the IEEE* 87 (7): 1127–1141 (July).

Cox, Ingemar J., and Matt L. Miller. 1997, February. "A review of watermarking and the importancs of perceptual modeling." *Human Vision and Electronic Imaging II, SPIE*, Volume 3016. San Jose, CA, USA, 92–99.

Cramer, Ronald, and Ivan Damgård. 1996. "New Generation of Secure and Practical RSA-Based Signatures." Edited by Neal Koblitz, *Advances in Cryptology – CRYPTO '96*, Volume 1109 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 173–185.

Cramer, Ronald, and Victor Shoup. 1998. "A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack." Edited by Hugo Krawczyk, *Advances in Cryptology – CRYPTO '98*, Volume 1462 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 13–25.

———. 2000. "Signature Schemes Based on the Strong RSA Assumption." *ACM Transactions on Information and System Security* 3 (3): 161–185 (August).

Craver, Scott, Boon-Lock Yeo, and Minerva Yeung. 1998. "Technical Trials and Legal Tribulations." *Communications of the ACM* 41 (7): 45–54.

Damgård, Ivan. 1998. "Commitment Schemes and Zero-Knowledge Protocols." In *Lectures on data security: modern cryptology in theory and practise*, edited by Ivan Damgård, Volume 1561 of *Lecture Notes in Computer Science*, 63–86. Springer-Verlag, Berlin Germany.

Damgård, Ivan Bjerre, and Maciej Koprowski. 2002, February. "Generic Lower Bounds for Root Extraction and Signature Schemes in General Groups." Report 2002/013, Cryptology ePrint Archive.

Davida, George, Yair Frankel, Yiannis Tsiounis, and Moti Yung. 1997, February. "Anonymity Control in E-Cash Systems." *Proceedings of the First Conference on Financial Cryptography (FC '97)*, Volume 1318 of *Lecture Notes in Computer Science*. International Financial Cryptography Association (IFCA), Anguilla, British West Indies: Springer-Verlag, Berlin Germany, 1–16.

Diffie, Whitfield, and Martin Hellman. 1976. "New Directions in Cryptography." *IEEE Transactions on Information Theory* IT-22 (6): 644–654 (November).

Dobbertin, Hans, Antoon Bosselaers, and Bart Preneel. 1996. "RIPEMD-160: A strengthened version of RIPEMD." *Proceedings of the 3rd International Workshop on Fast Software Encryption*, Volume 1039 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 71–82.

Domingo-Ferrer, Josep. 1999, March. "Anonymous Fingerprinting Based on Committed Oblivious Transfer." Edited by H. Imai and Z. Zheng, *International Workshop on Practice and Theory in Public Key Cryptography '99 (PKC '99)*, Volume 1560 of *Lecture Notes in Computer Science*. Kamakura, Japan: Springer-Verlag, Berlin Germany, 43–52.

Domingo-Ferrer, Josep, and Jordi Herrera-Joancomarti. 1998, September. "Efficient Smart-Card Based Anonymous Fingerprinting." Edited by J.-J. Quisquater and B. Schneier, *Third Smart Card Research and*

Advanced Application Conference. Louvain-la-Neuve, Belgium. PreProceedings.

Eggers, J. J., J. K. Su, and B. Girod. 2000, September. "Asymmetric Watermarking Schemes." *Sicherheit in Netzen und Medienströmen*, Springer Reihe, Informatik Aktuell.

ElGamal, Taher. 1985. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." *IEEE Transactions on Information Theory* IT-31 (4): 469–472 (July).

England, Paul, Butler Lampson, John Manferdelli, Marcus Peinado, and Bryan Willman. 2003. "A Trusted Open Platform." *IEEE Computer* 36 (7): 55–63.

Ergun, Funda, Joe Kilian, and Ravi Kumar. 1999. "A Note on the Limits of Collusion-Resistant Watermarks." Edited by Jacques Stern, *Advances in Cryptology – EUROCRYPT '99*, Volume 1599 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 354–371.

Feller, William. 1968. *An Introduction to Probability Theory and Its Applications*. Third. Volume I of *Wiley Series in Probability and Mathematical Statistics*. New York: John Wiley & Sons.

Fiat, Amos, and Moni Naor. 1994. "Broadcast Encryption." Edited by Douglas R. Stinson, *Advances in Cryptology – CRYPTO '93*, Volume 773 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 480–491.

Fiat, Amos, and Adi Shamir. 1987. "How to prove Yourself: Practical Solutions to Identification and Signature Problems." Edited by A. M. Odlyzko, *Advances in Cryptology – CRYPTO '86*, Volume 263 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research, Santa Barbara, CA, USA: Springer-Verlag, Berlin Germany, 186–194.

