Optimistic Fair Exchange

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

Eingereicht von Matthias Schunter

Saarbrücken, October 2000

Abstract

A fair exchange guarantees that a participant only reveals its items (such as signatures, payments, or data) if it receives the expected items in exchange. Efficient fair exchange requires a so-called third party, which is assumed to be correct. Optimistic fair exchange involves this third party only if needed, i.e., if the participants cheat or disagree.

In Part I, we prove lower bounds on the message and time complexity of two particular instances of fair exchange in varying models, namely contract signing (fair exchange of two signatures under a contract) and certified mail (fair exchange of data for a receipt). We show that all given bounds are tight by describing provably time- and message-optimal protocols for all considered models and instances.

In Part II, we have a closer look at formalizing the security of fair exchange. We introduce a new formal notion of security (including secrecy) for reactive distributed systems. We illustrate this new formalism by a specification of certified mail as an alternative to the traditional specification given in Part I.

In Part III, we describe protocols for generic and optimistic fair exchange of arbitrary items. These protocols are embedded into the *SEMPER* Fair Exchange Layer, which is a central part of the *SEMPER* Framework for Secure Electronic Commerce.

Kurzzusammenfassung

Ein Austausch ist fair, wenn eine Partei die angebotenen Güter, wie zum Beispiel digitale Signaturen, Zahlungen oder Daten, nur abgibt, wenn sie die erwarteten Güter im Tausch erhält. Ohne eine als korrekt angenommene dritte Partei, welche eine mit einem Notar vergleichbare Rolle übernimmt, ist fairer Austausch nicht effizient möglich. Ein fairer Austausch heißt optimistisch, falls diese dritte Partei nur in Problemfällen am Protokoll teilnimmt.

In Teil I werden beweisbar zeit- und nachrichtenoptimale Protokolle für die Spezialfälle „elektronische Vertragsunterzeichnung“ (fairer Austausch zweier Signaturen; engl. *contract signing*) und „elektronisches Einschreiben“ (fairer Austausch von Daten gegen eine Quittung; engl. *certified mail*) von fairem Austausch vorgestellt.

Teil II beschreibt einen neuen Integritäts- und Geheimhaltungsbegriff für reaktive Systeme. Dieser basiert auf einer Vergleichsrelation „so sicher wie“, welche die Sicherheit zweier Systeme vergleicht. Ein verteiltes, reaktives System wird dann als sicher bezeichnet, wenn es so sicher wie ein idealisiertes System (engl. *trusted host*) für diesen Dienst ist. Mit diesem Formalismus geben wir eine alternative Sicherheitsdefinition von „elektronischem Einschreiben“ an, deren Semantik im Gegensatz zu der in Teil I beschriebenen Definition nun unabhängig vom erbrachten Dienst ist.

Teil III beschreibt ein Design und optimistische Protokolle für generischen fairen Austausch von zwei beliebigen Gütern und den darauf aufbauenden *SEMPER* Fair Exchange Layer. Dieser ist ein wesentlicher Baustein des *SEMPER* Framework for Secure Electronic Commerce.

Zusammenfassung

Ein zentrales Problem beim elektronischen Handel ist der faire Austausch von Gütern, wie Zahlungen, Quittungen, Daten, oder Signaturen. Im herkömmlichen Handel kann dies durch Anwesenheit beider Geschäftspartner garantiert werden, was beim elektronischen Handel in der Regel nicht möglich ist.

Diese Arbeit beschreibt verschiedene Aspekte von fairen Austauschdiensten, welche zwei Güter genau dann atomar ausliefern, wenn die zu liefernden Güter die gegenseitigen Anforderungen der Handelspartner erfüllen.

Atomizität im Betrugsfall ist nur durch Einbeziehung einer dritten Partei effizient möglich. Da dies jedoch für den Normalfall, daß beide Partner korrekt sind, nicht gewünscht ist, beschränken die in dieser Arbeit beschriebenen *optimistischen* Protokolle die Teilnahme der dritten Partei auf Fehlerfälle.

Nach einem Literaturüberblick gliedert sich die Arbeit in drei Teile.

Teil I

In Teil I wird die beweisbar optimale Effizienz zweier Spezialfälle untersucht.

Elektronische Vertragsunterzeichnung ist der faire Austausch zweier Vertragssignaturen unter einem gegebenen Vertragstext. Für alle beschriebenen Modelle benötigt ein zeitoptimales Protokoll jeweils $2k$ Nachrichten in Zeit k und ein nachrichtenoptimales Protokoll jeweils $k + 1$ Nachrichten in Zeit $k + 1$. Für synchrones Netz ergibt sich bei Eingabe von identischen Vertragstexten $k = 2$ unabhängig davon, ob die dritte Partei bei der Vertragsüberprüfung teilnimmt oder nicht. Für asynchrone Netzwerke ergibt sich bei gleichen Vertragstexten $k = 2$, falls die dritte Partei an der Vertragsüberprüfung teilnimmt, und ansonsten $k = 3$. Falls die dritte Partei bei korrekten Teilnehmern nur im Fehlerfall, nicht jedoch bei Eingabe verschiedener Vertragstexte, einbezogen werden soll, erhöht sich k jeweils um 1.

Elektronisches Einschreiben ist der faire Austausch einer Nachricht gegen eine Quittung. Es wird die Effizienz von drei Varianten untersucht: elektronisches Einschreiben mit Betreff (engl. *labeled certified mail*) überträgt die Nachricht nur, wenn Sender und Empfänger den gleichen Betreff eingeben. Herkömmliches elektronisches Einschreiben (engl. *certified mail*) überprüft die Übereinstimmung des Betreffs nicht. Trotzdem kann der Empfänger im Einzelfall entscheiden, ob er die Nachricht empfangen und eine Quittung ausstellen will. Eingeschränktes elektronisches Einschreiben (engl. *simple certified mail*) erzwingt die Auslieferung in jedem Fall.

Wir beweisen obere und untere Schranken für die Nachrichtenkomplexität von elektronischem Einschreiben mittels gegenseitiger Simulation mit elektronischer Vertragsunterzeichnung. Wir zeigen, daß nachrichtenoptimales elektronisches Einschreiben mit Betreff und nachrichtenoptimale elektronische Vertragsunterzeichnung äquivalent hinsichtlich ihrer Komplexität sind.

Desweiteren zeigen wir, daß jedes optimistische Protokoll für herkömmliches elektronisches Einschreiben ohne zusätzliche Nachrichten zu einem Protokoll für elektronisches Einschreiben mit Betreff erweitert werden kann. Abschließend beschreiben wir ein beweisbar zeit- und nachrichtenoptimales Protokoll für eingeschränktes elektronisches Einschreiben, welches faire Quittierung bereits mit zwei Nachrichten in Zeit zwei garantiert.

Teil II

Die Spezialfälle für fairen Austausch in Teil I werden auf herkömmliche Weise definiert: Integritätsanforderungen sind Aussagen über die Ein- und Ausgaben der Teilnehmer, welche auf temporale Logik abgebildet werden können. Die Geheimhaltung der Nachricht wird durch ein interaktives Spiel definiert, in welchem ein Angreifer mit dem System interagiert und anschließend eine geheime Nachricht raten muß. Eine Folge dieses Ansatzes ist, daß für jeden neuen Dienst eine neue Definition benötigt wird.

In Teil II wird daher ein neuer Sicherheitsbegriff für beliebige reaktive Systeme beschrieben, welcher bestehende nichtreaktive oder spezialisierte Ansätze verallgemeinert. Dieser Sicherheitsbegriff definiert eine Relation „mindestens so sicher wie“, welche die Sicherheit zweier Systeme vergleicht. Der Dienst eines gewünschten Systems wird dann durch die Beschreibung eines idealisierten Systems (engl. *trusted host*) spezifiziert.

Eine zentrale Beobachtung hierbei ist, daß diese idealisierten Systeme nicht nur den Dienst, sondern auch die tolerierten Schwachstellen spezifizieren müssen, um effiziente Implementierungen zu ermöglichen. Ein System, welches einzelne verschlüsselte Nachrichten über ein unsicheres Netzwerk übermittelt, ist zum Beispiel nur dann sicher, wenn das idealisierte System dem Angreifer mitteilt, ob überhaupt eine Nachricht geschickt wurde oder nicht.

Als Beispiele für diesen neuen Sicherheitsbegriff stellen wir Spezifikationen und reaktive Systeme zum Nachrichtenversand und für elektronisches Einschreiben mit Betreff vor und beweisen deren Sicherheit.

Teil III

In herkömmlichen Diensten für fairen Austausch (wie z.B. in Teil I) wird jeweils ein spezielles Protokoll für jeden Spezialfall, d.h. für je zwei Typen von elektronischen Gütern, benötigt. Dementsprechend wurde bereits eine Vielzahl solcher Protokolle für Spezialfälle wie elektronische Vertragsunterzeichnung, elektronisches Einschreiben, fairer Kauf (Austausch Daten / Zahlung), sowie faire Zahlung (Austausch Zahlung / Quittung) vorgestellt.

Um die wachsende Anzahl der benötigten Protokolle zu mindern, beschreibt Teil III Protokolle und ein Design für den *generischen* und optimistischen fairen Austausch beliebiger elektronischer Güter, die gewisse Dienste zur Fehlerbehebung bereitstellen. Da diese Fehlerbehebungsdienste unabhängig von den Gütern sind, können auch zukünftige Güter mit den beschriebenen Diensten fair ausgetauscht werden.

Dieses Design und die zugrundeliegenden Protokolle sind ein zentraler Baustein der vom EU Projekt *SEMPER* von 1995-1999 entwickelten Architektur zur Realisierung sicherer elektronischer Handelsabläufe (vgl. `www.semper.org`).

Contents

Chapter 1

Introduction and Overview

It is common in electronic commerce that two parties need to exchange some items fairly, as over a physical counter in a shop. Imagine Bob has requested a delivery from Alice, e.g., a piece of software, a translation or a legal expertise. Alice wants to deliver to Bob a file. The file represents the work of several person-hours, so Alice wants a receipt if Bob receives the file. Bob, on the other hand, only wants to issue a receipt if he received the file.

In traditional commerce, fairness of an exchange can be guaranteed to a reasonable extent. The items are either physically exchanged over a counter, or else exchanged by fixing the terms in a contract while settling arising disputes at court or real-world notaries.

In electronic commerce, no physical counter exists and contracts may be difficult to enforce for cross-border business. Furthermore, the costs of enforcing a contract may not be worth the effort.

Therefore, we solve this problem by fair exchange services. For a fair exchange, each party describes the item it offers (e.g., signatures, payments, or data) and describes the item it desires in exchange. For instance, Bob expects the desired file in exchange for a receipt whereas Alice expects a receipt in exchange for the file.

An idea for providing this service is to exchange the items like over a counter. Alice sends the result and Bob sends the receipt. This solution, however, does not guarantee fairness: If, e.g., Bob does not send the receipt, Alice still sends the result and does not obtain the expected receipt.

Therefore, in order to avoid such situations, we will describe protocols which *guarantee* fairness in any case, i.e., either both receive what they expect or else, no one gets even part of the expected item.

Guaranteeing fairness is either inefficient or allows for a high probability of errors, or else requires a third party, which is assumed to be correct.

While inefficiency and high error probabilities are often unacceptable in electronic commerce, a third party participating in all protocol executions results in a performance bottleneck. Furthermore, the third party learns more information about ongoing transactions than needed.

Therefore, we focus on the concept of *optimistic* fair exchanges. These *optimistic* fair exchanges do not involve the third party in the regular case, i.e., if both participants behave correctly and agree on what shall be exchanged. Only if something goes wrong, the third party is invoked to restore fairness.

1.1 Overview

After an overview of related work in Chapter 2, this thesis is structured in three self-contained parts:

Part I analyzes the optimal efficiency of two particular instances of optimistic fair exchange, namely contract signing and certified mail.

Part II proposes a new and generic notion of security of reactive cryptographic systems. (A reactive system interacts with its user multiple times while keeping state in between.) This formalism is illustrated by describing an alternative to the traditional specification of certified mail as described in Part I.

Part III describes a design and protocols for *generic* optimistic fair exchange of any two business items generalizing the services described in Part I.

1.1.1 Part I

In Part I, we identify tight lower bounds on the message and time complexity optimistic contract signing as well as on the message complexity of certified mail.

Contract signing protocols fairly compute a contract, i.e., a non-repudiable agreement on a given contract text. We prove tight lower bounds on the efficiency of optimistic contract signing for eight different models: We consider synchronous or asynchronous communication, verification of evidence with or without participation of the third party, and optimism on agreement (i.e., the third party is not involved if the players are correct and agree) or also on disagreement (i.e., the third party is not involved if the players are correct; no matter whether they agree or disagree).

For all models, a time-optimal protocol requires $2k$ messages in time k whereas the message-optimal protocol requires $k+1$ messages in time $k+1$. For our two weakest models (i.e., synchronous network, optimism on agreement only, with / without the third party participating in verification), we describe protocols for $k = 2$ and prove their message and time optimality. On asynchronous networks, we obtain optimal protocols for $k = 2$ if the third party participates in the verification of the contract and $k = 3$, else. For optimism on disagreement, we have shown how to adapt any optimistic protocol that is optimistic on agreement. This simulation increases k by 1.

Certified mail is a fair exchange of a message for a receipt. We identified three distinct models of optimistic certified mail: Labeled certified mail assumes that the mail is only sent if the sender and the recipient agree on the label (e.g., a subject) under which the mail shall be sent. Traditional certified mail does not include labels but still allows the recipient to refuse to participate in particular runs of the protocol. Simple certified mail does not allow the recipient to reject participation in particular protocol executions.

We prove tight lower bounds on optimistic labeled certified mail by mutual simulation with optimistic contract signing where such bounds have already been identified. The main result is that optimistic labeled certified mail cannot be more efficient than optimistic contract signing and that message-optimal optimistic contract signing protocols can be adopted to provide message-optimal optimistic labeled certified mail.

A natural objection against these results is that labeled certified mail is a more powerful model than traditional optimistic certified mail. To defeat this objection, we show that traditional optimistic certified mail cannot be more efficient since it can be adapted to provide labeled certified mail as well.

Finally, we describe a provably time- and message-optimal protocol for simple certified mail. This service provides the essential service of certified mail (namely non-repudiation and fairness) while being considerably more efficient than the traditional service of certified mail.

1.1.2 Part II

The security of the instances of fair exchange described in Part I is defined in the traditional way: Integrity is defined by requirements on the in- and outputs of the participants, which can be mapped to temporal logic [Pfit8 96, Veit 99]. The secrecy of a message is defined by a game, where the adversary tries to guess the message after executing, e.g., the certified mail protocol. A consequence of this approach is that the designer of each new service is required to design a corresponding security definition as well.

Part II defines a new generic notion of security (including secrecy) for reactive systems. This is done by defining a relation "as secure as" comparing two reactive systems: A system Sys_A is as secure as a system Sys_B if and only if an adversary attacking Sys_A cannot achieve more than an adversary attacking Sys_B.

Similar to a secure multi-party computation of a function, which is specified by the function to be computed, one can then specify the intended service by defining an idealized system offering it. Given such a so-called trusted host, a distributed implementation is secure if it is "as secure as" the trusted host specifying the intended service.

A remarkable observation is that in order to enable practical yet provably secure real-world systems, a trusted host needs to provide a specific adversary service (we call this concept "real-life" trusted hosts). This interface allows the designer to specify all acceptable vulnerabilities of the service. Consider for example a system for sending encrypted messages that does not waste bandwidth by sending messages at all times. Such a system will only be secure if the trusted host notifies the adversary about who sent messages to whom, since the adversary will usually see the (encrypted) messages on the network.

We illustrate this new notion of security by formally comparing the security of reactive systems for message encryption and certified mail with two real-life trusted hosts specifying their intended services.

1.1.3 Part III

In existing concepts for fair exchange, a particular fair exchange protocol is needed for each instance of fair exchange, i.e., for any two types of items to be exchanged. There have been a large number of proposals for particular instances of fair exchange for contract signing (exchange of two signatures), certified mail (exchange of data for the signature under a receipt), or fair purchase (exchange of data for a payment). However, new instances or new types of items require new protocols.

In order to reduce the increasing number of required protocols, Part III describes the design for *generic* and optimistic fair exchange. We propose synchronous optimistic fair exchange protocols that can be used for fairly exchanging any two business items satisfying certain conditions. The basic idea is to use transfers of business items as the foundation of fair exchange. We fix four exchange-enabling properties of transfers and argue why most items can be adapted to provide one or more of these properties. The consequence of these exchange-enabling properties is that almost any two business items can then be exchanged fairly. Furthermore, new items can be added without adding new fair exchange protocols.

These protocols are finally embedded into the design of the *SEMPER* Transfer and Fair Exchange Layer (*SEMPER* [Semp 00] stands for Secure Electronic Marketplace for Europe).

1.2 Acknowledgments

Foremost, I would like to thank my advisor Birgit Pfitzmann for ongoing support throughout this thesis. Her critical and detailed comments have been a continuous challenge to me.

I would like to thank my co-advisor Joachim Biskup for guidance and support throughout my work on this thesis as well as for the opportunity to work at his institutes in Hildesheim and Dortmund.

Without the encouragement of Michael Waidner, writing this thesis would not have been possible.

I appreciated the inspiring cooperation with the members of the IBM Zurich Research Lab. in the course of *SEMPER*. I would like to thank in particular N. Asokan, Michael Steiner, and Michael Waidner for many productive discussions on design and protocols for generic and optimistic fair exchange.

Furthermore, I would like to thank the members of my research groups, namely Joachim Biskup, Gerrit Bleumer, and Birgit Pfitzmann at the Universities of Hildesheim and Dortmund as well as Tom Beiler, Jürgen Brauckmann, Ahmad-Reza Sadeghi, and Michael Steiner at the Saarland University for helpful discussions and a nice environment where doing research was productive and fun.

Finally, I express my gratitude towards my doctorate committee, namely to Gerhard Weikum, Birgit Pfitzmann, Joachim Biskup, and Christoph Weidenbach.

Some results in this thesis have been published as [PfSW 98, PfSW 00, PfSW1 00, PfSW2 00, Schu 00, ScWW1 99]. This work was partially supported by the *SEMPER* project `http://www.semper.org`; however, it represents the view of the author only.

Chapter 2

Related Work

In this chapter, we give an overview of the existing literature on fair exchange. An overview of existing notions of security for reactive systems will be given in Chapter 7 after introducing the required terminology.

2.1 Classification of Fair Exchange

In this overview, we classify fair exchange protocols by the items they allow to exchange. The existing work mainly deals with fair exchange of specific types of business items. Only few proposals come close to our generic notion of fair exchange. Items for which specific exchange protocols have been described in the existing literature are

Secret Data The transfer protocol reveals data which is not known to the recipient beforehand.

Signatures A signature is transferred by executing a (possibly interactive) signature protocol.

Payments A payment is transferred by executing a payment protocol.

In principle, any set of items can be exchanged for any other set of items. For the special case of exchanges of the single items mentioned above, this would lead to at least nine instances of fair exchange, if there were only one implementation of each item. In the existing literature, only few particular instances of fair exchange received attention. Thus, the remainder of this chapter concentrates on these instances:

Contract Signing (Section 2.2) is a protocol where two players compute a non-repudiable agreement on a contract text. It can be provided by fair exchange of signatures on a given contract.

Certified Mail (Section 2.3) is the sending of a message for a receipt. It can be provided using a fair exchange of secret data for a signature on the description of the sent message.

Fair Purchase (Section 2.5.1) is an exchange of secret data for a payment. A special case is “payment for receipt”, i.e., an exchange of a payment for a signature.

Fair Exchange of Secrets (Section 2.5.2) is the fair exchange of one secret data item for another.

Generic Exchange (Section 2.4) Generic exchanges exchange any two items satisfying certain conditions.

For each kind of fair exchange, we group the protocols by the extent to which a third party is involved:

Fair Exchange with In-line Third Party: These protocols include a third party which participates in all exchanges and guarantees fairness. The advantage of this approach is that the protocols guarantee a fair outcome in a limited time. The disadvantage of this approach is that the third party participates in all exchanges, which leads to performance and trust bottlenecks: Even if the participants are correct, they cause load at the third party and the third party may misbehave or infringe the privacy of the parties participating in the exchange.

Optimistic Fair Exchange: These protocols include a third party as well. However, this third party is not involved if both participants are correct and agree. Thus, the third party is no longer a performance and trust bottleneck while still being able to guarantee fairness in case of disagreement, misbehavior, or failure.

Gradual Fair Exchange without Third Party: These (cryptographic) protocols aim at a probabilistic guarantee of fairness and do not require any third party.

Naturally, the fair exchange protocols without a third party seem to be the best choice. However, these cryptographic solutions are based on a gradual exchange where tiny parts of the item to be exchanged are exchanged in many rounds. Compared to protocols with a third party the known probabilistic protocols have the following disadvantages:

- The protocols only achieve probabilistic fairness [EvYa 80]: Instead of guaranteeing fairness, the protocols only guarantee that the advantage of one party over the other can be made small.
- The round-complexity is linear in the degree of this probability of an unfair outcome, e.g., for achieving a fairness guarantee of 99%, about 100 rounds are required.
- Some protocols do not guarantee a definitive outcome in a limited time: The answer of a protocol that a house has been sold with probability 95% does not enable the seller to decide whether it is allowed to sell the house again or not.

Thus, we feel that using a third party cannot be avoided for most applications. Compared to the optimistic protocols, the *in-line protocols* have the disadvantages that

- the protocol provides a weaker guarantee of availability since the third party must be available during the protocol,
- the parties are required to trust the third party since their security depends on the well-behavior of the third party, and
- this also holds for the privacy of the players: The third party may be able to keep track of who exchanges what with whom.

Therefore, optimistic protocols are the best choice for most applications: In the error-free case, they achieve a better performance without requiring a third party. Only in case of failure (which should be seldom), a third party is needed.

Remark 2.1. Protocols for general multi-party computation [Yao 86, GMW 87] can usually not be used for fair exchange, since they assume an honest majority. ○

2.2 Contract Signing

Contract signing is a generalization of the fair exchange of two signatures under a contract. The term "contract signing" was first introduced in [Blum 81]. An ideal contract signing protocol should offer fairness without a third party or assumptions about the computational capabilities of the signatories. It should either be deterministic or probabilistic with a negligible error-probability. Unfortunately, such protocols do not exist [EvYa 80][1]. Therefore one has to make a tradeoff between deterministic protocols, which are efficient but require trust into a third party, and probabilistic protocols with error probabilities, which are linear in the number of rounds.

Remark 2.2. Fair exchange of signatures[2] and *fair contract signing* are different problems since contract signing does not require a contract to be a text and two signatures. Obviously, contract signing can always be implemented based on fair exchange of signatures, but not all contract signing schemes exchange signatures. They only guarantee non-repudiation of the agreement on a contract.

Remark 2.3. Contract signing protocols achieve more than distributed commitment: While commitment protocols [SiKS 97] usually[3] assume that the players are correct and that an agreement need not be proven, contract signing aims at a non-repudiable agreement while assuming a Byzantine failure model, i.e., even if the signatories are malicious, contract signing guarantees that consensus can be proved whereas commit-protocols do not.

Remark 2.4. Agreement protocols [PeSL 80], like contract signing, assume a Byzantine failure model. Still, they usually do not aim a non-repudiation of

[1]In [EvYa 80], it was shown that no deterministic contract signing scheme (called "public-key agreement system") without a third party exists if the verifier is state-less and only the two signatories participate in the contract signing protocol. In Section 3.3.2, we extend this result to include state-keeping verifiers.

[2]Each player A receives a digital signature $\mathrm{sign}_\mathsf{B}(C)$ on a contract text C if and only if the other signatory B receives $\mathrm{sign}_\mathsf{A}(C)$, too.

[3]Some extensions, such as [RoPa 93], also consider stronger failures.

the agreement. However, recent proposals provide non-repudiation and can be used as a building block for multi-party contract signing [DoSc 98, DoSc1 98]. ◦

2.2.1 In-line Protocols

Early work on contract signing with in-line third party was done in [Rab1 83]: In a sketched protocol, both signatories send their signatures to the third party who verifies and forwards them.

In order to reduce the involvement of the third party, [Rab1 83] proposes to use so-called beacons emitting random integers for a probabilistic protocol: After signing an initial agreement, both signatories choose random numbers and sign the sum of them together with the correct round number. Then, the contract is declared valid if the sum of the two chosen integers is the signed random number broadcasted by the beacon. Even though the involvement of the third party is reduced, this protocol has the disadvantage that the cheating probability is linear in the run-time of the protocol: If the random number is chosen in a space of n elements, it will take about n rounds until the beacon actually signs the chosen integer. For one round, however, a party may not send the signed sum while receiving it from the peer. In this case, the beacon will still broadcast the correct sum with probability $1/n$.

2.2.2 Optimistic Protocols

Recent research concentrated on optimistic contract signing schemes that avoid such uncertain situations, and guarantee a *definite* decision within limited time: The first *optimistic* scheme has been described in [Even 83]. [AsSW 97] (recently implemented in [Oser 99] and formally analyzed in [ShMi 99]) describes a synchronous contract signing protocol with four messages. This was improved in [AsSW 98] to a four-message protocol for asynchronous networks which, in addition, implements fair exchange of signatures using verifiable encryption. This enable the third party to replace missing signatures [AsSW1 98], without changing the format of the resulting signature. I.e., in order to enable the third party to generate a “real” signature (e.g., $\mathrm{sign}_{\mathsf{A}}(C)$) under the contract instead of just signing on behalf of an incorrect signatory (i.e., $\mathrm{sign}_{\mathsf{T}}(\mathrm{sign}_{\mathsf{A}}($“$\mathsf{T}$ may sign C on my behalf”$))$. Independently, [BaDM 98] proposed to use verifiable encryption for fair exchange of signatures as well.

Another extension called *abuse-free optimistic contract signing* has been proposed in [GaJM 99]. In all optimistic contract signing schemes mentioned above, a signatory may at some point during the protocol be able to choose whether to validate or invalidate the contract with the help of the third party. At this point, it already obtained a signature from the peer that the peer would, in principle, be willing to sign a particular contract. The authors claim that “his willingness to sign [can be used] to get leverage for another contract” and fix this “problem” using designated verifier signatures.

Optimistic *multi-party* contract signing protocols have first been described in [AsSW2 96] and have recently been extended in [ABSW 98, ABSW 99]. An asynchronous protocol for optimistic multi-party contract signing has been proposed in [BaWa 98].

2.2.3 Gradual Protocols

For contract signing, the gradual exchange protocols without third party can be further subdivided into two categories:

Gradual Exchange of Secrets: Both parties convert the secret signature to be exchanged into a secret in a format to which existing fair exchange of secret protocols can be applied. Then, the secret is exchanged gradually in many rounds.

Gradual Increase of Privilege: These protocols gradually increase the probability with which a contract is valid.

Gradual Exchange of Secrets: The first protocols based on gradual exchange of secrets have been proposed by Blum [Blum 81, Blum 83, Blu2 83] and were based on a specific number theoretic assumption which has been proved wrong in [HaSh 85]. The protocols were then adapted in [Gol4 83, Gold 83] to the concept of puzzles [Merk 78] which can be based on any public-key crypto system. Since the difference of computation needed by honest and dishonest signatories was small[4], the monetary value of the contract was one input to the protocols. This enabled the protocol to choose the security parameters so that forging the contract was estimated more expensive than the value of the contract. This was fixed in [Even 83, EvGL 83, EvGL 85] by using oblivious transfer as a primitive: The signatures are divided into n shares which are sent using oblivious transfer. Thus, after sending the shares using oblivious transfer, each player knows half of the shares but the other player does not know which ones. Then, both players reveal all shares bit-by-bit. In this stage, the shares already received are used to detect cheating. In [Damg 94, Damg 95], Damgård describes the first protocols for gradual fair exchange of RSA, Rabin, or ElGamal signatures which are provably secure, i.e., one can prove that after an interruption, the computational work left to both parties differs at most by one bit of a secret and breaking the scheme is as hard as factoring.

The disadvantage of using gradual exchange of signatures for contract signing is that the fairness depends on equal computational power of all signatories: One assumes that both parties would be able to compute the remainder of the secret to the same extent. Furthermore, if the protocol aborts while a large part of the secret has not been exchanged, both parties cannot be sure whether the contract is valid or not since, e.g., one cannot know whether the other signatory spends the remaining computational effort to make the contract valid. Thus, if a contract signing protocol for selling a house has been aborted, one cannot decide whether the house has been sold or not.
Gradual Increase of Privilege: An alternative approach is based on gradual increase of privilege. The protocol proposed in [BGMR 85, BGMR 90] exchanges signatures under a contract by means of gradually increasing the probability that a judge rules the contract as being valid. To our knowledge, this is the only protocol based on this principle. The protocol works as follows: In n messages sent by each signatory, the probability that a contract is declared valid is gradually increased from 0 to 1. If the protocol stops prematurely, each signatory

[4]An adversary can solve puzzles in the square of the amount required from the intended recipient, whereas an exponential amount is desirable.

can invoke a third party called “judge.” This third party waits until the signing protocol has terminated (i.e., we are in a synchronous model). After this timeout, the third party picks a real random number ρ in the interval $[0, 1]$, or retrieves it in case the third party was invoked for this contract before. If the probability given by the last message received by the invoking party is at least ρ, the contract is considered valid and an affidavit is issued and sent to both signatories. Otherwise the contract is considered invalid. By construction, if the protocol is prematurely stopped, one party might be “privileged”, i.e., has a slight advantage when invoking the judge: If the third party chooses a ρ that lies between the probabilities of the two signatories, only one of them can finalize the contract. Thus, if a correct player A invokes the third party and gets the answer that the contract is invalid, she cannot be sure that the same would happen if B invokes the third party, i.e., that the contract is indeed not signed. In the worst case, B might have a valid contract (i.e., probability 1) and hence knows that if A complains, it will succeed only with the probability contained in the last message sent by B. The probability that such an uncertain situation arises is linearly small in the number of messages exchanged [BGMR 90].

A problem with this approach is that one still has the problem of the half-sold house: I.e., if the protocol is interrupted in between, a seller cannot be sure whether his house was sold or not without contacting the third party.

2.3 Certified Mail

Certified mail is the fair exchange of secret data for a receipt [Blum 81, Rabi 81]. It can be provided by a fair exchange of secret data for a signature on the description of the data.

Certified mail is the only instance of fair exchange for which a standardized framework exists [ISO13888-1 97]: The players in a certified mail system are at least one sender S and one receiver R. Depending on the protocols used and the service provided, the protocol may involve one or more third parties T. If reliable time-stamping is desired, additional time-stamping authorities TS may be involved, too. For evaluating the evidence produced, a verifier V can be invoked after completion of the protocol. Sending a certified mail includes several actions [ISO13888-1 97]. Each of these actions may be disputable, i.e., may later be disputed at a verifier, such as a court (see Figure 2.1): A sender composes a message (non-repudiation of origin) and sends it to the first third party (non-repudiation of submission). The first third party may send it to additional third parties (non-repudiation of transport) and finally to the recipient (non-repudiation of delivery, which is a special case of non-repudiation of transport). The recipient receives the message (non-repudiation of receipt).

For each of these actions, the party involved may produce evidence, i.e., signed tokens, in order to enable other parties to later dispute the action at the verifier. In addition to the parties performing the action, additional witnesses may sign the token, too. A non-repudiation of receipt token, for example, may be signed by the recipient and one of the third parties. If we assume that evidence about the actions of a player should not be produced without its consent, one should require that each token at least contains the signature of the party performing the action. The protocols may provide time-stamping of the evidence produced, too.

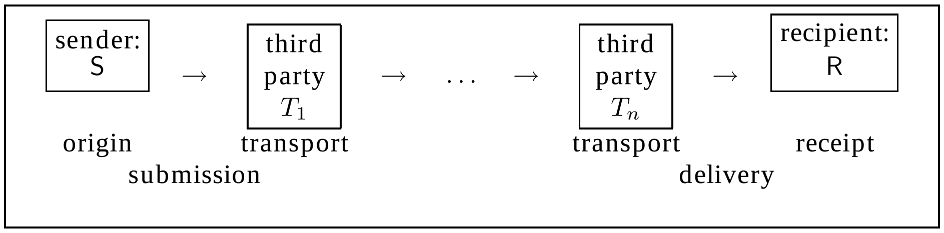


Figure 2.1: Framework for Certified Mail: Players and their actions.

The most common evidence which is produced by all certified mail protocols are “non-repudiation of receipt” tokens. The obvious problem when producing this kind of evidence is fairness, i.e., the receipt should only be issued if the recipient was able to obtain the message and vice versa. The obvious problem achieving this is that if the sender reveals the message first, the recipient may refuse to sign a receipt. If the receipt is sent first, the sender may refuse to send the message. A similar fairness requirement should hold for protocols providing non-repudiation of origin: If the recipient receives a message (and acknowledges the receipt), the recipient should be able to prove the origin of the message.

In addition to the kinds of non-repudiation provided, we can also distinguish by the structure of the non-repudiation tokens produced. Sometimes, the message itself or a hash values of it is signed[5]. A weaker form are signatures on keys or encrypted messages which require that the encrypted message cannot be decrypted into different messages using different keys. Furthermore, the produced evidence may include a signature on a time-stamp by a time-stamping authority TS.

Additional criteria for the classification of certified mail protocols given in Table 2.1 are:

Confidentiality: The protocol keeps the contents of the message confidential against the third party or against eavesdroppers observing the network.

Communication Model: All protocols require that the network is reliable, i.e., that messages can eventually be delivered or obtained from a third party. In addition to this property, some protocols require a synchronous network where the parties can decide on time-outs of messages.

In the following subsections, we will focus on fair mechanisms using asymmetric techniques, i.e., we will omit protocols which do not guarantee fairness (such as [ISO13888-3 97] or “CEM” in [ZhGo1 96]) as well as symmetric techniques (such as security envelopes [ISO13888-2 98]).

We now describe selected protocols for certified mail in detail. A comparison of all protocols offering deterministic fairness is given in Table 2.1. (A survey on the listed proposals of Zhou and Gollman is given in [ZhGo3 96].)

2.3.1 In-line Protocols

Early work on fair exchange with in-line third party was done in [Rabi 83, Rab1 83]. More recently, a multitude of similar protocols with in-line third

[5]A hash function $\mathcal{H}()$ is a collision-free one-way function, i.e., it is difficult to find a pre-image x given $\mathcal{H}(x)$ and two $x_1 \neq x_2$ such that $\mathcal{H}(x_1) = \mathcal{H}(x_2)$ [Damg 88].

	Signatures on NR-tokens					**Properties**	
Protocols	NRO	NRS	NRT	NRD	NRR	Conf.	Rem.
[AsSW 97]	S				R	T	O, S
[AsSW 98]	S				R	T	O
[BaTy 94]	S,T	T,S			r,S,t	-	
[BaDM 98]					R	T	O, S
[CoSa 96][1]	S,TS,T				R,TS,S,T	-	
CMP1 [DGLW 96]	S,T				R,T	-	
CMP2 [DGLW 96]	S,T				R,T	E	
[Han 96]	S,T				R,T	-	B
[ISO13888-3 97][a]	S,t		T	T		-	
[Mica 97]					R	E,T	O, S
[ZhGo 96]	S,t				r,t	T	
CEM-ip [ZhGo1 96][a]		T	T	T	R,T	-	
[ZhGo2 96]	S	S,T		S,R,T	S,R	-	ts
[ZhGo 97]	s				r	T	O, S
[ZhSh 96]	s,t				r	T	S,ts

[a]This paper aims at a non-repudiation infrastructure.

Legend:

NR	Non-repudiation of origin (O), submission (S), transport (T), delivery (D), and receipt (R).
NRO to NRR	Signature on tokens by sender (S, s), recipient (R, r), third party (T, t), or time-stamping authority (TS). Upper case means signature on message. Lower case signals signatures on ciphertexts or keys.
Conf.	Confidentiality against third parties (T). Some protocols (E) also have integrated link-to-link encryption for protection against network eavesdropping. Naturally, (E) could be added to any protocol afterwards whereas (T) would require protocol modifications.
Remarks	"O" stands for optimistic and "B" stands for black-board (see Page 13), "S" stands for synchronous networks, and "ts" stands for time-stamps from the third party.

Table 2.1: Overview of properties of protocols for certified mail.

parties have been proposed. They mainly differ in the evidence generated, i.e., which signatures are required on which non-repudiation token. Instead of describing all protocols, their main service properties are summarized in Table 2.1.

One example of a protocol using an in-line third party is the protocol proposed in [ZhGo 96, Zhou 96][6]. The basic idea is that the parties first exchange signatures under the encrypted message. Then, the third party signs and distributes the key. The signature on the encrypted message together with the signatures on the key then forms the non-repudiation of origin and receipt tokens. The protocol is sketched in Figure 2.2[7].

Another example is the protocol CMP1 proposed by [DGLW 96] as sketched in Figure 2.3. The basic idea of this protocol is that the sender encrypts the signed message with a symmtric cipher, encrypts the used key of the cipher

[6]This protocol was formally analyzed in [Schn3 98] using CSP.

[7]In the sequel, we do not consider any details of the sketched protocols, such as participant names or other identifiers in the messages. The goal of the subsequent descriptions is mainly to illustrate the message flows, i.e., for a real description, we have to refer to the cited literature.

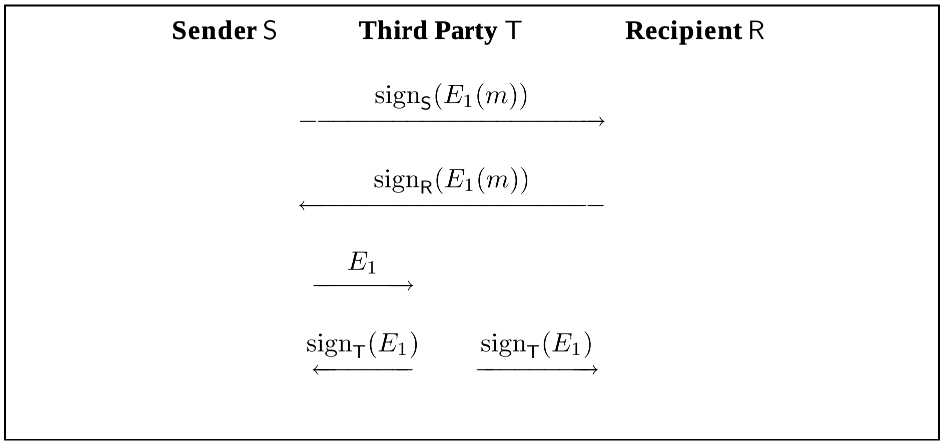

Sender S
Third Party T
Recipient R
$\text{sign}_{\mathsf{S}}(E_1(m))$
$\text{sign}_{\mathsf{R}}(E_1(m))$
E_1
$\text{sign}_{\mathsf{T}}(E_1)$
$\text{sign}_{\mathsf{T}}(E_1)$

Figure 2.2: Sketch of the Protocol Proposed in [ZhGo 96] (E_1 denotes symmetric encryption).