Fiat, Amos, and Tamir Tassa. 1999. "Dynamic Traitor Tracing." Edited by Michael Wiener, *Advances in Cryptology – CRYPTO '99*, Volume 1666 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 354–371.

Fischlin, Marc. 2000. "A Note on Security Proofs in the Generic Model." Edited by T. Okamoto, *Advances in Cryptology – ASIACRYPT '2000*, Volume 1976 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research, Kyoto, Japan: Springer-Verlag, Berlin Germany, 458–469.

Frankel, Yair, Yiannis Tsiounis, and Moti Yung. 1996. ""Indirect Discourse Proofs": Achieving Fair Off-Line Cash (FOLC)." Edited by K. Kim and T. Matsumoto, *Advances in Cryptology – ASIACRYPT '96*, Volume 1163 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 286–300.

———. 1998. "Fair Off-Line e-Cash Made Easy." Edited by K. Ohta and D. Pei, *Advances in Cryptology – ASIACRYPT '98*, Volume 1514 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 257–270.

Franklin, M., and M. Yung. 1993. "Secure and efficient off-line digital money." *20th International Colloquium on Automata, Languages and Programming (ICALP '93)*, Volume 700 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 256–276.

Fujisaki, Eiichiro, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. 2001. "RSA—OAEP is secure under the RSA Assumption." Edited by Joe Kilian, *Advances in Cryptology – CRYPTO '2001*, Volume 2139 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 260–274.

Furon, Teddy, and Pierre Duhamel. 2000, October. "An Asymmetric Public Detection Watermarking Technique." Edited by Andreas Pfitzmann, *Information Hiding—3rd International Workshop, IH'99*, Volume 1768 of *Lecture Notes in Computer Science*. Dresden, Germany: Springer-Verlag, Berlin Germany, 88–100.

Gennaro, Rosario. 2000. "An Improved Pseudo-random Generator Based on Discrete Log." Edited by Mihir Bellare, *Advances in Cryptology – CRYPTO '2000*, Volume 1880 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 469–481.

Goldreich, Oded. 2001a, November. "Concurrent Zero-Knowledge With Timing, Revisited." Report 2001/104, Cryptology ePrint Archive.

———. 2001b. *Foundations of Cryptography*. Volume Basic Tools. Cambridge University Press.

Goldreich, Oded, and Hugo Krawczyk. 1996. "On the composition of Zero-Knowledge Proof Systems." *SIAM Journal on Computing* 25 (1): 169–192.

Goldreich, Oded, Silvio Micali, and Avi Wigderson. 1991. "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems." *Journal of the ACM* 38 (3): 690–728 (July).

Goldreich, Oded, and Yair Oren. 1994. "Definitions and Properties of Zero-Knowledge Proof Systems." *Journal of Cryptology* 7 (1): 1–32.

Goldwasser, Shafi, and Silvio Micali. 1984. "Probabilistic Encryption." *Journal of Computer Security* 28:270–299.

Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. 1985, May. "The Knowledge Complexity of Interactive Proof Systems." *Proceedings of the 17th Annual Symposium on Theory of Computing (STOC).* Providence, RI USA: ACM Press, 291–304.

———. 1989. "The Knowledge Complexity of Interactive Proof Systems." *SIAM Journal on Computing* 18 (1): 186–208.

Goldwasser, Shafi, Silvio Micali, and Ron L. Rivest. 1988. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks." *SIAM Journal on Computing* 17 (2): 281–308 (April).

Gordon, Daniel M. 1993a. "Designing and Detecting Trapdoors for Discrete Log Cryptosystems." Edited by E.F. Brickell, *Advances in Cryptology – CRYPTO '92*, Volume 740 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 66–75.

———. 1993b. "Discrete logarithms in GF(p) using the number field sieve." *SIAM Journal on Discrete Mathematics* 6 (1): 124–138.

Goto, Hideaki, Masahiro Mambo, Hiroki Shizuya, and Yasuyoshi Watanabe. 2001, July. "Evaluation of Tamper-Resistant Software Deviating from Structured Programming Rules." Edited by V. Varadharajan and Y. Mu, *Information Security and Privacy — 6th Australasian Conference, ACISP 2001*, Volume 2119 of *Lecture Notes in Computer Science.* Sydney, Australia: Springer-Verlag, Berlin Germany, 145–158.

Group, infoMarket Business Development. 1997. "Cryptolope Containers." A white paper, IBM.