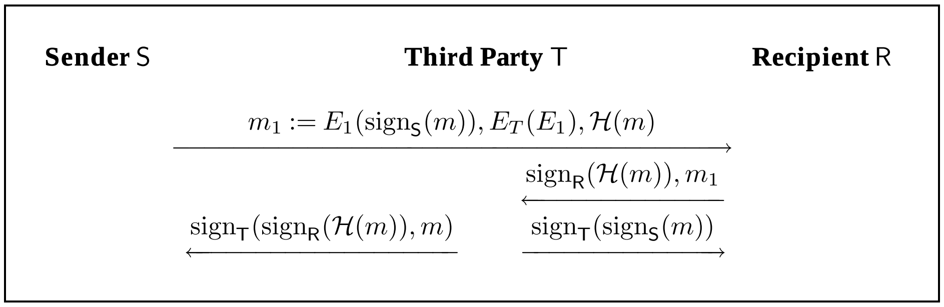

Sender S
Third Party T
Recipient R
$m_1 := E_1(\text{sign}_{\mathsf{S}}(m)), E_T(E_1), \mathcal{H}(m)$
$\text{sign}_{\mathsf{R}}(\mathcal{H}(m)), m_1$
$\text{sign}_{\mathsf{T}}(\text{sign}_{\mathsf{R}}(\mathcal{H}(m)), m)$
$\text{sign}_{\mathsf{T}}(\text{sign}_{\mathsf{S}}(m))$

Figure 2.3: Sketch of Protocol "CMP1" proposed in [DGLW 96].

with the public key of the third party, and sends these parameters together with the hash-value of the message to the recipient. The recipient then signs this hash value and forwards all information to the third party. The third party decrypts the message and sends a signed message containing the message signed by the sender to the recipient and sends a signed message containing the hash-value signed by the recipient and the actual message sent to the sender.

One protocol which pursues a different approach has been proposed in [Han 96]. The paper claims that no third party is needed. However, it uses strong blackboard-like communication primitives. It requires a blackboard that guarantees that both parties are able to publish a message under an address which cannot be erased afterwards and can only be retrieved by a judge or by knowing the address. Note that this model is worse than the third parties used in other protocols since the black-board needs persistent memory whereas most other third parties are memoryless. A similar proposal has been made in [Tang 96].

The activities around certified mail are reflected in recent standardization activities [Herd2 95]. ISO standards ISO13888 standardize the service and mechanisms for non-repudiation [ISO13888-1 97, ISO13888-2 98, ISO13888-3 97]: ISO13888-1 describes a framework for non-repudiation sketched in Figure 2.1 and describe above, ISO13888-2 describes symmetric techniques (so called security envelopes), and ISO13888-3 describes asymmet-

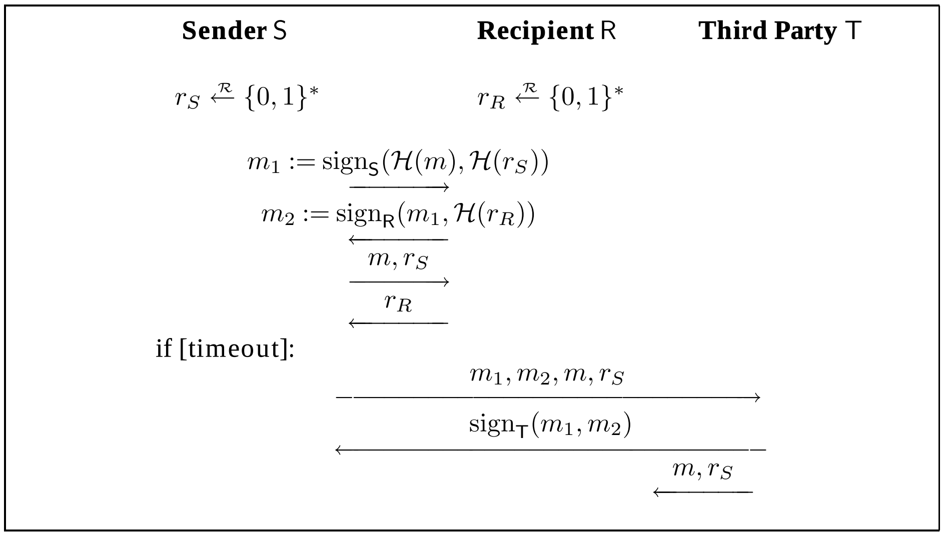

Sender S
Recipient R
Third Party T
$r_S \overset{\mathcal{R}}{\leftarrow} \{0,1\}^*$
$r_R \overset{\mathcal{R}}{\leftarrow} \{0,1\}^*$
$m_1 := \mathrm{sign}_{\mathsf{S}}(\mathcal{H}(m), \mathcal{H}(r_S))$
$m_2 := \mathrm{sign}_{\mathsf{R}}(m_1, \mathcal{H}(r_R))$
m, r_S
r_R
if [timeout]:
m_1, m_2, m, r_S
$\mathrm{sign}_{\mathsf{T}}(m_1, m_2)$
m, r_S

Figure 2.4: Sketch of certified mail protocol from [AsSW 97].

ric techniques using digital signatures. An overview of an early stage of standardization was described in [Herd2 95]. The initial drafts of this standard did not ensure that the recipient really issues a non-repudiation of receipt token (the so-called "selective receipt problem"). A proposal how to fix this problem has been made in [ZhGo2 96].

2.3.2 Optimistic Protocols

The basic goal of optimistic certified mail protocols is that the third party need not be involved to guarantee fairness if both parties follow the protocol [AsSW3 96, AsSW 97, Mica 97]. Only if one party misbehaves, the third party is invoked to restore fairness.

Optimistic certified mail has first been described in [AsSW3 96] generalizing the fair payment for receipt protocols from [BüPf 89]. Independently, Micali developed a similar protocol for certified electronic mail [Mica 97] (patented in [Mica1 97, Mica2 97]).

The protocol for optimistic certified mail proposed in [AsSW 97] (see Figure 2.4) provides non-repudiation of receipt and origin: The sender signs a hash-value $\mathcal{H}(m)$ of the message that it will send, the recipient signs an acknowledgment that it is willing to receive the fixed message. Then, the sender sends the message and the recipient sends the receipt. For saving one signature, this receipt consists of a pre-image r_R together with the signed message m_2 fixing the one-way image $\mathcal{H}(r_R)$. If the recipient does not send a receipt, the sender invokes the third party with the acknowledgment in which the recipient states that it is willing to accept the message in exchange for a receipt. The third party then signs a replacement receipt and forwards the message to the recipient. Another presentation as well as an efficiency improvement to the protocol proposed in [AsSW 97] has been described in [ZhGo 97].

The independently developed protocol for optimistic certified mail proposed in [Mica 97] (see Figure 2.5) provides non-repudiation of receipt as well

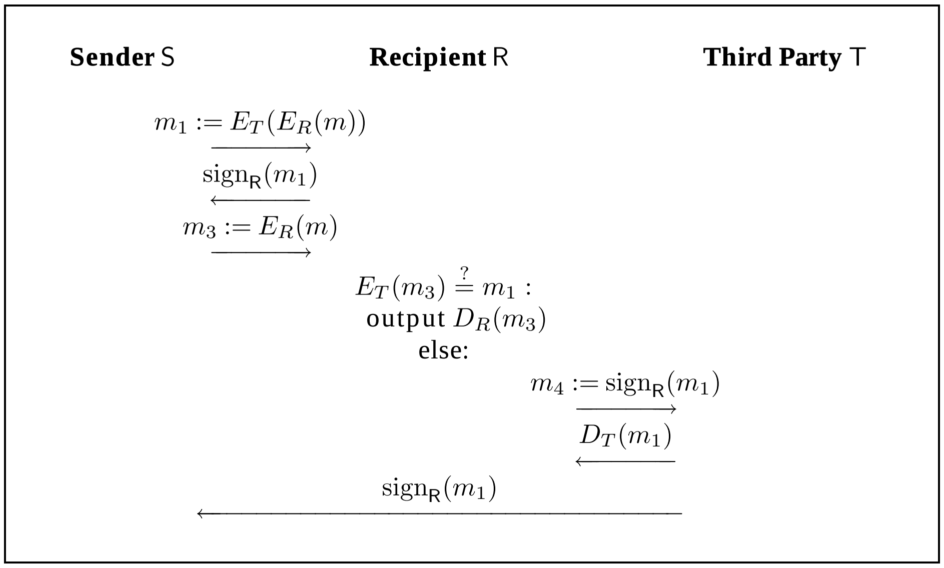

Sender S
Recipient R
Third Party T
$m_1 := E_T(E_R(m))$
$\mathrm{sign}_{\mathsf{R}}(m_1)$
$m_3 := E_R(m)$
$E_T(m_3) \overset{?}{=} m_1$:
output $D_R(m_3)$
else:
$m_4 := \mathrm{sign}_{\mathsf{R}}(m_1)$
$D_T(m_1)$
$\mathrm{sign}_{\mathsf{R}}(m_1)$

Figure 2.5: Certified mail protocol from [Mica 97].

as confidentiality against eavesdroppers and third parties: First, the sender encrypts the message with the public key of the recipient. The sender then encrypts the ciphertext with the public key of the third party. The recipient signs the encrypted message and the sender reveals the ciphertext. The recipient finally checks whether the ciphertext encrypted with the public key of the post-office is identical to the ciphertext signed[8]. If this is not the case or if the sender did not send anything at all, the recipient asks the third party to decrypt the ciphertext signed in the receipt. Note that since the non-repudiation of receipt token is not signed by the sender, the recipient may choose the message to be contained in the receipt, i.e., the token does not prove that the message was sent but only that the recipient claims that it was received.

More recent proposals for optimistic certified mail are included in [AsSW 98, AsSW1 98], which enable fair exchange on asynchronous networks and show how to enable the third party to produce replacement signatures using verifiable encryption[9].

2.3.3 Gradual Protocols

These protocols only involve the sender and the recipient during message transmission and a verifier for deciding disputes afterwards. All proposed protocols for this scenario need to be probabilistic [EvYa 80], i.e., they do not guarantee fairness but rather guarantee that the advantage of one party over the other can be made small. The basic idea of this proof is that both parties reveal a secret message or secret signature, respectively. For each party therefore exists a final message which allows the recipient to compute the secret. No

[8]This assumes that the encryption is deterministic, i.e., that any message is always encrypted into the same ciphertext.

[9]An independent proposal for using verifiable encryption for fair exchange on synchronous networks was published in [BaDM 98].

matter who sends this final message first has a disadvantage since the other party may terminate the protocol without sending his final message.

Early cryptographic schemes are based on a gradual exchange of the message for the receipt. These protocols only involve the sender and the recipient during message transmission and a verifier for deciding disputes afterwards. All schemes of this kind cannot achieve deterministic fairness [EvYa 80]: They do not give a definitive answer but only guarantee that the advantage of one party over the other can be made arbitrarily small (but still linear in the number of messages), e.g., computing the signature on the receipt requires a similar effort for the sender than computing the message requires from the recipient.

The scheme proposed in [Gol1 84] uses oblivious transfer for sending certified mail. In principle, the sender first creates n shares of the message to be sent using a so called threshold scheme[10]. The recipient signs a message that says that the receiver acknowledges having received a message, if the sender can show the pre-image of a given image produced by applying an one-way function to a randomly chosen secret. This pre-image is divided into n shares, too. Then, both parties send all their secrets using oblivious transfer[11]. Since the sender does not know whether the message has been received or not, each party only knows that the peer has about half of the secrets but does not know which of them. These shares received in oblivious transfer will later be used to detect cheating in the last phase: Both parties send one bit of each share to the peer. Since the recipient knows about half the secrets, it can verify that at least half of the bits received are identical to one of the known shares whereas the other bits can be used to construct the remaining shares. Then, they send the next bit of all shares and repeat this procedure until all bits have been sent. If any party interrupts before all bits have been sent, both parties have almost the same amount of information. For fairness, this protocol assumes that both parties have similar computational power, i.e., that no party is better at computing the shares to be exchanged bit-by-bit. Other cryptographic schemes like [Blum 81, BlVV 84, EvGL 85, Gold 82, VaVa 83] use similar techniques, i.e., they convert the certified mail problem into a problem of fair exchange of secrets by using a threshold–scheme and then perform a gradual exchange of these n secrets.

2.4 Generic Fair Exchange

The model of generic fair exchange has been proposed in [AsSW 97] together with an optimistic protocol for several instances of it. The basic idea of generic fair exchange is to enable the fair exchange of different items fulfilling certain conditions using one protocol.

Existing generic fair exchange protocols usually assume that the items can be exchanged by single messages, i.e., unlike the protocols described in Part III, they cannot be used to exchange interactive protocols.

The first protocol for multiple instances of fair exchange was published in [AsSW 97]. The protocol requires four rounds and time four for synchronous

[10] A (n, k)-threshold scheme [Sham 79] divides the secret message into n shares where, e.g., $k \leq n$ shares are needed for reconstructing the secret.

[11] Oblivious transfer [Blum 81] sends a message so that the probability of receiving it is, e.g., 0.5

optimistic fair exchange. They assume that the item can be sent in one message, which can be re-sent via the third party without changing the outcome (i.e., they assume that one item provides verifiability; see Section 12.2). They guarantee fairness if at least one of the items can be replaced or revoked by the third party (i.e., the other item is required to provide generatability or revocability; see Section 12.2). The protocol provides non-repudiation for both players, i.e., the exchange can be proven afterwards. Without non-repudiation of receipt, the protocol needs three messages in three rounds.

In [AsSW 97], generatability of signatures was provided by so-called affidavits, i.e., the third party signs on behalf of an absent or incorrect signer. Therefore, [AsSW1 98] and [BaDM 98] independently proposed to use verifiable encryption as an implementation of generatability of signatures. This has the advantage that the resulting replacement signature cannot be distinguished from ordinary signatures, i.e., fairly exchanged signatures look like ordinary signatures.

In [AsSW 98] (see also [Asok 98]), the protocol of [AsSW 97] was extended to asynchronous networks. The protocol requires four messages in four rounds without producing evidence for non-repudiation.

Generic optimistic multi-party fair exchange has first been described in [AsSW2 96] recently extended in [ABSW 98, ABSW 99]. Protocols for multi-party fair exchange using in-line third parties have been proposed in [FrTs 98, KeGa 96].

2.5 Other Instances of Fair Exchange

This section describes instances of fair exchange, which are not considered in so much detail in this thesis.

2.5.1 Fair Purchase

Fair purchase is the fair exchange of a payment for secret data to be purchased. It involves a seller and a buyer. The protocols can be divided into three categories:

Generic: Protocols which do not make any assumption about the payment system.

Part of Payment System: Protocols using the payment switch of the bank as a third party guaranteeing the fairness of the exchange. This enables more efficient protocols.

Gradual: Tiny payments are exchanged for tiny parts of the item. This assumes that the tiny parts are useful in themselves[12].

An inherent problem of fair purchase is that for many items, such as computer software, the protocol cannot guarantee that the items delivered really match the description of the buyer: Whether a program is a word-processor satisfying a set of requirements cannot be decided automatically. In those cases, the protocol can only “bind” the description to the items, i.e., produce evidence that a

[12]In our opinion, this is currently the most practical instance of gradual fair exchange.

seller promised that the delivered items fulfill the given requirements. The dispute whether the delivered items fulfill the promised requirements then have to be resolved by a human arbiter outside the system. Naturally, the more precise this description is, the more verifications can be done inside the system and thus the stronger the guarantees which can be given automatically. Only for machine-verifiable items, such as credentials, one can guarantee fairness automatically.

Generic Fair Purchase and Payment for Receipt Protocols

Since protocols for fair purchase can also be used to “buy” simple receipts[13], we now describe both concepts together. We omit schemes like [SuTy 96, Zwiß 97] which do not guarantee fairness of the outcome.

The first generic fair purchase protocol has been proposed in [BüPf 90, PWP 90]. In this protocol, both parties first agree on the description of the data and the amount to be paid. Then, the money is sent to the third party. After the third party acknowledges the receipt of the money to the seller, the seller sends the data to the third party. The third party forwards data and money, if the data fulfills the agreement.

This protocol requires an on-line third party participating in the protocol. This problem has been solved by the generic protocol proposed in [AsSW 97]: After the agreement, the buyer sends the money and the seller sends the data. Only if the goods are not as expected, the buyer invokes the third party which verifies the data and revokes the payment, if the data do not match the description. Note that this protocol only guarantees fairness if the third party is able to revoke the payment or if it knows the data to be delivered beforehand. If this is not the case, the protocol still produces evidence that the seller misbehaved which can be used outside the system.

Producing such evidence which can be used outside the system is the aim of the generic protocol which has been proposed in [HaTs 96]: The seller sends a signed offer. The buyer pays, the seller delivers. The buyer sends a receipt. If the seller does not send the data, the buyer raises a dispute at a third party which then observes a replay of the protocol. If the seller again does not send the data, the resulting claim is settled outside the system.

In the protocols described up to now, a third party participates in the fair purchase. Another class of fair exchange protocols use third parties which provide disputable sub-services such as disputable storage or disputable communication, i.e., a purchase then includes a buyer, a seller, the bank, and the third party. The alleged advantage is that this disputable communication could be used for other services, too, or that if the hardware provides it, no additional third parties are needed. If the services are installed for fair purchase only, it makes no difference to third parties participating in the protocols.

[Tang 96, Tang1 96] describes a service called verifiable transaction atomicity. One goal of this service is to enable third parties to later verify which items have been exchanged. This is achieved by a third party called “transaction log” (i.e., in our terminology, the third party is in-line). Another goal is that a buyer who paid receives the data. This is achieved by means of an unspecified

[13]This is the case if the receipt is known beforehand, i.e., if it only contains parameters such as the value and date of the payment. It is not the case if the receipt should fix payment internals, such as coin numbers.

“disputation-resolve” method. Another proposal for using a transaction log for non-repudiation has been made by [CHTY1 96] (see also [Camp 96]).

Payment Systems Enabling Fair Purchase

Most recent payment system proposals (e.g., [BBCM1 94, BGHH 95, SET 97]) provide a means to link a text to the payment made. This field called “merchandise description” in [BGHH 95] appears on the statement of account and is part of the produced receipt. It can be used to dispute the purchase later. However, these payment protocols do not include a reference to the data being delivered: The buyer is enabled to prove that a payment has been made to a particular seller for data matching a given description. The seller is then required to prove that it delivered the data by other means.

Some payment systems also provide a real proof of the transaction, i.e., a receipt containing the data, the description, and the amount paid. In NetBill [CoTS 95], for example, the buyer includes a reference to the encrypted data into the merchandise description. The NetBill server then sends a receipt to the buyer which contains the key to decrypt the data after the payment has been made. Thus, this receipt sent to the buyer contains references to the description of the data, the encrypted data, and the key used to encrypt them. This enables the buyer to dispute the complete purchase afterwards.

A limited form of fair purchase is the exchange of payments for receipts. All research in this area is closely related to research in payment systems. Payment-system-independent optimistic receipt mechanisms have been proposed in [AsSW3 96, BüPf 89].

A number of payment systems provide means to settle disputes about whether a payment has been made or not involving the bank as a witness [BüPf 89, BGHH 95, BBCM1 94]. For a survey of recent payment systems see [AJSW 97].

[ShBD 98] uses an in-line third party for fair purchase. The protocols are based on an integrated on-line payment system where money is kept in accounts at the third party. For efficiency, the authors describe how the third party can be distributed onto multiple machines.

Gradual Fair Purchase

Gradual exchanges of payments for data are provided by so-called micropayment systems first described in [Pede 95] (more recent publications are [AnMS 96, HaSW 96, Mana1 95, Pede 97, RiSh 97]). The exchange protocols then exchange each low-value part of the data against one tick of the micropayment scheme. Examples are to bill a few cents for each subsequent couple of video-seconds or each subsequent web-page. If any of the parties does not send its fraction, the protocol ends. At the end, any party may loose at most one part of the data or one payment tick.

A slightly different approach was described in [Jako1 95]: The protocol proposes to adapt the mechanism of two-spendable coins for fair exchange: The first spending is done before the data is received. Then, the seller sends its data. Finally, the seller hopes that it receives the second payment which enables it to deposit the coin. The rationale behind this hope is that after the first payment (i.e., after spending half of the coin), the coin is defined to be unspendable, i.e., the buyer would not gain by not paying the second half.