Gunter, C., S. Weeks, and A. Wright. 2001. "Models and Languages for Digital Rights." *34th Annual Hawaii International Conference on System Sciences.* IEEE Computer Society.

Hada, Satoshi. 2000. "Zero-Knowledge and Code Obfuscation." Edited by T. Okamoto, *Advances in Cryptology – ASIACRYPT '2000*, Volume 1976 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research, Kyoto, Japan: Springer-Verlag, Berlin Germany, 443–457.

Handschuh, Helena, Yiannis Tsiounis, and Moti Yung. 1999, March. "Decision oracles are equivalent to matching oracles." Edited by H. Imai and Z. Zheng, *International Workshop on Practice and Theory in Public Key Cryptography '99 (PKC '99)*, Volume 1560 of *Lecture Notes in Computer Science.* Kamakura, Japan: Springer-Verlag, Berlin Germany, 276–289.

Hartung, Frank, and Martin Kutter. 1999. "Multimedia Watermarking Techniques." *Proceedings of the IEEE, Special Issue on Identification and Protection of Multimedia Information* 87 (7): 1079–1107 (July).

Herrigel, Alexander, Joseph Ó Ruanaidh, Holger Petersen, Shelby Pereira, and Thierry Pun. 1998, April. "Secure Copyright Protection Techniques for Digital Images." Edited by David Aucsmith, *Information Hiding—Second International Workshop, IH'98*, Volume 1525 of *Lecture Notes in Computer Science.* Portland Oregon, USA: Springer-Verlag, Berlin Germany, 169–190.

Jajodia, S., P. Samarati, and V.S. Subrahmanian. 1997, May. "A Logical Language for Expressing Authorizations." *Proceedings of the IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society, Technical Committee on Security and Privacy, Oakland, CA: IEEE Computer Society Press, 31–42.

Kaliski, Jr., Burton S. 1998. "Emerging Standards for Public-Key Cryptography." In *Lectures on data security: modern cryptology in theory and practise*, edited by Ivan Damgård, Volume 1561 of *Lecture Notes in Computer Science*, 87–104. Springer-Verlag, Berlin Germany.

Katzenbeisser, Stefan, and Fabien A.P. Petitcolas. 2000. *Information Hiding: techniques for steganography and digital watermarking.* Artech House Publishers.

Kilian, J., T. Leighton, L. R. Matheson, T. G. Shamoon, R. E. Tarjan, and F. Zane. 1998. "Resistance of digital watermarks to collusive attacks." Technical report TR-585-98, Department of Computer Science, Princeton University.

Kiltz, Eike. 2001. "A Tool Box of Cryptographic Functions related to the Diffie-Hellman Function." *Advances in Cryptology – INDOCRYPT '2001*, Volume 2247 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin Germany, 339–350.

Koblitz, Neal. 1987. *A Course in Number Theory and Cryptography.* Graduate Texts in Mathematics no. GTM 114. Berlin: Springer-Verlag, Berlin Germany.

Kundur, Deepa, and Dimitrios Hatzinakos. 1998, May. "Digital watermarking using multiresolution wavelet decomposition." *International Conference on Acoustic, Speech and Signal Processing (ICASP)*, Volume 5. Seattle, WA, USA, 2969–2972.

Kurosawa, Kaoru, and Yvo Desmedt. 1998. "Optimum Traitor Tracing and Asymmetric Schemes." Edited by Kaisa Nyberg, *Advances in Cryptology – EUROCRYPT '98*, Volume 1403 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 145–157.

Kutter, Martin. 1998, November. "Watermarking resisting to translation, rotation, and scaling." *Proceedings of SPIE.* Boston, USA.

Laboratory, National Institute of Standards and Technology (NIST)Computer Systems. 1995, April. Secure Hash Standard. Federal Information Processing Standards Publication (FIPS PUB) 180-1.

Lenstra, A. K., and H. W. Lenstra. 1993. "The Development of the number field sieve." *Lecture Notes in Mathematics*, Volume 1554. Springer-Verlag, Berlin Germany.

Lenstra, Arjen K. 2000. "Integer Factoring." *Designs, Codes and Cryptography* 19:101–128.

———. 2001, August. Computational Methods in Public Key Cryptology. `http://www.cryptosavvy.com/notes.ps`.

Lenstra, Arjen K., and Eric R. Verheul. 2001. "Selecting Cryptographic Key Sizes." *Journal of Cryptology* 14 (4): 255–293.