2.5.2 Fair Exchange of Secrets

The service of fair exchange of secrets exchanges two secrets. Each secret is described by a publicly known description which can later be used to verify the received secret. The protocols can be divided into categories by means of the kinds of secrets and descriptions which are supported by them.

Data: These protocols exchange secret data described by hash values or images of one-way functions. We assume that this image is known beforehand. For purchases, for example, it could be part of the offer.

Discrete Logarithms: The secret is a discrete logarithm x of a publicly known value g^x in a known group.

Factorizations: The secret is the prime factorization (p, q) of a publicly known number $n = pq$.

Bit-Strings: A sequence of single bits is revealed. The description is a random subset of the bits that are known by the recipient (e.g., “00100010” can be described as “0xx00x1x” where “x” stands for unknown).

Exchanging discrete logarithms or factorizations can be adapted to a fair exchange of signatures, i.e., these protocols can be used for contract signing, too. Naturally, the generic fair exchange protocol proposed in [AsSW 97] can be applied to any fair exchange of secrets. A protocol for the exchange of data described by a hash value has been proposed in [FrRe 97]. The authors describe a non-optimistic protocol using a third party to guarantee fairness. Besides offering the fair exchange service, the authors aim at reducing the trust into the third party:

- The third party is not able to learn the data to be exchanged.
- Any two honest players can detect dishonesty of the third player.

The protocol of [FrRe 97] uses verifiable secret sharing [CGMA 85] to achieve these requirements: In principle, the secret data is shared with a 2-out-of-2 scheme into two parts. Each party sends one share to the third party and the other part to the other party participating in the exchange. The verifiable secret sharing scheme guarantees that both recipients of the shares can verify that the shares contain data corresponding to the given one-way image[14]. The third party then exchanges and forwards the received shares. Finally both participants reconstruct the received secret using the shares received from the third party and from the other participant.

In [Syve2 98], a proposal how to apply fair exchange of secrets to fair exchange of data is made. The participating parties first exchange encrypted items. Then, in the subsequent rounds, they reveal information which reduces

[14]The paper contains proposals for a number of different on-way functions based on different cryptographic assumptions.

the computational effort for decrypting these items. The advantage of this approach is that no third party is needed at all. Like existing protocols for fair exchange of secrets, the disadvantages are that it requires many rounds and that it assumes that both participants have similar computational power.

Multiple protocols for a gradual exchange of specific secrets without third party have been proposed in the literature. Protocols for exchanging factorizations have been proposed by [Blu2 83, Tedr 85, Damg 95, Yao 86]. Protocols for exchanging discrete logarithms have been proposed in [BCDG 88, HaLi1 93, Damg 95]. [BCDG 88] reveals intervals and proves that the secret is in this interval. [HaLi1 93] reveals some bits of the representation of the secret value x itself. Single bits can be released by using the protocols proposed by [Tedr 84, Clev 90, LuMR 83]. Data described by one-way images is gradually exchanged by, e.g., [OkOh 94].

2.5.3 Some Relations among Instances of Fair Exchange

Some relations among different existing instantiations of fair exchange have been described in the literature[15]. The question we would like to answer is:

- Can any given instance of type A be used to construct an instance of type B?

One example is whether any certified mail service can be used to provide contract signing (which is true; see Part I), or whether any payment for receipt mechanism can be used to provide a fair purchase service.

Some relations between specific protocols for certified mail, fair exchange of data, and contract signing have been proven in [BlVV 84]. They define that a system Sys_A can be weakly reduced to a system Sys_B, iff Sys_B can be used to construct a system of type Sys_A. A reduction is "strong", if it requires an independent signature scheme $\mathsf{Sign}()$ and a cryptographically secure hash function $\mathcal{H}()$.

Fair Exchange of Secret can be used to provide Certified Mail: The strong reduction to construct a certified mail protocol from a fair exchange of secret data protocol and any signature scheme works as follows [BlVV 84]: The originator selects a random number s and then signs a message saying "I acknowledge having received the message which is pre-image to $f(m)$, iff anybody is able to present the pre-image of $f(s)$". Then, the secrets s and m are exchanged using the given fair exchange of data protocol. Note that this reduction makes additional assumptions about the one-way function: Normally, one only assumes that the one-way function is difficult to invert. This protocol makes the additional assumption that it keeps all bits of its input secret, i.e., that the recipient is not able to obtain parts of the message given the one-way image of it.

Fair Exchange of Secret can be used to provide Contract Signing: The weak reduction to construct a contract signing protocol given a fair exchange of secret protocol requires three-party disputes, i.e., a third party keeping track of invalid contracts: Each party, say A, signs a message guaranteeing that a given number f_A was computed as $f_A = f(\mathrm{sign}_\mathsf{A}(C))$[16]. Then, they exchange these

[15]Some more will be described in Part I.

[16]Even though this reduction is called "weak" in [BlVV 84], it assumes that "contract signatures" can be produced independently, i.e., they nevertheless assume the existence of a digital signature scheme. Thus, this reduction should rather be called strong.

pre-images using f_A and f_B as the descriptions of the expected secrets. If one of the parties later finds out that the pre-image was no valid contract, it contacts the third party and revokes the contract.

The other reductions proposed in [BlVV 84] are not generic: The weak reduction to construct a contract signing protocol using fair exchange of secrets assumes that all schemes are based on the same cryptosystem $(E_A(), D_A())$ where the same key-set is used for encryption (written as $m := D_A(E_A(m)))$ and signatures (written as $E_A(D_A(m)) \overset{?}{=} m$). The same holds for the weak reduction to construct a fair exchange of secrets from contract signing: It assumes that the exchanged signatures can be used as a one-time-pad on the secret to be exchanged. This reduction also needs a “court” since the resulting protocol does not verify that the one-way function applied to the given secrets s_A and s_B outputs the given images $f(s_A)$ and $f(s_B)$.

Part I

Optimal Efficiency of Optimistic Fair Exchange

Chapter 3

Optimal Efficiency of Optimistic Contract Signing

3.1 Introduction

A *contract* is a non-repudiable agreement on a given text [Blum 81]. A contract signing scheme includes at least three players and two protocols: Two signatories participate in a contract signing protocol “sign“ which fairly computes a contract, i.e., guarantees that either both or none of the signatories obtains a contract. This contract can then be used as input to a contract verification protocol “show“ to convince arbitrary verifiers such as a court that the signatories reached agreement on the given text.

Note that unlike cryptographic contract signing protocols [Blum 81], our notion does not tolerate uncertainty about the outcome. In the end, the user must have a definitive answer whether a valid contract was produced or not. Furthermore, we achieve deterministic fairness if the underlying digital signature scheme is secure.

In all practical schemes, contract signing involves an additional player, called third party. This party is (at least to some extent) trusted to behave correctly, thus playing the role of a notary in paper-based contract signing. A well-known protocol for contract signing by exchanging signatures via a third party works as follows [Rab1 83]: Both signatories send their signatures to the third party. The third party then verifies and forwards them. At the end, both signatories end up having two signatures on the contract which may be sent to any verifier for verification. In this and similar protocols, the third party has to be involved in *all* executions of the contract signing protocol.

In order to minimize this involvement while guaranteeing fairness, the concept of so called “optimistic contract signing” has been introduced [AsSW 97, BGMR 90]. The basic idea of optimistic schemes[1] is that the third party is not needed in the fault-less case: After the execution of the optimistic signing protocol, two correct signatories that agree on the contract to be signed always end up with a valid contract. Only if one of the signatories misbehaves, the third party is involved to decide on the validity of the contract.

[1]See also [BüPf 89, BüPf 90] for optimistic fair exchange of payments against goods or receipts.

	Model			Efficiency Analysis			Proof
Scheme	Op	C	TP	$C_A = C_B$ "(+)"	$C_A \neq C_B$	TP	in
Scheme 3.1	(+)	s	2v, sk	$\underline{\text{3m}}$ 3t	1m 5t	sl	Th. 3.3
Scheme 3.2	(+)	s	2v, sk	4m $\underline{\text{2t}}$	2m 4t	sl	Th. 3.3
[AsSW 98]	(+)	a	2v, sk	$\underline{\text{4m}}$ 4t	4m 4t	sk	Th. 3.4
Scheme 3.3	(+)	a	2v, sk	6m $\underline{\text{3t}}$	6m 5t	sk	Th. 3.5
Scheme 3.4	(+)	a	3v, sk	$\underline{\text{3m}}$ 3t	4m 4t	sk	Th. 3.6
Scheme 3.5	(+)	a	3v, sk	4m $\underline{\text{2t}}$	6m 5t	sk	Th. 3.6

Legend:

Op	"+" stands for optimistic on disagreement (Def. 3.4; optimized for all optimistic cases), "(+)" stands for optimistic on agreement (Def. 3.4; optimized for agreement).
C	Communication Model: "s" for synchronous, "a" for asynchronous.
TP	Properties of the third party: "3v" if the third party is allowed to participate in verification, "2v" if not, "sl" for state-less, "sk" for state-keeping.
Eff.	"4m 2t", e.g., means that during the "sign"-protocol, four correct messages were sent in time two. Underlined figures denote optimal efficiencies. "$C_A = C_B$" denotes that correct signatories input identical contracts, and "$C_A \neq C_B$" denotes different contracts.
Proof	Theorems in which optimality of the scheme is proven for the given case.

Table 3.1: Provably Optimal Schemes By Model.

3.1.1 Results of this Chapter

In this chapter, we examine the efficiency of optimistic contract signing based on digital signatures in detail.

At first, we define a variety of contract signing models: Optimistic contract signing can be designed for synchronous and asynchronous networks. It may need no third party if both signatories are correct and either agree or disagree (we call this optimistic on disagreement) or only if the correct signatories agree on the contract (we call this optimistic on agreement). Finally, the third party may be allowed or disallowed to participate in the verification of a contract. The third party is always allowed to keep state.

In Theorem 3.1 we show that no asynchronous optimistic contract signing scheme without a state-keeping third party exists. This proves that in our protocols, the third party keeps state only if needed. In Theorem 3.2, we show that the third party is always needed during the "sign"-protocol if the signatories misbehave, i.e., that no protocol is optimistic even in the case of faults.

We present provably time- and message-optimal optimistic contract signing protocols for all models identified above (see Tables 3.1, 3.2, and 3.3): For all our models, a time-optimal protocol requires $2k$ messages in time k whereas the message-optimal protocol requires $k + 1$ messages in time $k + 1$. For all

Model			Efficiency		Model			Efficiency
Op	C	TP	"+"		Op	C	TP	"+"
(+)	*	*	m $\underline{t}$	$\xrightarrow{\text{Scheme 3.6}}$	+	*	*	m+2 $\underline{(t+1)}$
(+)	*	*	$\underline{m}$ t	$\xrightarrow{\text{Scheme 3.7}}$	+	*	*	$\underline{(m+1)}$ t+1

Table 3.2: Adapting Contract Signing Schemes for Optimism on Disagreement (optimality proofs can be found in Th. 3.8-3.13, a legend in Table 3.1).

Theorem	Model			Results		
No.	C	TP	Op	$t^{(+)}$		$m^{(+)}$
Th. 3.1	a	3v, sl	(+)	Does not exist		
Th. 3.3	s	3v	(+)			≥ 3
	s	3v	(+)	≥ 3	$\leftarrow$	$= 3$
	s	3v	(+)	≥ 2		
Th. 3.4	a	2v	(+)			≥ 4
Th. 3.5	a	2v	(+)	≥ 3		
	a	2v	(+)	$= 3$	$\rightarrow$	≥ 6
Th. 3.6	a	3v	(+)			≥ 3
	a	3v	(+)	≥ 3	$\leftarrow$	$= 3$
	a	3v	(+)	≥ 2		

Theorem	Model			Results		
No.	C	T	Opt	t^{+}		m^{+}
Th. 3.8	s	3v	+			≥ 4
Th. 3.9	s	3v	+	≥ 3		
	s	3v	+	$= 3$	$\rightarrow$	≥ 6
Th. 3.10	a	2v	+			≥ 5
Th. 3.11	a	2v	+	$= 4$	$\rightarrow$	≥ 8
Th. 3.12	a	3v	+			≥ 4
Th. 3.13	a	3v	+	≥ 3		
	a	3v	+	$= 3$	$\rightarrow$	≥ 6

Table 3.3: Our Impossibility Theorems and What They Prove (a legend can be found in Table 3.1).

models, each message / time-optimal protocol in time $k + 1 / k$ is optimal with respect to time / messages given the message / time limitation.

At first, in Sections 3.4 and 3.5, we analyze the (hopefully) most likely case that both signatories are correct and agree. For our two weakest models (i.e., synchronous network, optimism on agreement only, with / without the third party participating in verification), we describe protocols for $k = 2$ and show that they are message and time optimal. On asynchronous networks, we obtain optimal protocols with $k = 2$ if the third party participates in the verification of the contract and $k = 2$, else. Since these protocols are required to contact the third party if two correct participants disagree, we then show in Section 3.6 how to adapt these schemes such that no third party is needed if the participants are correct but may as well disagree (see Table 3.2). This simulation increases k by 1.

Remark 3.1. As in [Lync 96], we do not consider the memory and computation complexity of our protocols. It is linear in the length of the inputs as well as the complexity of the cryptographic primitives and the constant factors can be made small. Therefore, we feel that the network latency will be one of the main factors determining the speed of the resulting protocol. ○

3.1.2 Notations and Assumptions

We assume that a digital signature scheme $(\mathsf{gen}, \mathrm{sign}_{\mathsf{A}}, \mathrm{test}_{\mathsf{A}})$ for a given message-space M and security parameter n is given [GoMR 88, DiHe 76]. The keys for each party, e.g., A, are computed by the probabilistic algorithm $\mathsf{gen}(\cdot)$. For a given message $msg \in M$, $s := \mathrm{sign}_{\mathsf{A}}(msg)$ denotes the signature of a player named A under a message msg. Such digital signatures computed by A can be verified using the corresponding verification function $\mathrm{test}_{\mathsf{A}}$, which is distributed using a given certification infrastructure. A signature s on a message m is called *valid* iff $\mathrm{test}_{\mathsf{A}}(s, m) = \mathsf{true}$. The security of the signature scheme guarantees authentication and non-repudiation, i.e., $\mathrm{sign}_{\mathsf{A}}$ can be used to compute a valid signature on m and without knowledge of $\mathrm{sign}_{\mathsf{A}}$, a polynomial-time adversary cannot compute valid signatures s' on messages m' not previously signed by A, except with negligible probability. In our protocols, we assume that A as well as msg can efficiently be computed given the signature. Furthermore, the security analysis of our protocols is done as if digital signatures would provide error-free authentication, i.e., we do not consider the negligible case that signatures may be broken.

With $|S|$ for any set S, we denote the number of the elements in set S.

We assume that signed messages are typed and labeled with the protocol parameters, e.g., that sending $m_2 = \mathrm{sign}_{\mathsf{O}}(text)$ in protocol "prot" using a third party T executed by machine S running with a tid started at time t_0 (if synchronous) to machine R actually sends the signed message $\mathrm{sign}_{\mathsf{O}}($"prot", S, R, T, $tid, t_0,$ "m_2", $text)$ in order to prevent interchanging messages between different protocols and runs (the identifier "m_2" denotes the unique name identifying message m_2; for clarity of our protocols, we may nevertheless mention some of the parameters that are included and signed automatically). Messages without this form or with unexpected parameters are simply ignored. In the synchronous case, messages that do not arrive in their designated round are ignored.

In our figures, $Ⓐ \xrightarrow{a/b} Ⓑ$ depicts that a machine is in state A and receives a message called a. It sends a messages called b and changes to state B. Sending multiple messages $b_1, b_2, \ldots$ is denoted by sending $b_1 + b_2 + \ldots$. The recipient of a message is not depicted. It is described in the text and can be determined by searching for an input of the given message at another machine. Dashed arrows denote non-optimistic exception handling. If the message name is bold, the message is exchanged with the third party. Subscripts in message names usually denote the time at which they are sent (e.g., m_3 would be a message from Round 3). Bold states are final states. If a message m_i is *not* received on a synchronous network, this is modeled by receiving the message $\neg m_i$.

Our figures depict the automata for one run with a given tid, only. This run is identified by the tid received in the first input or message. To enable parallel execution of multiple protocols, a new process needs to be started for each new tid received. A state-keeping third party, for example, will always listen to incoming messages. If it receives a message, it checks whether a process for this tid and these signatories exists. If this is the case, the message is forwarded to this process. Else, a new copy of the third party automaton is started in a new process and parameterized with the new tid.

In our text, the end of a definition or a scheme is marked with the symbol "$\diamond$", the end of a theorem or lemma is marked with "$\square$", the end of a proof is marked with "■", and the end of a list of remarks is marked with the symbol "○".

3.2 Definitions

We now define our network and complexity model as well as the notion of optimistic contract signing.

3.2.1 Network Models and Protocol Complexity

We distinguish between the "standard" synchronous and asynchronous network models [Lync 96]. On synchronous networks, messages are guaranteed to be delivered within a so-called "round", i.e., a recipient of a message can decide whether a message was sent or not[2]. On asynchronous networks, messages are eventually delivered but may be delayed and reordered arbitrarily. Therefore, a recipient cannot decide in general whether a message will eventually arrive or not. Therefore, our asynchronous protocols accept a distinguished input wakeup. After this input the protocol stops waiting for pending messages and guarantees termination within a limited time by only interacting with the third party that is assumed to be correct. In our optimistic protocols, this input signals the end of the optimistic phase and notifies the machine that the correct third party may be involved.

A machine is called *correct* if it adheres to its algorithm. We assume a byzantine failure model, i.e., a faulty machine may send arbitrary messages it is able to compute but must not be able to prevent delivery of messages between two correct machines. The time complexity of a synchronous protocol is the number of rounds required for its execution, i.e., the protocol complexity is the last round where a message may be sent or the output is made. The time complexity of an asynchronous protocol is the time required for its execution if transmission of each message requires time 1 and local computations require no time.

For both network types, we assume that each algorithm receives its messages from other algorithms and its local inputs, then does a computation on them and outputs at most one local output and one message for each other algorithm.

The time complexity of any protocol sketched above can be formalized by defining a logical time [Lync 96]. This complexity will be used in Definition 3.6 to define the complexity of optimistic protocols, which shall be optimized.

Definition 3.1 (Time Complexity)
The time complexity of a protocol is defined to be the highest clock assignment at the end of the protocol obtained by the following rules:

1. Each machine participating in the protocol has a time assignment $time \in \mathbb{N}$ and a mode assignment $mode \in \{\mathsf{send}, \mathsf{receive}\}$. In send-mode, messages can only be sent. In receive-mode, messages can only be received. Initially, $time := 0$ and $mode :=$ receive is assigned.

[2]Note that the reliability of the connection between the signatories is not needed for security rather than for correct optimism.

2. The $time$ assignment of an algorithm is increased whenever an event happens.

   Unlike Lamport's time-stamps [Lamp 78], an event here is defined as "changing from receive-mode to send-mode". Consecutive send or receive operations as well as changes from send to receive mode do not change the local clock.

3. The assigned $time$ of the local clock of the sender is assigned to every message sent.

4. Whenever a time-stamp higher than the local $time$ assignment has been assigned to a received message, the local $time$ assignment is set to the time assigned to the received message.

$\diamond$

Remark 3.2. In the synchronous model, the time complexity defined by Definition 3.1 equals the minimum number of rounds needed for the protocol. $\circ$

3.2.2 Optimistic Contract Signing

We now define different flavors of contract signing. Each flavor can then be optimized for time- or message-optimality on synchronous or asynchronous networks.

We assume that the signatories agreed on the unique and fresh tid and know their mutual identities before starting the protocol. A common method to guarantee this is to use a pair of two locally unique numbers as the global transaction identifier. Note that unlike Part III, we link subsequent protocol executions (e.g., "sign" and "show") using the input tid. Furthermore, we assume that on synchronous networks, the players agreed on a starting Round t_0 in which the protocol is started.