Lidl, Rudolf, and Harald Niederreiter. 1997, January. *Finite Fields*. Second edition. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Lin, Ching-Yung, and Shih-Fu Chang. 1998, September. "Generating Robust Digital Signature for Image/Video Authentication." *ACM Multimedia*. Bristol, UK, 49–54.

MacKenzie, Philip. 2001, July. "On the Security of the SPEKE Password-Authenticated Key Exchange Protocol." Report 2001/057, Cryptology ePrint Archive.

Maurer, Ueli M. 1994. "Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms." Edited by Yvo G. Desmedt, *Advances in Cryptology – CRYPTO '94*, Volume 839 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 271–281.

———. 2001. "Cryptography 2000±10." In *Informatics – 10 Years Back, 10 Years Ahead*, edited by Reinhard Wilhelm, Volume 2000 of *Lecture Notes in Computer Science*, 63–85. Springer-Verlag, Berlin Germany.

Maurer, Ueli M., and Stefan Wolf. 1996. "Diffie-Hellman Oracles." Edited by Neal Koblitz, *Advances in Cryptology – CRYPTO '96*, Volume 1109 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 268–282.

———. 1998a, August. "Diffie-Hellman, Decision Diffie-Hellman, and Discrete Logarithms." *IEEE Symposium on Information Theory*. Cambridge, USA, 327.

———. 1998b. "Lower bounds on generic algorithms in groups." Edited by Kaisa Nyberg, *Advances in Cryptology – EUROCRYPT '98*, Volume 1403 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 72–84.

McCurley, Kevin S. 1990. "The Discrete Logarithm Problem." Edited by Carl Pomerance, *Cryptology and Computational Number Theory*, Volume 42 of *Proceedings of Symposia in Applied Mathematics.* American Mathematical Society, Providence, 49–74.

Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone. 1997. *Handbook of Applied Cryptography.* CRC Press series on discrete mathematics and its applications. CRC Press. ISBN 0-8493-8523-7.

Mori, Ryoichi, and Masaji Kawahara. 1990, July. "Superdistribution: the concept and the architecture." Technical Report 7, Inst. of Inf. Sci. & Electron (Japan), Tsukuba Univ., Japan.

Muratani, Hirofumi. 2001. "A Collusion-Secure Fingerprinting Code Reduced by Chinese Remindering and its Random-Error Resilience." Edited by Ira S. Moskowitz, *Information Hiding—4th International Workshop, IHW 2001*, Volume 2137 of *Lecture Notes in Computer Science.* Pittsburgh, PA, USA: Springer-Verlag, Berlin Germany, 303–315.

Naccache, David, and Jacques Stern. 1998, November. "A New Public Key Cryptosystem Based on Higher Residues." *Proceedings of the 5th ACM Conference on Computer and Communications Security.* San Francisco, California: ACM Press, 59–66.

Naor, Moni, and Benny Pinkas. 1998. "Threshold Taitor Tracing." Edited by Hugo Krawczyk, *Advances in Cryptology – CRYPTO '98*, Volume 1462 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 502–518.

National Institute of Standards and Technology (NIST). 2000, January. The Digital Signature Standard (DSS). Federal Information Processing Standards Publication (FIPS PUB) 186-2. updated 2001-10-05.

National Research Council. 2000. *The Digital Dilemma, Intellectual Property in the Information Age.* National Academy Press.

Nechaev, V. I. 1994. "Complexity of a determinate algorithm for the discrete logarithm." *Mathematical Notes* 55 (2): 165–172. Translated from Matematicheskie Zametki, 55(2):91–101, 1994.

Odlyzko, Andrew. 2000. "Discrete logarithms: The past and the future." *Designs, Codes and Cryptography* 19:129–145.

Okamoto, Tatsuaki, and Shigenori Uchiyama. 1998. "A New Public-key Cryptosystem as Secure as Factoring." Edited by Kaisa Nyberg, *Advances in Cryptology – EUROCRYPT '98*, Volume 1403 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 308–318.

Patel, Sarvar, and Ganapathy S. Sundaram. 1998. "An Efficient Discrete Log Pseudo Random Generator." Edited by Hugo Krawczyk, *Advances in Cryptology – CRYPTO '98*, Volume 1462 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 304–317.

Pedersen, Torben Pryds. 1992. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing." Edited by Joan Feigenbaum, *Advances in Cryptology – CRYPTO '91*, Volume 576 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 129–140. Extended abstract.

Perrig, Adrian, Alexander Herrigel, and Joseph Ruanaidh. 1997. "A Copyright Protection Environment for Digital Images." *Verlässliche IT-Systeme, GI-Fachtagung VIS '97, DuD Fachbeiträge*. Vieweg, 1–16.

Petitcolas, Fabien A. P., Ross Anderson, and Markus G. Kuhn. 1998, April. "Attacks on Copyright Marking Systems." Edited by David Aucsmith, *Information Hiding—Second International Workshop, IH'98*, Volume 1525 of *Lecture Notes in Computer Science*. Portland Oregon, USA: Springer-Verlag, Berlin Germany, 218–238.

Petitcolas, Fabien A. P., and Ross J. Anderson. 1998, September. "Weaknesses of copyright marking systems." *ACM Multimedia*. Bristol, UK, 55–61.

Petitcolas, Fabien A. P., Ross J. Anderson, and Markus G. Kuhn. 1999. "Information hiding — A survey." *Proceedings of the IEEE* 87 (7): 1062–1078 (July). Special issue on protection of multimedia content.

Pfitzmann, Birgit. 1996a. *Digital Signature Schemes — General Framework and Fail-Stop Signatures*. Volume 1100 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany.

———. 1996b, May/June. "Information Hiding Terminology." Edited by Ross Anderson, *Information Hiding—First International Workshop, IH'96*, Volume 1174 of *Lecture Notes in Computer Science*. Cambridge, U.K.: Springer-Verlag, Berlin Germany, 347–350.

———. 1999, July. Higher cryptographic protocols (Höhere kryptographische Protokolle). Vorlesungsskript.

Pfitzmann, Birgit, and Ahmad-Reza Sadeghi. 1999. "Coin-Based Anonymous Fingerprinting." Edited by Jacques Stern, *Advances in Cryptology – EUROCRYPT '99*, Volume 1599 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 150–164.

———. 2000. "Anonymous Fingerprinting with Direct Non-Repudiation." Edited by T. Okamoto, *Advances in Cryptology – ASIACRYPT '2000*, Volume 1976 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research, Kyoto, Japan: Springer-Verlag, Berlin Germany, 401–414.

Pfitzmann, Birgit, and Matthias Schunter. 1996. "Asymmetric Fingerprinting (Extended Abstract)." Edited by Ueli Maurer, *Advances in Cryptology – EUROCRYPT '96*, Volume 1070 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 84–95.

Pfitzmann, Birgit, and Michael Waidner. 1997a. "Anonymous Fingerprinting." Edited by Walter Fumy, *Advances in Cryptology – EUROCRYPT '97*, Volume 1233 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 88–102.

———. 1997b, April. "Asymmetric Fingerprinting for Larger Collusions." Edited by Tsutomu Matsumoto, *Proceedings of the 4th ACM Conference on Computer and Communications Security*. Zurich, Switzerland: ACM Press, 151–160.

———. 1997c. "Strong Loss Tolerance of Electronic Coin Systems." *ACM Transactions on Computer Systems* 15 (2): 194–213 (May).

———. 1998. "How to Break Fraud-detectable Key Recovery." *ACM Operating Systems Review* 32 (1): 23–28 (January).

Pitas, Ioannis, and George Voyatzis. 1999. "The Use of Watermarks in the Protection of Digital Multimedia Products." *Proceedings of te IEEE, Special Issue on Identification and Protection of Digital Multimedia Products* 87 (7): 1197–1207 (July).

Pohlig, Stephen C., and Martin E. Hellman. 1978. "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance." *IEEE Transactions on Information Theory* 24:106–110.

Pointcheval, David, and Jacques Stern. 2000. "Security Arguments for Digital Signatures and Blind Signatures." *Journal of Cryptology* 13 (3): 361–396 (March).

Pollard, J. M. 1978. "Monte Carlo methods for index computation mod p." *Mathematics of Computation* 32:918–924.

Preneel, Bart. 1998. "The State of Cryptographic Hash Functions." In *Lectures on data security: modern cryptology in theory and practise*, edited by Ivan Damgård, Volume 1561 of *Lecture Notes in Computer Science*, 158–182. Springer-Verlag, Berlin Germany.

Reed, I. S., and G. Solomon. 1960. "Polynomial Codes over Certain Finite Fields." *Journal of Society for Industrial and Applied Mathematics* 8:300–304.

Rivest, Ron. 1992, April. "The MD5 Message-Digest Algorithm." Internet request for comment RFC 1321, Internet Engineering Task Force.

Rivest, Ron L., Adi Shamir, and Leonard M. Adleman. 1978. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Communications of the ACM* 21 (2): 120–126 (February).