Definition 3.2 (Contract Signing Scheme)
A *contract signing scheme* for a contract space M with $|M| \geq 2$, an identifier space id_space, and a set of transaction identifiers $TIDs$ is a triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ of probabilistic interactive algorithms (such as probabilistic interactive Turing Machines) where V does not keep state between subsequent protocol runs. The algorithms A and B are called signatories, and V is called verifier. In addition, the scheme may specify a set AM of auxiliary machines without in- or outputs. The algorithms can carry out two interactive protocols:

Contract Signing (Protocol "sign"*):* Each signatory, e.g., A, obtains a local input $(\mathsf{sign}, \mathsf{B}, C_A, tid)$ where sign indicates that the "sign"-protocol shall be executed with signatory $\mathsf{B} \in id\_space$, $C_A \in M$ is the contract text A wants to sign, and $tid \in TIDs$ is the common unique transaction identifier which is used to distinguish different protocol runs. At the end, each of A and B returns a local output, which is either (signed, tid) or $(\mathsf{rejected}, tid)$.

Verification (Protocol "show"*):* This is the contract verification protocol between any particular verifier V and only one of the signatories A or B[3][4]. The signatory, say A, obtains a local input (show, tid). The verifier V outputs either (signed, A, B, C, tid) with the identities[5] of the two signatories and the contract text, or else (rejected, tid).

◇

Intuitively, an output signed of the "sign"-protocol means that the user can now safely act upon the assumption that the input contract has been signed, i.e., that a subsequent verification will succeed. If the protocol outputs rejected, the user can safely assume that no contract was signed, i.e., the other signatory will not be able to pass verification.

The set AM subsumes all auxiliary machines needed for the correct operation of the scheme. Since they do not make in- or outputs, there is no need to identify them individually. Examples include network machines, certification authorities, or third parties.

The requirements on termination of the protocols depend on the underlying network: On asynchronous networks, nobody can decide whether a message will eventually arrive or not. Thus, with incorrect players, termination cannot be guaranteed without precautions. Therefore, we allow the user to request termination manually: After a local input (wakeup, tid), the protocol stops waiting for pending messages and is required to terminate and produce a correct output (e.g., by only interacting with a correct third party).[6]

Remark 3.3. Most existing protocols only consider one single run of a protocol, i.e., they do not introduce transaction identifiers to distinguish two independent runs. ○

Definition 3.3 (Secure Contract Signing)
A contract signing scheme (Def. 3.2) is called secure if it fulfills the following requirements if the machines in AM are correct[7]:

Requirement 3.3a (Correct Execution): Consider an execution of "sign" by two correct signatories with an input (sign, B, C_A, tid) to A and (sign, A, C_B, tid) to B with a unique and fresh $tid \in TIDs$ and $C_A, C_B \in M$. If these inputs are made in the same round on synchronous networks and if wakeup is not input on asynchronous networks, the "sign"-protocol outputs (signed, tid) iff $C_A = C_B$ or else (rejected, tid) to both signatories.

Requirement 3.3b (Unforgeability of Contracts): If a correct signatory, say A, did not receive an input (sign, B, C, tid) so far, any correct verifier V will not output (signed, A, B, C, tid).

[3]Note that we did not consider verification including A *and* B, even though this may lead to more efficient protocols for some models (see Remark 3.15).

[4]Without this restriction, a signatory would not be enabled to switch off its machine as long as the contract is valid.

[5]We assume that the order of the names does not matter. E.g., a requirement that the verifier outputs (signed, A, B, C, tid) is also fulfilled if it actually outputs (signed, B, A, C, tid).

[6]Note that this termination may lead to a different outcome as compared to an undisturbed run of the protocol.

[7]Note that a specific system may weaken this strong trust assumption.

Requirement 3.3c (Verifiability of Valid Contracts): If a correct signatory, say A, outputs (signed, tid) on input (sign, B, C, tid) and later executes “show“ on input (show, tid) with a particular correct verifier V then this verifier V will output (signed, A, B, C, tid).

Requirement 3.3d (No Surprises with Invalid Contracts): If a correct signatory, say A, outputs (rejected, tid) on input (sign, B, C, tid) then no correct verifier will output (signed, A, B, C, tid).

Furthermore, one of the following requirements must be fulfilled:

Requirement 3.3e (Termination on Synchronous Network): On input of (sign, B, C, tid), a correct signatory, say A, will either output (signed, tid) or (rejected, tid) after a fixed number of rounds.

Requirement 3.3f (Termination on Asynchronous Network): On input of (sign, B, C, tid) and (wakeup, tid), a correct signatory, say A, will either output (signed, tid) or (rejected, tid) after a fixed time[8].

◇

Remark 3.4. Note that wakeup can only be input to the signatories during the “sign“ protocol. Termination of the “show“ protocol is implied by [R. 3.3c] if a contract was produced.[9]

Remark 3.5. If two correct signatories input different contracts $C_A \neq C_B$ on asynchronous networks and later input wakeup, then an output (rejected, tid) to both is not required by “correct execution”.

However, if the protocol would not output (rejected, tid) to both, then at least one signatory would obtain an output signed. For “verifiability”, this signatory would be able to convince the verifier. This, however, contradicts “unforgeability” of the other correct signatory who did not input the same contract. ○

Definition 3.4 (Optimistic Contract Signing with Two-Party Verification) An *optimistic contract signing scheme* for a contract space M, an identifier space id_space, and a set of transaction identifiers $TIDs$ is a triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ together with a machine T. The scheme must fulfill the following requirements:

Requirement 3.4a (Security): $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ with a set $AM := \{\mathsf{T}\}$ with a correct machine T is a secure contract signing scheme (Def. 3.3) where the third party does not participate in the “show“ protocol.

Requirement 3.4b (Limited Trust in T): R. 3.3b and R. 3.3c are fulfilled even if T is incorrect.[10]

Furthermore, one of the following requirements must be fulfilled:

[8]This is a logical time as defined in Def. 3.1. Note that “eventually” is too weak in our opinion: The term “fixed time”, requires that there exists a fixed bound on the maximum number of message exchanges after wakeup.

[9]Implicitly, this assumes that a user only executes the “show“-protocol after obtaining a contract.

[10]Intuitively, R. 3.3d cannot be guaranteed for an incorrect T. Since the outcome of one correct signatory must depend on the behavior of T (otherwise, T would not be needed), T could trick this signatory into an output rejected while enabling an output signed to its peer.

Requirement 3.4c (Optimistic on Agreement on Synchronous Network):
If two correct signatories input $(\mathsf{sign}, \mathsf{B}, C, \mathit{tid})$ and $(\mathsf{sign}, \mathsf{A}, C, \mathit{tid})$ in a given round with a fresh and unique tid and a $C \in M$, the third party does not send or receive messages in the "sign"-protocol.

Requirement 3.4d (Optimistic on Agreement on Asynchronous Network):
If two correct signatories input $(\mathsf{sign}, \mathsf{B}, C, \mathit{tid})$ and $(\mathsf{sign}, \mathsf{A}, C, \mathit{tid})$ with a fresh and unique tid and a $C \in M$ and do not input $(\mathsf{wakeup}, \mathit{tid})$, then the "sign"-protocol terminates in a fixed time and the third party does not send or receive messages.

An optimistic contract signing scheme is called *optimistic on disagreement* if R. 3.4c or R. 3.4d hold even if the correct signatories input different contract texts $C_A \neq C_B$. ◇

Remark 3.6. R. 3.3a for an incorrect T automatically holds in all optimistic executions (if T is not asked, it cannot influence the outcome). This means that if the participants are correct, agree, and do not input wakeup, all optimistic protocols will automatically produce a correct output signed. Only protocols that are optimistic on disagreement usually produce a correct output of rejected without contacting the third party. ○

A weaker definition allows the third party to participate in the verification of a contract. This enables, e.g., revocation of contracts during recovery:

Definition 3.5 (Optimistic Contract Signing with Three-party Verification)
An *optimistic contract signing scheme with three-party verification* for a contract space M, an identifier space $\mathit{id\_space}$, and a set of transaction identifiers TIDs is a triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ together with a machine T of probabilistic interactive algorithms. The scheme must fulfill the following requirements:

Requirement 3.5a (Security): $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ with with a set $AM := \{\mathsf{T}\}$ with a correct machine T is a secure contract signing scheme (Def. 3.3).

Requirement 3.5b (Limited Trust in T*):* R. 3.3b is fulfilled even if T is incorrect.

The scheme is called optimistic on agreement if R. 3.4c or R. 3.4d hold. It is called optimistic on disagreement if these requirements hold even if the correct signatories input different contract texts $C_A \neq C_B$. ◇

Remark 3.7. Note that the involvement of the third party in the verification is not as bad as its involvement in the actual contract signing protocol: In practice, only few signed contracts should be disputed at court.

Remark 3.8. Including the other signatory into the verification changes the resulting optimal protocols, since the third party can ask the other signatory to, e.g., prove dishonesty of the signatory showing its contract. ○

3.2.3 Complexity of Optimistic Schemes

The complexity of optimistic protocols that will be optimized in the sequel is defined as follows:

Definition 3.6 (Complexity of Optimistic Schemes)
Let $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T be an optimistic scheme.

The optimistic message complexity of the scheme is defined to be the maximum number of messages sent/received by correct players in all optimistic cases[11].

The optimistic time complexity of the scheme is defined to be the maximum time where a correct player may be required to send or receive messages in all optimistic cases. More precisely, it is the time where one can safely switch the machine off in all optimistic cases. ◇

Remark 3.9. The fine distinction in the definition, i.e., that it does not simply state "the maximum time where a player sends or receives a message" is needed for cases where a player in the optimistic case waits for a message which will only arrive in the non-optimistic case. In this case he will not receive a message in the optimistic case, nevertheless the machine cannot yet be switched off in the optimistic case because it is not yet clear to the machine that it is in this case. Hence our definition assigns a higher time complexity to this case than the simple definition would.

Such situations may occur in the synchronous case, where the third party may ask both participants to show evidence during a non-optimistic execution. Therefore, e.g., a correct participant who already output signed and has finished in the optimistic execution may be required to prove this decision to the third party in a non-optimistic execution. If the protocol includes such an "ask-back", the time of the last ask-back message is the time complexity of the protocol.

E.g., Scheme 3.1 below would also be optimal for optimism on disagreement in the simpler definition: In case of disagreement, m_1 is the only message sent. The fact that A must wait until Round 5 is not considered by this complexity measure.

Remark 3.10. Local in- and outputs are no messages in our sense.

Remark 3.11. Yet another time measures in the synchronous case is the time of the output of the decision. ○

Lemma 3.1 (Using Asynchronous Protocols on Synchronous Networks)
Any asynchronous contract signing protocol that is optimistic on agreement/disagreement in time t with m messages can be used as a synchronous contract signing protocol in t rounds and m messages that is optimistic on agreement/disagreement. □

Proof. The asynchronous protocol is adapted as follows:

1. A correct player sends an outgoing message immediately whenever all preconditions are fulfilled (i.e., a message m_i will usually be sent in Round i).

2. After the number t of rounds given by the optimistic time complexity, a player automatically inputs wakeup to the asynchronous protocol.

[11] I.e., for "optimistic on disagreement", the complexity is defined to be the maximum of the agreement and the disagreement case.

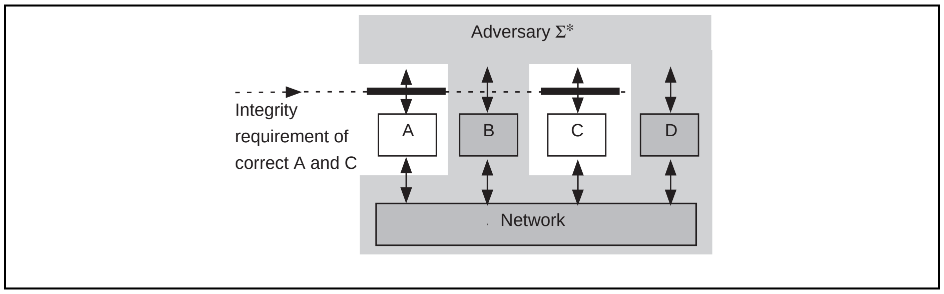


Figure 3.1: Evaluating an Integrity Requirement (the adversary Σ^* is depicted gray, the correct machines Σ' white, and the predicate as black bars).

It is clear that the synchronous protocol fulfills all requirements where occurrences of wakeup are not explicitly mentioned.

Correct execution for the optimistic cases follows with asynchronous optimism from the fact that wakeup is not input while the protocol execution can still be optimistic.[12] If $C_A \neq C_B$ for a protocol that is optimistic on agreement only, we input wakeup automatically even though the protocol may still be running. In this case, the output $(\mathsf{rejected}, tid)$ to both signatories follows from the argument in Remark 26.

Synchronous termination follows from asynchronous termination since wakeup will be input automatically (see Item 2). Optimism on synchronous networks holds because the input wakeup is only made at a time where all optimistic executions have finished. ■

3.2.4 Traditional Formalization of the Integrity Requirements

We now sketch a formal model for evaluating the requirements listed above which follows the ideas in [Pfit8 96, PfWa 94]. In Part II, we will describe a new alternative for defining integrity and secrecy using *Trusted Hosts* specifying the intended service.

The basic idea of this traditional formalization is that for each requirement, a given subset of the machines is assumed to work correctly. Integrity requirements are then expressed over sequences of in- and outputs of these correct machines: No matter what the incorrect machines do, the probability of the requirements not being fulfilled should be negligible. This simple approach towards formalizing requirements can be used since we do not deal with secrecy requirements. Note that as in [Pfit8 96], we require integrity even if the users behave not as expected. We simulate this by allowing the adversary to choose the user-inputs of all users. The set-up underlying the following definition is depicted in Figure 3.1.

Definition 3.7 (Fulfillment of Integrity Requirement)
Let $\Sigma(n) = \{A_1(n), A_2(n), \ldots, A_k(n)\}$ be a given scheme consisting of a list of interacting machines that are polynomial-time in an input security parameter n. Let $\Sigma'(n) \subseteq \Sigma(n)$ be the subset of machines that are assumed to be correct.

[12]Since all optimistic executions terminate before time $t+1$ and wakeup is input at time $t+1$, wakeup can only be input in non-optimistic executions.

An integrity requirement is a predicate on the in- and outputs of $\Sigma'(n)$. It is fulfilled if the probability of winning the following game is negligible[13] for any polynomial-time machine $\Sigma^*(n)$ replacing all machines in $\Sigma(n) - \Sigma'(n)$:

1. $\Sigma^*(n)$ interacts arbitrarily with the machines in $\Sigma'(n)$. This includes $\Sigma^*(n)$ generating user-inputs.
2. The adversary $\Sigma^*(n)$ wins the game if the predicate is *not* fulfilled by the resulting user in- and outputs of the correct machines in $\Sigma(n)$.

$\diamond$

3.3 Some Basic Impossibility Results

We now describe some basic limitations of optimistic contract signing.

3.3.1 On Asynchronous Recovery with a Third Party

The third party has to keep state and asynchronous recovery[14] never involves both signatories:

Lemma 3.2 (Asynchronous Recovery is 2-party)
After an input wakeup *on asynchronous networks, the outcome of the* “sign“-*protocol is determined only by the states and inputs from the third party and the signatory starting it.* $\square$

Proof. If the third party is invoked by a correct player, the recovery phase is required to terminate in order to guarantee termination of the “sign“-protocol. However, if the third party asks the other signatory, the third party cannot decide on asynchronous networks whether the message would eventually be answered or not. Thus, if the third party would wait forever and the signatory is not correct, “resolve“ would not terminate. $\blacksquare$

A state-keeping third party cannot be avoided on asynchronous networks:

Theorem 3.1 (Asynchronous T Keeps State)
There is no asynchronous contract signing scheme with state-less third party[15]*, which is optimistic on agreement.* $\square$

Proof. Assume there is an asynchronous optimistic contract signing scheme with this property. Then there is an equivalent “sign“-protocol which has only messages, say $m_1, \ldots, m_n$, in a row where A sends m_1 and m_n (if not, prepending an empty message helps): Messages sent in parallel are independent of each other. The security of the scheme must not depend on their order since the asynchronous network may reorder the messages. Therefore, the protocol can be converted into subsequent messages by shoving messages up or down (see Figure 3.2).

[13] I.e., for each polynomial $p(n)$ and all sufficiently large n, the probability of winning the game is smaller than $1/p(n)$.

[14] With *recovery* we denote the non-optimistic phase involving the third party. On asynchronous networks, t is started by an input wakeup. Else, by messages that have not been received.

[15] For a state-less third party, the answer to any message does not depend on earlier messages that have been processed.

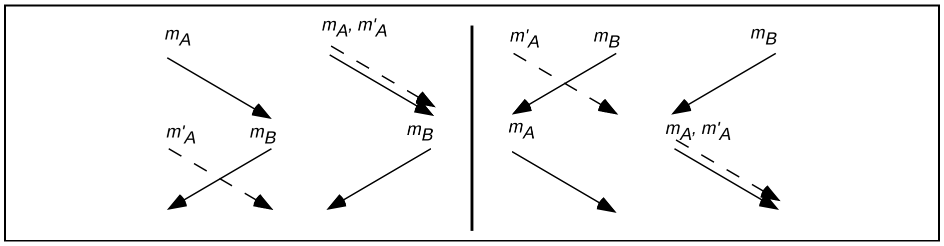


Figure 3.2: Saving a Dashed Message by Shoving it Up or Down.

Furthermore, we assume that after an input wakeup, the invoker (either A or B) sends all messages the invoker has sent or received so far to the third party, i.e., a prefix $(m_1, \ldots, m_k)$ of $(m_1, \ldots, m_n)$. Since we are in an asynchronous model, the third party's decision cannot depend on the non-invoking signatory (Lemma 3.2). Since the third party is assumed to be state-less, the decision can be modeled as a set of functions $TP()$ on $(m_1, \ldots, m_k)$ to $\{\text{signed}, \text{rejected}\}$ for each k for which a request is allowed.

Consider a run with correct A and B where both input identical contracts and B inputs wakeup after m_{n-1} and before the last message m_n from A has been received. Since A may have received a valid contract, the third party must decide $TP(m_1, \ldots, m_k) :=$ signed for $k = n-1$ to fulfill the "No Surprises"-requirement.

Now assume that $TP(m_1, \ldots, m_k) =$ signed for some $k \geq 2$. If we now consider the case that one player gets a wakeup after sending m_{k-1}, a recovery request must be allowed since the other player will eventually receive m_{k-1} which enables it to recover to signed. For consistency reasons, we have $TP(m_1, \ldots, m_{k-1}) := TP(m_1, \ldots, m_k)$=signed. Thus, inductively we get $TP(m_1) =$ signed which contradicts the unforgeability requirement. ■

3.3.2 On Two-party Contract Signing without Third Party

In this theorem, we extend the impossibility results from [Even 83] to include protocols with state-keeping verifier. In our language, this corresponds to a scheme with four-party verification and state-keeping verifier where the third party does not participate in the "sign"-protocol at all, i.e., even if one of the players is dishonest[16].

Theorem 3.2 (No 2-party Contract Signing)
There exists no synchronous contract signing scheme with three-party verification (Definition 3.5) including both signatories[17] and state-keeping verifier, which is optimistic even in case of faults.[18] □

Remark 3.12. Recall that asynchronous protocols can be used on synchronous networks, too (see Section 3.2.1). As a consequence, this theorem proves that no such protocol exists on asynchronous networks as well. ○

[16]Recall that contract signing protocols that are optimistic on disagreement never need the third party if the signatories are correct and do not input wakeup.

[17]Recall, in Definition 3.5 only one signatory participates in the verification, hence this theorem is more general.