Ruanaidh, J. J. K. 'O, F. M. Boland, and O. Sinnen. 1995. Watermarking Digital Images for Copyright Protection. Department of Electronic and Electrical Engineering, Trinity College.

Sadeghi, Ahmad-Reza. 2001. "How to Break a Semi-Anonymous Fingerprinting Scheme." Edited by Ira S. Moskowitz, *Information Hiding—4th International Workshop, IHW 2001*, Volume 2137 of *Lecture Notes in Computer Science*. Pittsburgh, PA, USA: Springer-Verlag, Berlin Germany, 384–394.

Sadeghi, Ahmad-Reza, and Michael Steiner. 2001. "Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference." Edited by Birgit Pfitzmann, *Advances in Cryptology – EUROCRYPT '2001*, Volume 2045 of *Lecture Notes in Computer Science*. Innsbruck, Austria: Springer-Verlag, Berlin Germany, 243–260.

Sadeghi, Ahmad-Reza, and Christian Stüble. 2003. "Taming "Trusted Computing" by Operating System Design." *Information Security Applications*, Volume 2908 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 286–302.

Schirokauer, Oliver. 1993. "Discrete logarithms and local units." *Philosophical Transactions of the Royal Society of London* A 345:409–423.

Schnorr, Claus P. 1991. "Efficient Signature Generation by Smart Cards." *Journal of Cryptology* 4 (3): 161–174.

———. 1992, February. "Comparison of the DSA of NIST and the Signature Schemes of ElGamal and Schnorr." Edited by Bruno Struif, *2. GMD SmartCard Workshop*.

Schwartz, J. T. 1980. "Fast probabilistic algorithms for verification of polynomial identities." *Journal of the ACM* 27 (4): 701–717 (October).

Shaw, James H. 2000, June. "Information Theoretical Protection for Intellectual Property." Ph.D. diss., Princeton University.

Shmuely, Zahava. 1985, February. "Composite Diffie-Hellman Public-Key Generating Systems are Hard to Break." Computer science technical report 356, Israel Institute of Technology (Technion).

Shoup, Victor. 1997. "Lower Bounds for Discrete Logarithms and Related Problems." Edited by Walter Fumy, *Advances in Cryptology – EUROCRYPT '97*, Volume 1233 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 256–266.

———. 1999, April. "On Formal Models for Secure Key Exchange." Research report RZ 3120 (#93166), IBM Research. A revised version 4, dated November 15, 1999, is available from `http://www.shoup.net/papers/`.

———. 2001. "OAEP Reconsidered." Edited by Joe Kilian, *Advances in Cryptology – CRYPTO '2001*, Volume 2139 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 239–259.

Singh, Simon. 1999. *The Code Book — The Secret History of Codes and Code-breaking*. London, UK: Fourth Estate Ltd.

Steiner, Michael, Gene Tsudik, and Michael Waidner. 1996, March. "Diffie-Hellman Key Distribution Extended to Groups." Edited by Clifford Neuman, *Proceedings of the 3rd ACM Conference on Computer and Communications Security*. New Delhi, India: ACM Press, 31–37.

Swanson, Mitchell D., Bin Zhu, and Ahmed H. Tewfik. 1998. "Multiresolution Scene-Based Video Watermarking Using Perceptual Models." *IEEE Journal on Selected Areas in Communications* 16 (4): 540–550.

Tirkel, A. Z., G. A. Rankin, R. van Schyndel, W. J. Ho, N. Mee, and C. F. Osborne. 1993. "Electronic Watermark." *Digital Image Computing, Technology and Applications*. Syndey, Australia, 666–672.

Trusted Computing Platform Alliance (TCPA). 2002, February. Main Specification. Version 1.1b.

Tsiounis, Yiannis, and Moti Yung. 1998, February. "On the security of ElGamal-based encryption." *International Workshop on Practice and Theory in Public Key Cryptography '98 (PKC '98)*, Volume 1431 of *Lecture Notes in Computer Science*. Yokohama, Japan: Springer-Verlag, Berlin Germany, 117–134.

US Copyright Office. 2002. US Copyright Law. `http://www.loc.gov/copyright/`.

van de Graaf, J., and R. Peralta. 1988. "A simple and secure way to show the validity of your public key." Edited by Carl Pomerance, *Advances in Cryptology – CRYPTO '87*, Volume 293 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research, Santa Barbara, CA, USA: Springer-Verlag, Berlin Germany, 128–134.

van Lint, J.H. 1999. *Introduction to Coding Theory*. Springer-Verlag, Berlin Germany.

Vanstone, Scott A., and Paul C. van Oorschot. 1995. *An Introduction to*

Error-Correcting Codes with Applications. Kluwer Academic Publishers.

Wagner, N. 1983. “Fingerprinting.” *Proceedings of the IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society, Technical Committee on Security and Privacy, Oakland, CA: IEEE Computer Society Press, 18–22.

Wang, H., F. Guo, D. D. Feng, and J. S. Jin. 1998, September. “Signature for Content-based Image Retrieval Using a Geometrical Transform.” *ACM Multimedia.* Bristol, UK, 229–233.

Wolf, Stefan. 1999. “Information-Theoretically and Computionally Secure Key Agreement in Cryptography.” Ph.D. diss., ETH Zürich.

Wolfgang, R. B., C. I. Podilchuk, and E. J. Delp. 1999. “Perceptual Watermarks for Digital Images and Video.” *Proceedings of the IEEE*, July.

Yacobi, Yacov. 2001, April. “Improved Boneh-Shaw Content Fingerprinting.” Edited by David Naccache, *Progress in Cryptology - CT-RSA 2001*, Volume 2020 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin Germany, 378–391.

Index

Appendix A

Deriving Formal Assumptions from the Parameters

The "mechanics" of deriving the formal assumption statement from its short form \$s-\$t$\$\mathcal{P}^{\$\mathrm{a}}$(c:\$c; g:\$g; f:$\$\mathcal{G}$) — as described in Section 2.3 the \$X's are placeholders of the parameters defined in Section 2.2 — is as follows:

1. **Group and problem family:** Just fix the group, generator and problem instance sampler $SG_{\mathcal{G}}$, Sg, and $SPI_{\mathcal{P}}$ corresponding to group family $\$\mathcal{G}$ and problem family $\$\mathcal{P}$, respectively. In the context of generic relations, $\$\mathcal{G}$ does normally not fix a particular group family and sampler but gives just some specific constraints on group families, e.g., groups with large prime factors indicated by "lprim". In such a case $SG_{\mathcal{G}}$ denotes an arbitrary sampler for an arbitrary group family fulfilling the given constraints on the group family and the constraints on samplers given in Section 2.1.7.[1]

2. **Problem type:** Prepare the assumption formula $\$\mathbf{F}$ as the probability statement $\$\mathbf{P}$ defined as "**Prob**[". $\$\mathbf{P}_{\mathrm{pred}}$." :: ". $\$\mathbf{P}_{\mathrm{def}}$."]". The . denotes the string-concatenation operator and the variables $\$\mathbf{P}_{\mathrm{pred}}$ and $\$\mathbf{P}_{\mathrm{def}}$ are the probability predicate and the probability space instance definition, respectively. They are defined depending on the problem type \$t as follows (where $SPI_{\mathcal{P}}$ is the problem sampler fixed in item 1 above and where the source of SI is explained in item 3 below):

   - \$t = C: Initialize $\$\mathbf{P}_{\mathrm{def}}$ to "$PI \leftarrow SPI_{\mathcal{P}}(SI);$" (the problem instance to solve) and add "$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U};$" (the random coins for the adversary) to it. Define $\$\mathbf{P}_{\mathrm{pred}}$ as "$\mathcal{A}(\mathcal{C}, SI, PI^{publ}) \in PI^{sol}$".

[1]In practice, only the later application of this relation using specific assumptions implied by a cryptographic systems will determine the concrete choices of group family and sampler.