[18]This implies that the third party does not send or receive messages even in case of one faulty signatory.

Proof. If there were such a protocol, in the “sign“-protocol, the signatories would interact only with each other. Let us assume that all messages of the “sign“-protocol are sent in a row and that A sends the first and the last message (if not, delaying or appending some messages helps). Since a correct party must not contact the third party or the verifier, we can w.l.o.g. assume that they ignore all other messages during the “sign“-protocol. Let us assume that T is unconditionally trusted (i.e., unlimited trust in T; this is possible since it makes the impossibility result even stronger) and that the verifier V plays the role of T as well (i.e., we use a state-keeping verifier instead of a state-keeping third party and a state-less verifier that interact).

Let $(m_1, \dots, m_n)$ be the correct messages that are exchanged in a correct execution of “sign“ with matching parameters (i.e., they include identical participant identifiers, contracts, and tids).

In the first execution of the “show“ protocol for a particular tid, the verifier V as well as the machines A and B then perform a dialogue to derive the decision on a given prefix of messages $m_1, \dots, m_k$.

Let us consider a correct signatory A: If A receives all protocol messages $(m_1, \dots, m_n)$ correctly and with matching parameters, its output must be signed. For unforgeability, the output on (m_1) must be rejected. Thus, for A there exists a message m_i such that receiving this message changes its output from rejected to signed. Furthermore, for “no surprises”, a correct B would output signed after sending m_i.

Let us now consider an initial decision of the verifier for B if signatory A presents a prefix $m_1, \dots, m_{i-1}$ while signatory B presents a prefix $m_1, \dots, m_i$.

If the verifier decides on rejected, this contradicts no surprises for a correct B, since an incorrect A may deny receiving m_i.

If the verifier decides on signed, this contradicts no surprises for a correct A, since an incorrect B may claim that it sent m_i without actually sending it, i.e., the correct A output rejected while the verifier decides on signed. ■

3.4 Optimistic on Agreement

We now describe contract signing protocols which are optimistic in case of agreement. For these protocols, we aim at optimizing the optimistic case, i.e., we prove that their efficiency is optimal if the participants are correct and agree on the contract text.

3.4.1 A Message-optimal Synchronous Scheme

The message-optimal optimistic scheme[19] for synchronous networks requires three messages in the optimistic case using a state-less third party. Its optimistic behavior is depicted in Figure 3.3. The individual machines of the players are depicted in Figures 3.4, 3.5, and 3.6.

Scheme 3.1 (Message-optimal Synchronous)
This scheme consists of the triple (A, B, V) and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

[19]The message flows are similar to the optimistic protocol in [Mica 97] which provides certified mail instead of contract signing.

Contract Signing (Protocol "sign"*; Figure 3.3):* On input (sign, B, C_A, tid), the signatory A initiates the protocol by sending the signed message $m_1 :=$ sign_{A}(A, B, T, t_0, C_A, tid) with contract C_A to the responding signatory B. B receives the input (sign, A, C_B, tid) and message m_1 and verifies whether the received contract text C_A is identical to the input contract text C_B. If not, the players disagree about the contract and B returns (rejected, tid). Else, it signs the received message and sends it as $m_2 :=$ $\text{sign}_{\mathsf{B}}(m_1)$ to A. Player A then signs the received message again, sends it back as $m_3 := \text{sign}_{\mathsf{A}}(m_2)$ and outputs (signed, tid). On receipt of message m_3, B outputs (signed, tid) as well. After a successful execution of this optimistic protocol, A and B store m_3 under the input tid for later use in a verification protocol.

If A does not receive message m_2 it waits until Round 5, and, if m_5 is not received, it outputs (rejected, tid). If B did not receive message m_3, it may be that A nevertheless was able to compute a valid contract m_3 after receiving m_2. Therefore it starts the "resolve"-protocol to invoke the third party to guarantee fairness.

Recovery from Exceptions (Sub-protocol "resolve"*):* B sends a message $m_4 :=$ $\text{sign}_{\mathsf{B}}(m_2)$ containing m_1 and m_2 to the third party T. The third party then checks whether both players have agreed and then forwards m_2 in $m_5 := m_2$ to A, which might still wait for it. This guarantees that A receives a valid contract $m_3 := \text{sign}_{\mathsf{A}}(m_2)$ and outputs (signed, tid). Furthermore T sends an affidavit on m_2 in $m_5' := \text{sign}_{\mathsf{T}}(m_2)$ to B and B outputs (signed, tid). After the "resolve"-protocol, A keeps m_3 and B keeps m_5' to be used in later verification protocol executions.

Verification of a Contract (Protocol "show"*):* On input (show, tid), a signatory looks up m_3 or m_5' and sends it to the verifier. The verifier verifies it and outputs (signed, A, B, C, tid) if this succeeds and (rejected, tid) else.

◇

Remark 3.13. Note that in our protocols, the contract and the contents of most messages are fixed after the first message sent by a signatory. Therefore, each player can save signatures by including one-way images $\mathcal{H}(r_i)$ of random authenticators r_i into the initial message which can then be released instead of signing subsequent messages [AsSW 97].

Remark 3.14. For simplicity, our protocols do not keep the contract confidential against the third party. This can easily be improved by signing a contract on $\mathcal{H}(C)$ and modifying the verification as follows: For verifying a contract, a signatory sends C to the verifier and inputs (show, tid) to the contract signing scheme. The verifier then outputs (signed, A, B, C, tid) if the verification outputs (signed, A, B, h_C, tid) and $h_C = \mathcal{H}(C)$ for the received C. ○

We now prove the security of the scheme. Note that in [Veit 99], we also conducted a more formal analysis of this protocol using the model checker "SPASS" [Weid 97, Weid 98].

Lemma 3.3 (Security of Scheme 3.1)
Scheme 3.1 is a secure synchronous optimistic contract signing scheme, which is optimistic on agreement. □

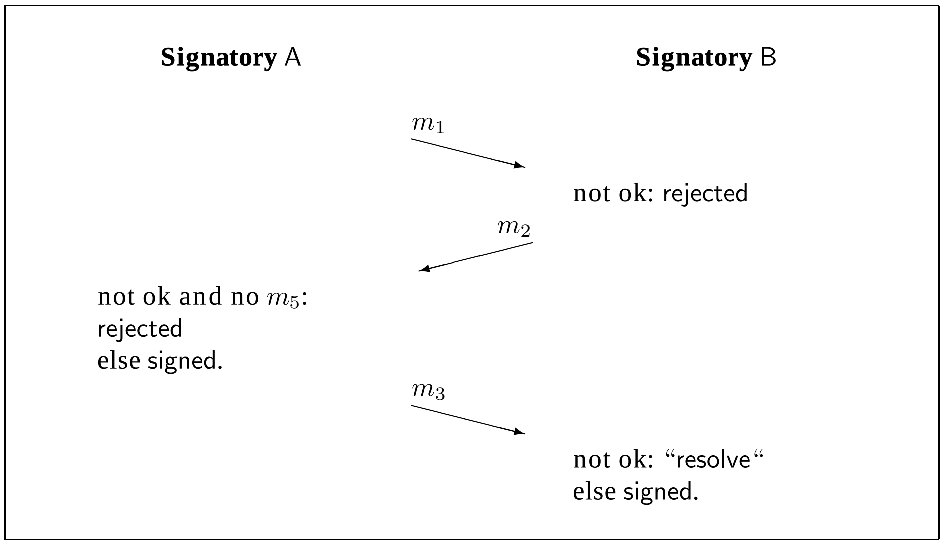


Figure 3.3: Optimistic Behavior of Scheme 3.1.

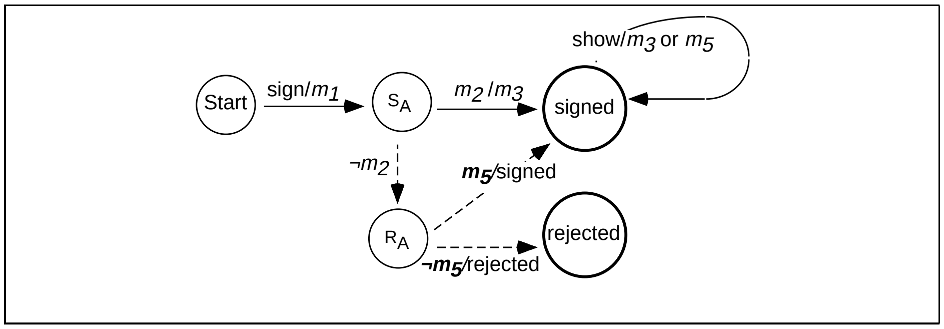


Figure 3.4: Signatory A of Scheme 3.1.

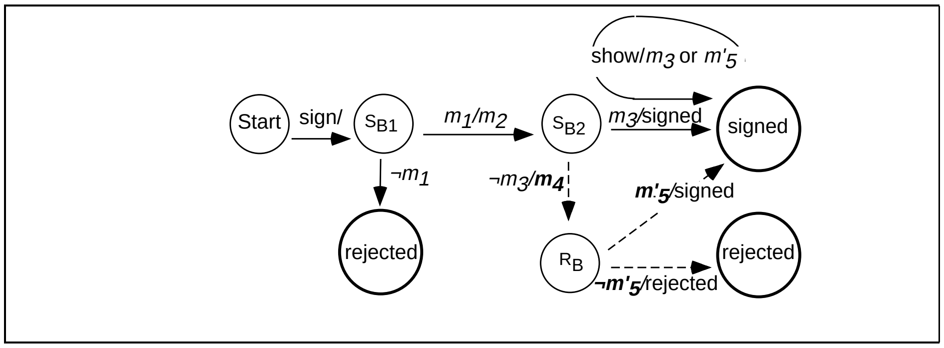


Figure 3.5: Signatory B of Scheme 3.1.

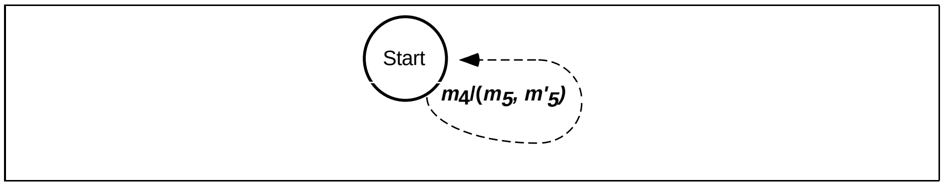


Figure 3.6: Third Party T of Scheme 3.1.

Proof. The scheme adheres to Definition 3.2 by construction. We now show that each of the requirements described in Definitions 3.3 and 3.4 are fulfilled:

Correct Execution: If both correct players A and B input (sign, B, C_A, tid) and (sign, A, C_B, tid) with identical tids and $C_A = C_B$, then both receive a valid contract m_3 and output (signed, tid). If the contracts or tid's differ, B outputs (rejected, tid_B) after receiving m_1 and A outputs (rejected, tid_A) after not receiving m_5 in Round 5.

Unforgeability of Contracts: In order to convince a correct verifier V for a given tid, C, and partner, one needs correct messages m_3 or m'_5 containing this tid. Since m_3 as well as m'_5 contain signatures from both participants, a correct signatory input (sign, A, C, tid) or (sign, B, C, tid), respectively.

Verifiability of Valid Contracts: If a correct machine A outputs (signed, tid) then it received m_2 (or m_5 containing m_2) which will be accepted by the verifier as a correct contract m_3 after being signed by A. B outputs (signed, tid) only if it received m_3 or m'_5 which are accepted by the verifier, too.

No Surprises with Invalid Contracts: Let us first assume that a correct signatory A returned (rejected, tid) on input (sign, B, C, tid) whereas B is able to convince the verifier. This requires that B knows m_3 or m'_5 for the given tid and C. Since A returned rejected, it did not receive m_2 until Round 5 and it did not send m_3. Therefore, only m'_5 could lead to successful verification. However, if the third party was correct, it will not accept recovery requests from B after Round 4[20]. Furthermore, in Round 4, no recovery was started since A did not receive m_5 in Round 5. Thus B did not receive m'_5 in Round 5.

Let us now assume that a correct signatory B returned (rejected, tid) on input (sign, A, C, tid). Then B either did not send m_2 for this tid or it sent m_4 to T. In the first case, A cannot convince the verifier since m_2 is part of m_3 and also m'_5. In the second case, a correct third party would necessarily have answered with m'_5 and thus B would not have returned (rejected, tid).

Termination on Synchronous Network: The scheme requires at most 5 rounds (3 in "sign" and 2 in "resolve") to terminate.

Limited Trust in T*:* Even if T is incorrect, it cannot forge any signature of the two signatories. Therefore, unforgeability still holds.

Verification is independent of T. Therefore, verifiability even holds if T is incorrect.

Optimistic on Disagreement on Synchronous Network: If the correct signatories input (sign, B, C_A, tid) and (sign, A, C_B, tid) with $C_A = C_B$, signatory A outputs (signed, tid) after Round 2 whereas player B outputs (signed, tid) after Round 3. In this case, the scheme sends the m_1, m_2, m_3 and requires 3 rounds.

[20]This round number is relative to the time t_0 as fixed by the initial input of A and implicitly included and signed in m_1 (see Section 3.1.2).

> If $C_A \neq C_B$, A outputs $(\mathsf{rejected}, \mathit{tid})$ after Round 5 and B after Round 1 without contacting the third party by starting "resolve". In this case, the scheme sends m_1 only but requires 5 rounds.

■

We now show that no optimistic contract signing scheme with only two messages exists. This proves that the number of messages of Scheme 3.1 is optimal. Furthermore, we show that it cannot be done with three messages in two rounds. Thus, the number of rounds of Scheme 3.1 is optimal, too, given the restriction to 3 messages.

Theorem 3.3 (Optimality of Scheme 3.1)
In the synchronous model with three-party verification[21]*, there exists no contract signing scheme which is optimistic on agreement with a* "sign"*-protocol with less than 3 messages in case of agreement, and a protocol which needs 3 messages needs at least 3 rounds.* □

Proof. Let us assume that there exists an optimistic contract signing scheme which requires three messages in two rounds in case of agreement. In the optimistic phase, one player, say B, sends two messages m_{1B} in Round 1 and m_{2B} in Round 2 whereas the other player sends one single message m_A in Round 1 or 2.

Let us first assume that A sends its single message m_A in Round 1. Since two correct players who input identical contracts $C_A = C_B$ must not contact the third party, the single message m_A from A needs to be sufficient to enable B to convince the verifier in the optimistic case. Now assume that an incorrect B receives the valid contract m_A but sends nothing. Then A must either be able to obtain a valid contract or else, the third party is required to revoke the contract, i.e., invalidate m_A.

If A obtains a contract, this contradicts the "unforgeability" for "limited trust" requirements, since A and T could forge the resulting contract that does not contain any inputs from B.

If we now assume that the contract is revoked (e.g., by storing the tid in a revocation list at the third party) then an incorrect A may revoke a valid contract m_A of a correct B. (Recall, in the optimistic case, a correct B output signed after Round 2 and does not answer subsequent recovery requests, which cannot arrive before Round 3.)

If we now assume, on the other hand, that A sends m_A in Round 2, then m_A and (m_{1B}, m_{2B}) must be valid contracts, i.e., sufficient for "show". If A now omits sending m_A, it will end up with a valid contract. Therefore B must be enabled to run "resolve". The resulting recovery without any message from A, however, again contradicts the "unforgeability" for "limited trust" requirements (again, the contract cannot be revoked since A terminated already). Thus no protocol with 3 messages in 2 rounds exists.

If a two-message scheme existed, adding an empty message would produce a 3 message scheme in 2 rounds which does not exist. ■

[21]See Def. 3.5. Note that this is a stronger result than actually needed for proving the optimality of Scheme 3.1. It shows that even with three-party one cannot do better.

Remark 3.15. This optimality proof only holds for our particular complexity measure. [Even 83], describes a protocol with only two messages in one round in case of agreement. However, since the signatories are required to answer recovery messages up to Round 3 (i.e., two rounds after the output signed), it is a three-round protocol in our model. ○

3.4.2 A Round-optimal Synchronous Scheme

We now describe the round-optimal Scheme 3.2 for synchronous networks and prove its security in Lemma 3.4. It requires only two rounds but four messages. Since any three-message "sign"-protocol needs at least three rounds (Theorem 3.3), there exists no one-round protocol at all and no 2-round protocol with only three messages. So the scheme described is optimal with respect to rounds and given the limitation to two rounds also with respect to the number of messages. The optimistic behavior of the scheme is depicted in Figure 3.7. The players are depicted in Figures 3.8 and 3.9.

Scheme 3.2 (Round-optimal Synchronous)
This scheme consists of the triple (A, B, V) and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "sign"*; Figure 3.7):* On input (sign, B, C_A, tid) a signatory, say A, sends message $m_{1A} := \mathrm{sign}_{\mathsf{A}}(\mathsf{A}, \mathsf{B}, \mathsf{T}, t_0, C_A, tid)$ with the protocol parameters in the first round. If it does not receive a message m_{1B} with $C_A = C_B$, it waits for recovery message m_4 and outputs (rejected, tid) if m_4 is not received in Round 4. If a message m_{1B} with $C_A = C_B$ is received, A sends $m_{2A} := \mathrm{sign}_{\mathsf{A}}(m_{1A}, m_{1B})$ in the second round and waits for m_{2B}. If m_{2B} with a correct contract text $C_A = C_B$ is received, it outputs (signed, tid). Else, it starts "resolve".

Recovery from Exceptions (Sub-protocol "resolve"*):* A signatory, say A, sends message $m_{3A}{:=}m_{2A}$ to the third party which verifies its consistency and signs an affidavit. This affidavit is sent as $m_4 := \mathrm{sign}_{\mathsf{T}}(m_{2A})$ to both parties. If the parties receive an affidavit in Round 4, they output (signed, C, tid). Else, they output (rejected, tid).

Verification of a Contract (Protocol "show"*):* On input (show, tid), a signatory, say A, looks up (m_{2A}, m_{2B}) or m_4 and sends it to the verifier V. The verifier checks that the signatures are correct. If these checks fail, it outputs (rejected, tid) and else (signed, C, tid).

◇

Remark 3.16. By involving both signatories in the "resolve" sub-protocol, the scheme could output a decision after time 2 in any case: If, e.g., A output rejected and B tries to recover to signed while using another $m'_{1B} \neq m_{1B}$, machine A could show the "real" m_{1B} and thus force the third party to abort recovery for a cheating B. However, the resulting scheme is not optimal for optimism on agreement since after an output signed, a correct signatory is required to wait for a recovery message that may arrive in Round 3. ○

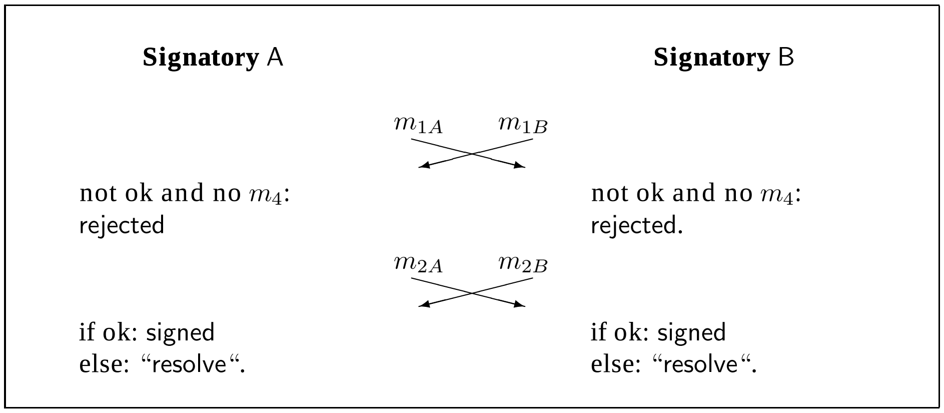


Figure 3.7: Optimistic Behavior of Scheme 3.2.