- \$t = D: Initialize \$$\mathbf{P}_{\mathrm{def}}$ to the concatenation of "$b \overset{\mathcal{R}}{\leftarrow} \{0,1\};$" (the random bit used as challenge), "$PI_0 \leftarrow SPI_{\mathcal{P}}(SI);$" and "$PI_1 \leftarrow SPI_{\mathcal{P}}(SI);$" (the real problem instance and an auxiliary problem instance for the random public part), "$sol_c \overset{\mathcal{R}}{\leftarrow} PI_b{}^{sol};$" (one possible solution), and "$\mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U};$". \$$\mathbf{P}_{\mathrm{pred}}$ is defined as "$\mathcal{A}(\mathcal{C}, SI, PI^{publ}, sol_c) = b$". Additionally, the probability statement \$$\mathbf{P}$ is normalized to "$2 \cdot |\mathbf{Prob}[\$\mathbf{P}_{\mathrm{pred}} :: \$\mathbf{P}_{\mathrm{def}}] - 0.5|$".
- \$t = M: Initialize \$$\mathbf{P}_{\mathrm{def}}$ to the concatenation of "$b \overset{\mathcal{R}}{\leftarrow} \{0,1\};$" (the random bit used as challenge), "$PI_0 \leftarrow SPI_{\mathcal{P}}(SI);$" and "$PI_1 \leftarrow SPI_{\mathcal{P}}(SI);$" (the two problem instances to match), "$sol_0 \overset{\mathcal{R}}{\leftarrow} PI_0{}^{sol}$" and "$sol_1 \overset{\mathcal{R}}{\leftarrow} PI_1{}^{sol}$" (two corresponding solutions), and "$\mathcal{C} \overset{\mathcal{R}}{\leftarrow} \mathcal{U};$". \$$\mathbf{P}_{\mathrm{pred}}$ is defined as "$\mathcal{A}(\mathcal{C}, SI, PI_0{}^{publ}, PI_1{}^{publ}, sol_b, sol_{\bar{b}}) = b$". Additionally, the probability statement \$$\mathbf{P}$ is normalized as above to "$2 \cdot |\mathbf{Prob}[\$\mathbf{P}_{\mathrm{pred}} :: \$\mathbf{P}_{\mathrm{def}}] - 0.5|$".

3. **Granularity:** Depending on the granularity value \$g do the following (where $SG_{\mathcal{G}}$ and Sg are the group and generator sampler fixed in item 1):

   - \$g = l: Prepend "$G \leftarrow SG_{\mathcal{G}}(1^k);$", "$g_i \leftarrow Sg(G);$" (for as many $i \in \mathbb{N}$ as required by the problem family, e.g., one generator for DL and n generators for RP(n)), and "$SI \leftarrow (G, g_1, \ldots);$" to \$$\mathbf{P}_{\mathrm{def}}$.
   - \$g = m: Prepend "$\forall G \in [SG_{\mathcal{G}}(1^k)];$ to \$$\mathbf{F}$. Prepend "$g \leftarrow Sg(G);$" and "$SI \leftarrow (G, g_1, \ldots);$" to \$$\mathbf{P}_{\mathrm{def}}$.
   - \$g = h: Prepend "$\forall G \in [SG_{\mathcal{G}}(1^k)];$", "$\forall g_i \in [Sg(G)];$", and "$SI \leftarrow (G, g_1, \ldots);$" to \$$\mathbf{F}$.

4. **Computational complexity and algebraic knowledge:** Depending on the computational complexity \$c do the following:

   - \$c = u: Prefix \$$\mathbf{F}$ with "$\forall \mathcal{A} \in \mathcal{UPTM};$", "$\exists k_0;$", and "$\forall k > k_0;$".
   - \$c = n: Prefix \$$\mathbf{F}$ with "$\forall (\mathcal{A}_i \mid i \in \mathbb{N}) \in \mathcal{NPTM};$", "$\exists k_0;$", and "$\forall k > k_0;$". In \$$\mathbf{P}_{\mathrm{pred}}$ replace "$\mathcal{A}$" by "$\mathcal{A}_k$".

   If the considered assumption is in the generic model (\$$a = \sigma$) then replace everywhere "$\mathcal{A}$", $\mathcal{UPTM}$ and $\mathcal{NPTM}$ by "$\mathcal{A}^\sigma$", $\mathcal{UPTM}^\sigma$ and $\mathcal{NPTM}^\sigma$, respectively. Furthermore, append "$\sigma \overset{\mathcal{R}}{\leftarrow} \Sigma_{G,g};$" (the choice of the random encoding function) to \$$\mathbf{P}_{\mathrm{def}}$.

5. **Success probability:** Depending on the success probability \$s do the following to finish the formal assumption statement:

   - \$s = 1: Append "< 1" to \$$\mathbf{F}$, i.e., immediately after \$$\mathbf{P}$.

- \$s $= (1-1/\mathsf{poly}(k))$: Append "$\exists d_1$;" immediately after the all-quantifier on adversary algorithms in \$**F**. Append "$< (1-1/k^{d_1})$" to \$**F**.
- \$s $= \epsilon$: Append "$< \epsilon$" to \$**F**.
- \$s $= 1/\mathsf{poly}(k)$: Append "$\forall d_1$;" immediately after the all-quantifier on adversary algorithms in \$**F**. Append "$< 1/k^{d_1}$" to \$**F**.

Evaluating \$**F** by expanding the variables , i.e., \$**P**, \$$\mathbf{P}_{\mathrm{pred}}$ and \$$\mathbf{P}_{\mathrm{def}}$, and applying the string-concatenation operator gives now the desired precise formal assumption statement.