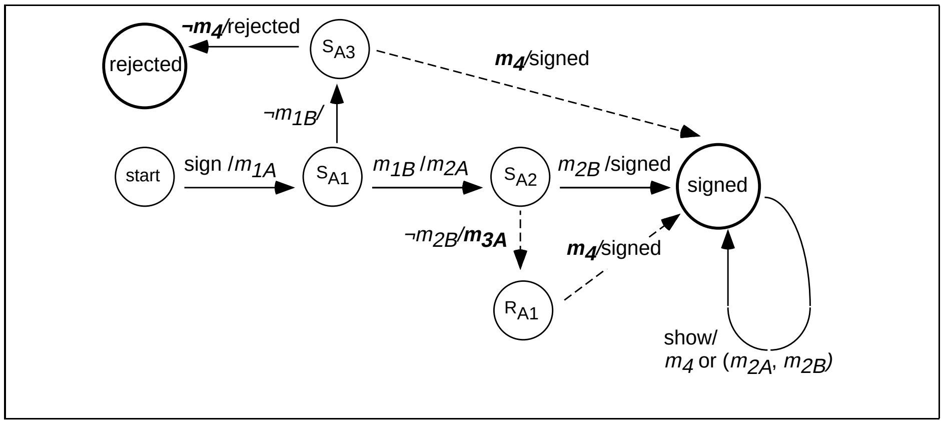


Figure 3.8: Signatory, e.g., A, of Scheme 3.2.

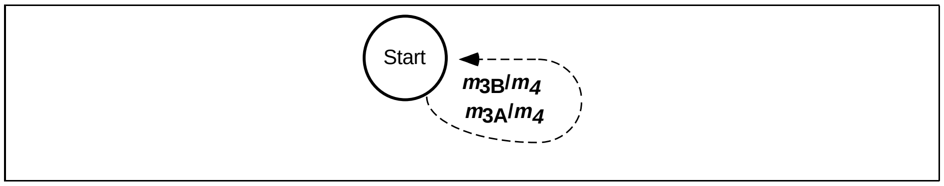


Figure 3.9: Third Party T of Scheme 3.2.

Lemma 3.4 (Security of Scheme 3.2)
Scheme 3.2 is a secure synchronous contract signing scheme, which is optimistic on agreement. □

Proof. The scheme adheres to Definition 3.2 by construction. We now show that it fulfills the requirements stated in Definitions 3.3 and 3.4:

Correct Execution: If both players behave correctly and input identical contracts, each signatory, say A, receives m_{1B} and m_{2B}. Thus, the protocol outputs (signed, tid) on both machines. If the signatories disagree, both will receive inconsistent messages in Round 1 and will wait for recovery until Round 4. Since no recovery message m_4 will be received, they will output (rejected, tid).

Unforgeability of Contracts: In order to convince a correct verifier, a signatory, say A, needs (m_{2A}, m_{2B}) or m_4. Since (m_{2A}, m_{2B}) as well as m_4 contain signatures from both signatories, a correct signatory input (sign, A, C, tid) or (sign, B, C, tid), respectively.

Verifiability of Valid Contracts: A signatory, say A, only outputs (signed, tid) after receiving m_4 or after sending m_{2A} and receiving m_{2B}. Thus, they are able to convince the verifier.

No Surprises with Invalid Contracts: If a signatory, say A, outputs rejected, this signatory did not start "resolve" and did not receive m_4 in Round 4 which means that B also did not receive m_4. In order to convince a verifier, B therefore needs m_{2A}. However, since A output rejected, it did not send m_{2A}.

Termination on Synchronous Network: At most 4 rounds are required for termination.

Limited Trust in T*:* Even if T is incorrect, it cannot forge any signature of the two signatories. Therefore, unforgeability still holds.

Verification is independent of T. Therefore, verifiability even holds if T is incorrect.

Optimistic on Disagreement on Synchronous Network: If two correct signatories input (sign, B, C_A, tid) and (sign, A, C_B, tid), iff $C_A = C_B$ they output (signed, tid) after Round 2 without contacting the third party. In this case, the scheme sends $m_{1A}, m_{1B}, m_{2A}, m_{2B}$ and requires 2 rounds.

If the signatories are correct and disagree, they send m_{1A} and m_{1B} and output (rejected, tid) after Round 4 without contacting the third party.

■

3.4.3 A Time-optimal Asynchronous Scheme

We now describe a new time-optimal asynchronous contract signing scheme. It terminates in time 3 and requires six messages in the optimistic case. In Theorem 3.5 we prove that this is time-optimal. Its optimistic behavior is sketched in Figure 3.10, the machines are depicted in Figures 3.11 and 3.12. Note that the third party is state-keeping: Once a contract is accepted (i.e., m'_5 or m''_5 was sent), the third party enters its signed state which disables aborting the protocol. A state-less third party would be more convenient, but we prove in Theorem 3.1 that this is not possible.

A message-optimal scheme has been proposed in [AsSW 98]. It requires four consecutive messages and time four in case of agreement. This is message-optimal in the optimistic case since, as we will prove, there is no asynchronous optimistic contract signing scheme with only three messages (Theorem 3.4).

On disagreement, the scheme as described in [AsSW 98] does not terminate without an input wakeup. As proposed in [AsSW 98], this can be fixed by an additional disagreement message. With this fix, the scheme requires 4 messages in time 4 in case of disagreement while involving the third party to abort the protocol run.

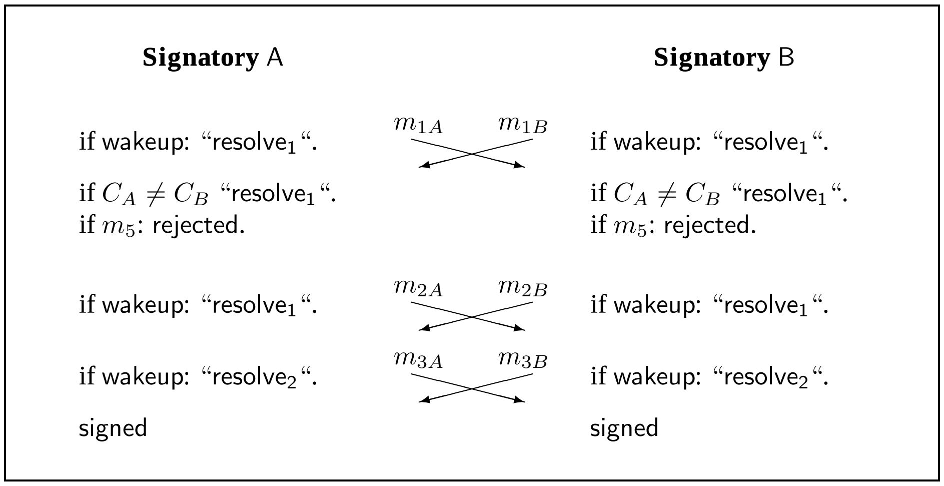
Scheme 3.3 (Time-optimal Asynchronous)
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "sign"*; Figure 3.10):* On input (sign, B, C_A, tid) the signatory, say A, sends its signed contract in message $m_{1A} := \text{sign}_\mathsf{A}(\mathsf{A}, \mathsf{B}, \mathsf{T}, C_A, tid)$. If A receives m_{1B} with an identical contract, it sends $m_{2A} := \text{sign}_\mathsf{A}(m_{1A},\ m_{1B})$. If a message m_{2B} from B is received, A sends $m_{3A} := \text{sign}_\mathsf{A}(m_{2A}, m_{2B})$. After receiving m_{3B}, the signatory outputs (signed, tid). If m_{2B} is received before m_{1B} since the messages have been reordered by the asynchronous network, both m_{2A} and m_{3A} are sent (in this case, m_{2A} is composed using m_{1B} as included in m_{2B}). If m_{3B} is received before m_{2B}, m_{3A} is sent and (signed, tid) is output.

If an m_{1B} with a different contract is received before m_{2B} or if (wakeup, tid) occurs before m_{2A} has been sent, "resolve_1" is started by sending $m_{4A} := \text{sign}_\mathsf{A}(m_{1A})$. If wakeup occurs after m_{2A} has been sent but before m_{3A}, "resolve_1" is started by sending $m'_{4A} := \text{sign}_\mathsf{A}(m_{2A})$. If wakeup occurs after m_{3A} has been sent, "resolve_2" is started by sending $m''_{4A} := \text{sign}_\mathsf{A}(m_{3A})$. Messages m_{2B} or m_{3B} from a cheating player B containing different contracts $C_A \neq C_B$ are ignored.

Recovery from Exceptions (Sub-protocol "resolve_1"*):* This protocol is used in a situation where the status of a contract may not be clear. If the signatory sends m_{4A}, the third party either re-sends a previously sent decision m_5, m'_5 or m''_5 for this tid or else an abort acknowledgment $m_5 := \text{sign}_\mathsf{T}(m_{4A})$ and changes to the aborted-state for the aborting signatory. If the signatory sends m'_4, the third party either re-sends a previous decision m_5, m'_5, or m''_5 or else signs an affidavit $m'_5 := \text{sign}_\mathsf{T}(m'_4)$. After receiving m_5, the signatory outputs (rejected, tid). After receiving m'_5 or m''_5, the signatory outputs (signed, tid).

Recovery from Exceptions (Sub-protocol "resolve_2"*):* This sub-protocol is used to complete the contract if it is clear that it must be completed. One signatory, say A, sends its message m''_{4A} to the third party. The third party then either re-sends a previous decision if it was m'_5 or m''_5 (but not m_5) or else



Signatory A
Signatory B
if wakeup: "resolve$_1$".
m_{1A}
m_{1B}
if wakeup: "resolve$_1$".
if $C_A \neq C_B$ "resolve$_1$".
if m_5: rejected.
if $C_A \neq C_B$ "resolve$_1$".
if m_5: rejected.
if wakeup: "resolve$_1$".
m_{2A}
m_{2B}
if wakeup: "resolve$_1$".
if wakeup: "resolve$_2$".
m_{3A}
m_{3B}
if wakeup: "resolve$_2$".
signed
signed

Figure 3.10: Optimistic Behavior of Scheme 3.3.

produces an affidavit and sends it as $m''_5 := \text{sign}_\mathsf{T}(m''_{4A})$ to A who outputs (signed, C, tid). This recovery by A overrides the effects of a previous abort message m_{4B} sent by an incorrect player B, i.e., even if the third party sent m_5 after receiving m_{4B}, a correct player A may later ask the third party to over-rule this decision by sending m''_{4A}.

Verification of a Contract (Protocol "show"*):* After the input (show, tid), a signatory, say A, looks up (m_{3A}, m_{3B}), m'_5, or m''_5 and sends it to the verifier V. The verifier verifies the messages. If these checks fail, it outputs (rejected, tid) and else (signed, A, B, C, tid).

◇

Lemma 3.5 (Security of Scheme 3.3)
Scheme 3.3 is a secure asynchronous contract signing scheme which is optimistic on agreement. □

Proof. Scheme 3.3 adheres to Definition 3.2 by construction. We now show that it also fulfills the requirements stated in Definitions 3.3 and 3.4.

Correct Execution: If both signatories A and B start with inputs (sign, B, C, tid) and (sign, A, C, tid) and do not input wakeup then both will eventually receive all messages and will output (signed, tid). If they disagree, both will abort by sending m_4 and will finally output (rejected, tid).

Unforgeability of Contracts: Assume that a correct verifier outputs (signed, A, B, C, tid). This means that he received at least messages m_{1A}, m_{1B} (maybe included in m'_5 or m''_5) containing identical contracts which are signed by A and B, respectively. Thus, the correct parties have input (sign, B, C, tid) and (sign, A, C, tid) since otherwise they would not have sent m_{1A} and m_{1B} at all.

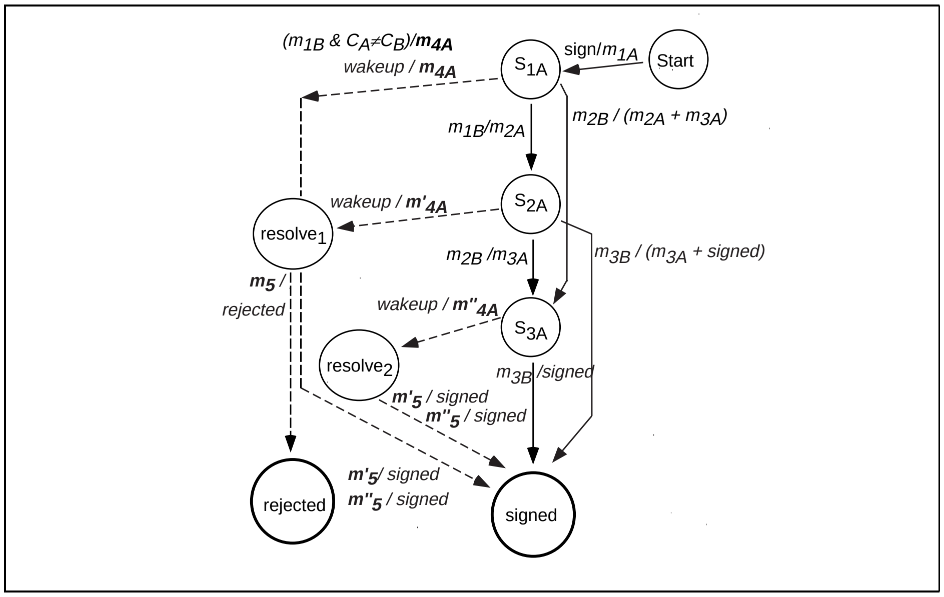


Figure 3.11: Signatory, e.g., A, of Scheme 3.3 (states S_{2A} or S_{3A} may be bypassed if messages are reordered).

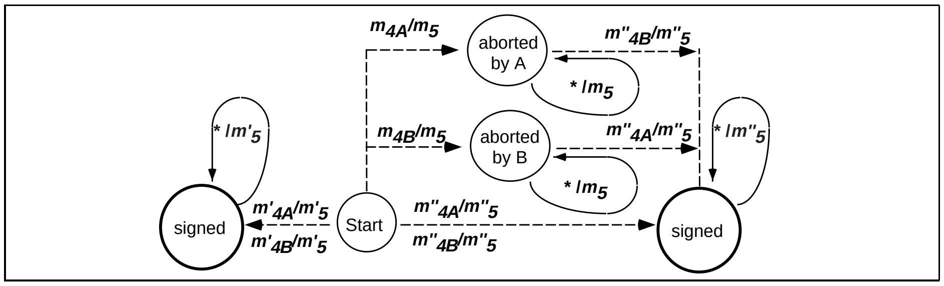


Figure 3.12: Third Party T of Scheme 3.3.

Verifiability of Valid Contracts: A signatory, say A, only outputs (signed, tid) after receiving m_{3B} or m'_5 or m''_5 containing identical contracts in messages m_{1A} and m_{1B}. Thus, it is able to convince a verifier in all cases.

No Surprises with Invalid Contracts: Let us assume that (rejected, tid) was output by a correct signatory, say A, after receiving m_5 and a correct verifier invoked by B outputs (signed, tid). Then either (m_{3A}, m_{3B}), m'_5 or m''_5 must be known by B. Let us first assume that (m_{3A}, m_{3B}) was shown to the verifier; then A sent both m_{3A} and m_{4A} or m'_{4A}, i.e., A was incorrect. Let us now assume that m'_5 was shown to the verifier; then T sent both m_5 and m'_5, i.e., the third party was incorrect. Let us finally assume that m''_5 was shown to the verifier. Since m_5 as well as m''_5 were produced by the third party, the machine T was in one of the aborted states and thus A must have sent either m_{4A} or m''_{4A}. Since A received m_5, it did not send m''_{4A}. Together this implies that A sent m_{4A}. This contradicts the assumption that m''_5 was shown to the verifier, since a correct A which sends m_{4A} does not send m_{2A} which is part of m''_5.

Termination on Asynchronous Network: If the user inputs wakeup, one of the "resolve"-protocols is started. In this protocol, the other signatory is not involved anymore. Since the third party is assumed to be correct, it will answer. Thus, the "resolve"-protocol terminates with a definitive answer after time 2, i.e., a fixed time after the input of wakeup.

Limited Trust in T*:* Even if T is incorrect, it cannot forge any signature of the two signatories. Therefore, unforgeability still holds.

Verification is independent of T. Therefore, verifiability even holds if T is incorrect.

Optimistic on Agreement: If two correct signatories do not input wakeup and input identical contracts, they both receive the outputs (signed, tid) from the "sign"-protocol after time 3 without contacting the third party. In this case, the messages m_{iA}, m_{iB} (i=1, 2, 3) are sent.

On disagreement, the messages $m_{1A}, m_{1B}, m_{4A}, m_{4B}, m_5, m_5$ are sent in time 4.

■

We now prove in Theorem 3.4 that asynchronous contract signing with only 3 messages is impossible. Then we prove the optimality of Scheme 3.3 in Theorem 3.5.

Theorem 3.4 (Message-optimality of [AsSW 98])
There exists no asynchronous optimistic contract signing scheme with a "sign"*-protocol with less than four messages in case of agreement.* □

Proof. Let us assume that A sends two messages, say m_1 and m_3, in the optimistic phase whereas B sends only one message, say m_2. Then (m_1, m_2, m_3) must be sufficient for both parties to convince the verifier. If A sends m_1 and m_3 without having received m_2, B can convince a verifier without sending m_2 to A. For "no surprises", A is required to be able to recover to signed given m_1, m_3 and without contacting B (Lemma 3.2) which contradicts the unforgeability requirement. Thus, m_3 is sent after m_2 has been received. If we now assume that B sends m_2 before receiving m_1, A could convince a verifier without sending any message to B and B would be required to be able to recover to signed without contacting A (Lemma 3.2) which again contradicts the unforgeability requirement.

Therefore, the messages are sent in the order m_1, m_2, m_3 (similar to Scheme 3.1 depicted in Figure 3.3). Since the protocol is optimistic, at least (m_1, m_2) shown by A and (m_1, m_2, m_3) shown by B are sufficient to convince the verifier. Now consider the exceptions: If B does not receive m_3 and T did not decide for this tid before, the third party has to decide locally on the output signed for B (Lemma 3.2 and termination of B) since A may have obtained a valid contract (m_1, m_2). Thus B may obtain a valid contract from the third party even if A only sent m_1. Therefore, A must be able to start recovery with the third party after sending m_1, too. In this case and if the request from A if the first for this tid, the third party is required to decide locally whether the contract is valid or not given only m_1 from A. For unforgeability for B, it has to decide on rejected based on m_1 only. If B now asks for recovery with m_1 and m_2, T has to decide locally on a decision for B (remembering its first decision). If T decides on rejected, a correct player B may be surprised since A, which already obtained rejected, may have obtained a valid contract and dishonestly recovered with m_1 only. If T decides on signed since a dishonest A may have started recovery after receiving valid contract (m_1, m_2, m_3), the "no surprises" requirement for a correct A would not be fulfilled. ■

This enables us to prove the optimality of Scheme 3.3:

Theorem 3.5 (Optimality of Scheme 3.3)
There exists no asynchronous optimistic contract signing scheme with a "sign"-*protocol in less than time 3 in case of agreement and a protocol in time 3 needs at least 6 messages.* □

Proof. If we assume that a 2-time 4-message optimistic "sign"-protocol exists, then this can be used to construct a 3-time 3-message protocol: Since the two-party signing protocol has 4 messages labeled with two subsequent times, two messages (m_{1A}, m_{1B}) are labeled with time 1 and two messages (m_{2A}, m_{2B}) are labeled with time 2, where messages labeled with the same time are independent of each other. Therefore, one player, say B, can send m_{1B} together with m_{2B}, and m_{2A} can be sent after receiving these two messages. The result is a three-message protocol with the messages $m'_{1A} := m_{1A}$, $m'_{2B} := (m_{1B}, m_{2B})$, and $m'_{3A} := m_{2A}$, which does not exist according to Theorem 3.4.

If we assume that a 5-message protocol in time 3 exists, we can construct an equivalent protocol with 3 messages in time 3 by shoving a message up or down (see Figure 3.2): If 5 messages are sent in time 3, there exists a time t for which only one message m_A sent by one signatory, say A, exists. Furthermore, two messages m'_A and m_B are labeled with a time t' which is either $t+1$ or $t-1$. If $t' = t+1$ then the messages m_A and m'_A can be sent together at time t. This is possible since A does not receive a message at time t, which guarantees that the contents of m'_A have already been fixed when m_A was sent. For B, receiving m'_A earlier must not make a difference since the network may have reordered the messages anyhow. If $t' = t-1$ then the messages m'_A and m_A can also be sent together at time t. This is possible since B does not send a message at time t, which implies that m'_A is not needed by B to compute a message. This construction enables us to change two subsequent times with two and one messages into two subsequent times with one message each. Two applications of this construction result in the desired 3-message protocol in time 3 which contradicts Theorem 3.4. ■

3.5 Optimistic on Agreement with Three-party Verification

We now describe optimistic contract signing schemes which are optimistic on agreement but require the third party to participate in the verification of a contract. For the synchronous case with three-party verification, Scheme 3.1 is message optimal whereas Scheme 3.2 is time optimal. Thus, we only consider asynchronous networks in the sequel.

3.5.1 A Message-optimal Asynchronous Scheme

We now describe an asynchronous version of the optimistic Scheme 3.1. This scheme can only be made asynchronous by including the third party into the verification of a contract. The individual machines of the players are depicted in Figures 3.14, 3.15, and 3.16.

Scheme 3.4 (Message-optimal Asynchronous)
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "sign"*; Figure 3.13):* On input (sign, B, C_A, tid), the signatory A initiates the protocol by sending the signed message $m_1 := \text{sign}_{\mathsf{A}}(\mathsf{A}, \mathsf{B}, \mathsf{T}, C_A, tid)$ with contract C_A to the responding signatory B. B receives the input (sign, A, C_B, tid) and message m_1 and verifies whether the received contract text C_A is identical to the input contract text C_B. If not or if wakeup is input before m_1 is received, B sends $m_2' := \text{sign}_{\mathsf{B}}(\mathsf{abort}, tid)$ and outputs (rejected, tid). Else, it signs the received message and sends it as $m_2 := \text{sign}_{\mathsf{B}}(m_1)$ to A. If it received m_2, A signs the received message again, sends it as $m_3 := \text{sign}_{\mathsf{A}}(m_2)$ back and outputs (signed, tid). On receipt of message m_3, B outputs (signed, tid) as well. After a successful execution of this optimistic protocol, A and B store m_3 under the tid for later use in a verification protocol.

If A gets an input wakeup before receiving message m_2 or if it receives m_2', it starts "resolve$_1$" to abort the protocol. If B did not receive message m_3, it may be that A nevertheless was able to compute a valid contract m_3 after receiving m_2. Therefore B starts the "resolve$_2$"-protocol.

Recovery from Exceptions (Sub-protocol "resolve$_1$"*):* To start this recovery protocol, A sends the message $m_{4A} := \text{sign}_{\mathsf{A}}(\mathsf{abort}, m_1)$ to T. If the third party made a decision before, it re-sends the decision. If the third party is in its start-state, it changes to the aborted state and acknowledges this to A by sending $m_5 := \text{sign}_{\mathsf{T}}(\mathsf{aborted}, m_1)$. If A receives m_5, it outputs (rejected, tid). If A receives m_5', it outputs (signed, tid).

Recovery from Exceptions (Sub-protocol "resolve$_2$"*):* Machine B sends $m_{4B} := \text{sign}_{\mathsf{B}}(m_2)$ containing m_1 and m_2 to the third party T. If the protocol was aborted, the third party now sends m_5. Else, it sends an affidavit $m_5' := \text{sign}_{\mathsf{T}}(m_2)$ to B and changes to the signed-state. If B receives m_5', it outputs (signed, tid). If it receives m_5, it outputs (rejected, tid).

Verification of a Contract (Protocol "show"*):* On input (show, tid), signatory A looks up m_3 or m_5' and sends m_5' or $m_A := \text{sign}_{\mathsf{A}}(m_3)$ to the verifier. In the case of m_5', the verifier immediately outputs signed. Otherwise it forwards m_A to the third party. If the third party re-sends an abort message m_5, the verifier outputs (rejected, tid). If the third party answers with m_5', it outputs (signed, A, B, C, tid) (note that the third party may change state during verification).

Signatory B on the other hand looks up either m_5' or else signs $m_B := \text{sign}_{\mathsf{B}}(m_3)$ and sends it to the verifier. If the verifier receives a correct message, it outputs (signed, A, B, C, tid) and (rejected, tid), else.

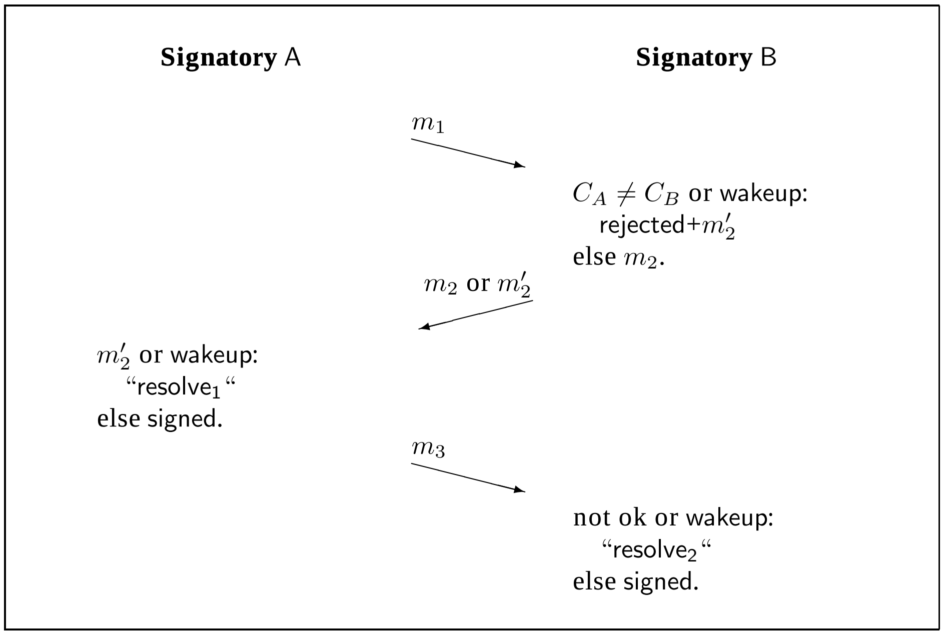


Figure 3.13: Optimistic Behavior of Scheme 3.4.

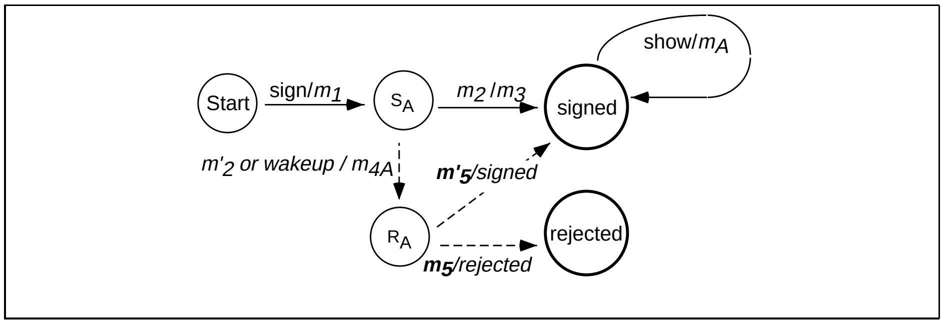


Figure 3.14: Signatory A of Scheme 3.4.

◇

Note that m_3 can be used by B to convince the verifier in any case whereas it need not be a valid contract for A if the protocol was invalidated.

Lemma 3.6 (Security of Scheme 3.4)
Scheme 3.4 is a secure asynchronous contract signing scheme with three-party verification which is optimistic on agreement. □

Proof. The scheme adheres to Definition 3.5 by construction. We now show that each of the requirements described in Definitions 3.3 and 3.4 is fulfilled:

Correct Execution: If both correct players A and B input (sign, B, C_A, tid) and (sign, A, C_B, tid) with identical tid and $C_A = C_B$, then both receive m_3 and output (signed, tid). If the contracts differ then signatory B sends m_2' and outputs (rejected, tid). On receipt of m_2' signatory A sends m_{4A} to T,

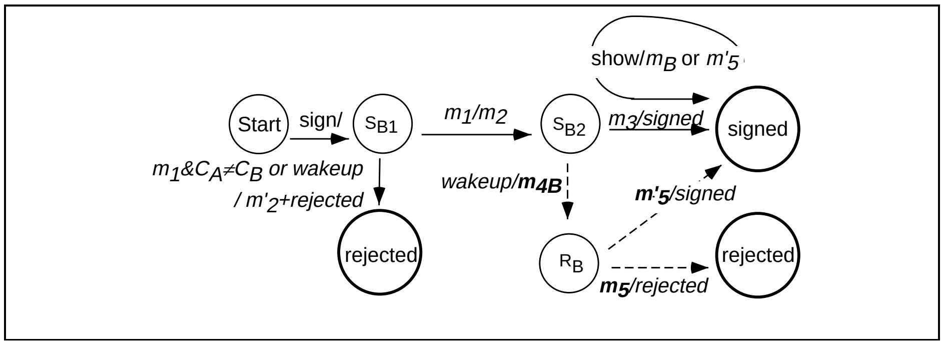


Figure 3.15: Signatory B of Scheme 3.4.

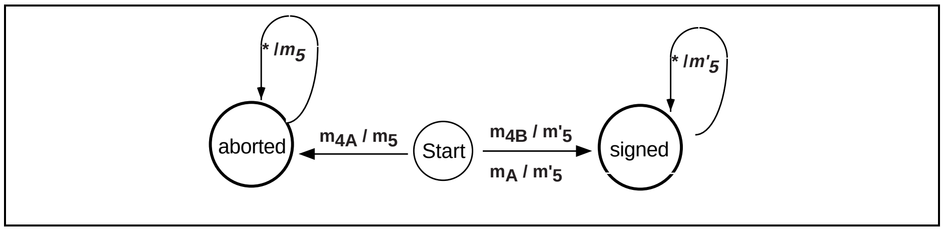


Figure 3.16: Third Party T of Scheme 3.4.

who will answer with m_5 since it did not decide for this tid before. Upon receipt of m_5, signatory A outputs (rejected, tid), too.

Unforgeability of Contracts: In order to convince a correct verifier V for a given tid, C, and partner, one needs correct messages m_3 or m'_5 for these parameters. Since m_3 as well as m'_5 contain signatures from both participants, a correct signatory input sign.

Verifiability of Valid Contracts: If a correct signatory A outputs (signed, tid) then it received m_2 or m'_5 from which it can compute m_A. This is a valid contract if the third party does not re-send m_5. If m'_5 has been received and the third party re-sends m_5, the third party is incorrect. If m_2 has been received and a correct third party re-sends m_5, A is incorrect since it sent m_{4A} while receiving m_2.

If B output (signed, tid), it received m_3 or m'_5 which will can be used to convince the verifier in any case.

No Surprises with Invalid Contracts: Let us first assume that a correct signatory A returned rejected on input (sign, B, C, tid) whereas B is able to convince the verifier. This requires that B knows m_3 or m'_5 for the given tid and C because it cannot construct m_A in order to pretend to be A. Since A returned rejected, it executed "resolve$_1$" and received m_5. Thus, B did not receive m'_5 from the correct T. If B received m_3, A was incorrect since it did not ignore m_2 after sending m_{4A}.

Without m_2, machine A cannot obtain an output signed at the verifier. If B would output rejected after sending m_2 and A would be able to convince a verifier, then the third party sent m'_5 during "show" and m_5 during "resolve$_2$", i.e., the third party would be incorrect.

Termination on Asynchronous Network: The scheme requires at most time 2 after an input wakeup.

Limited Trust in T*:* Unforgeability also holds if the third party misbehaves since any valid contract includes signatures from both signatories.

Optimistic on Agreement on Asynchronous Network: If two correct signatories input (sign, C, tid), signatory A outputs (signed, C, tid) after time 2 and player B after time 3. Thus, on agreement, the scheme sends m_1, m_2, m_3 and requires time 3. On disagreement, the scheme sends m_1, m_2', m_{4A}, m_5 and requires time 4.

■

Theorem 3.6 (Optimality of Scheme 3.4)
There exists no asynchronous optimistic contract signing scheme with three-party verification with a "sign"*-protocol with less than three messages in case of agreement, and every three-message protocol requires at least time 3.* □

Proof. If we assume that there exists a two-message asynchronous optimistic "sign"-protocol, unforgeability requires that each signatory sends one of these messages. Furthermore, each of these messages received by a correct player must be sufficient to convince the verifier since optimism requires that if this single message is received correctly and the signatories agree, this signatory outputs signed (unlike synchronous networks, it cannot wait for later revocation messages from T).

Let us assume that the signatory sending the first message receives wakeup after sending its message but before receiving the message from the peer. If this is the first request to T for this tid, this signatory is required by Lemma 3.2 to recover with the third party, and the decision must be rejected in order to guarantee unforgeability. This contradicts the "no surprises" requirement since the other signatory outputs signed.

Let us assume that there exists an optimistic "sign"-protocol with three messages in two rounds. Then this can be used to construct a two-message protocol by shoving messages as in the proof of Theorem 3.5 and Figure 3.2, but such a two-message protocol does not exist. ■

3.5.2 A Time-optimal Asynchronous Scheme

We now describe a time-optimal Scheme 3.5 for asynchronous networks and three-party verification and prove its security in Lemma 3.7. It requires only two rounds but four messages. Since any three-message "sign"-protocol needs at least three rounds (Theorem 3.5), there exists no one-round protocol at all and no 2-round protocol with only three messages. So the scheme described is optimal with respect to rounds and given the limitation to two rounds also with respect to the number of messages. The optimistic behavior of the scheme is depicted in Figure 3.17. The players are depicted in Figures 3.18 and 3.19.

Scheme 3.5 (Time-optimal Asynchronous)
This scheme consists of the triple (A, B, V) and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "sign"*; Figure 3.17):* On input (sign, B, C_A, tid) a signatory, say A, sends message $m_{1A} := \text{sign}_\mathsf{A}(\mathsf{A}, \mathsf{B}, \mathsf{T}, C_A, tid)$ with the protocol parameters in the first round. If it receives an input (wakeup, tid) or an incorrect message m_{1B}, it starts the sub-protocol "abort". If a message m_{1B} with $C_A = C_B$ is received, A sends the message $m_{2A} := \text{sign}_\mathsf{A}(m_{1A}, m_{1B})$ and waits for m_{2B}. If (wakeup, tid) is input or no correct m_{2B} is received, it executes the sub-protocol "resolve". Else, if a correct m_{2B} is received, it outputs (signed, tid). If A receives m_{2B} before receiving m_{1B}, it sends m_{2A} and outputs (signed, tid).

Aborting a Protocol Run (Sub-protocol "abort"*):* This subprotocol is used to abort a run unless it was resolved before.

The signatory, say A, sends a message $m'_{3A} := \text{sign}_\mathsf{A}(m_{1A})$ to the third party. If the third party is still in state Start it sends a message $m'_{4A} := \text{sign}_\mathsf{T}(\text{aborted}, m'_{3A})$ and changes to the state aborted by A.

If the third party is in the state aborted by A, it re-sends m'_{4A}. If the third party is in the state aborted by B, it re-sends m'_{4B}. If the third party is in the signed-state, it re-sends m_{4A}, m_{4B}, or m_6.

If A receives m_{4B}, m_{4B}, or m_6 it outputs (signed, tid). If it receives m'_{4A} or m'_{4B} it outputs (rejected, tid).

Recovery from Exceptions (Sub-protocol "resolve"*):* This sub-protocol completes a contract if this run was not aborted before.

The signatory, say A, sends a message $m_{3A} := \text{sign}_\mathsf{A}(m_{2A})$ to the third party. If T is in the Start state, T sends a message $m_{4A} := \text{sign}_\mathsf{T}(\text{signed}, m_{3A})$ to A.

If the third party is in the state aborted by A, it re-sends m'_{4A}. If the third party is in the state aborted by B, it re-sends m'_{4B}. If the third party is in the signed-state, it re-sends m_{4A}, m_{4B}, or m_6.

If A receives m_{4B}, m_{4B}, or m_6 it outputs (signed, tid). If it receives m'_{4A} or m'_{4B} it outputs (rejected, tid).

Verification of a Contract (Protocol "show"*):* On input (show, tid), a signatory, say A, sends either m_{4A}, m_{4B}, m_6, or $m_{5A} := \text{sign}_\mathsf{A}(m_{2A}, m_{2B})$ to the verifier.

If the verifier receives m_{4A}, m_{4B}, or m_6, it outputs (signed, A, B, C, tid).

Else, it forwards m_{5A} to the third party. If the third party is in the Start-state, it answers with $m_6 := \text{sign}_\mathsf{T}(m_{5A})$ and changes to the signed-state. Else, it re-sends an earlier decision. If this earlier decision is not m'_{4A}, the verifier outputs (signed, A, B, C, tid). Else, it outputs (rejected, tid).

◇

Lemma 3.7 (Security of Scheme 3.5)
Scheme 3.5 is a secure asynchronous contract signing scheme with three-party verification, which is optimistic on agreement. □

Proof. The scheme adheres to Definition 3.5 by construction. We now show that it fulfills the requirements stated in Definitions 3.3 and 3.4:

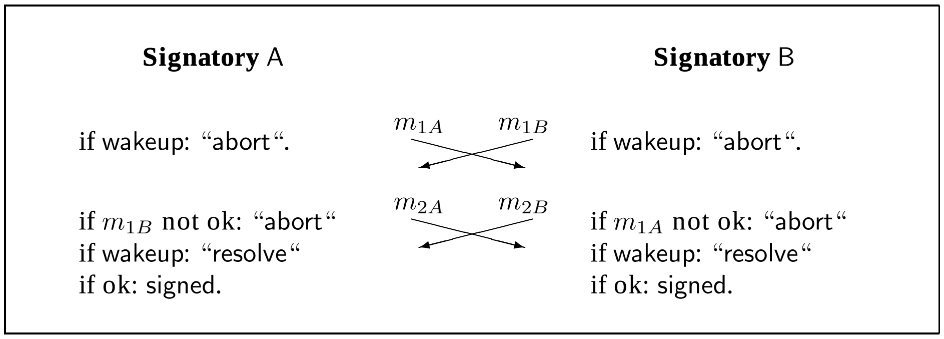
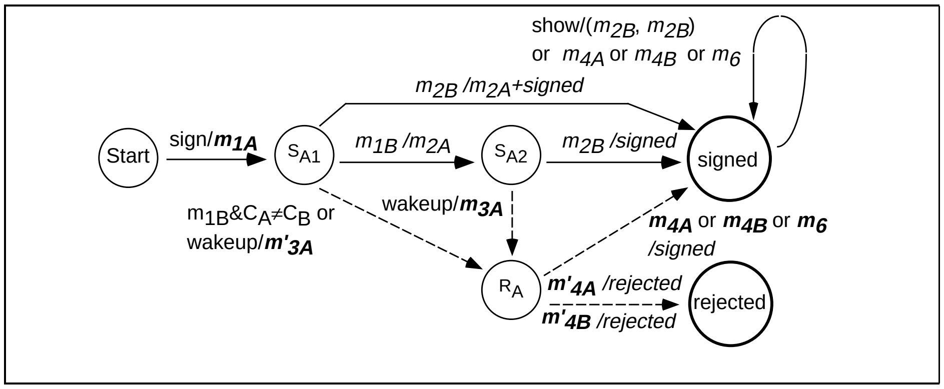
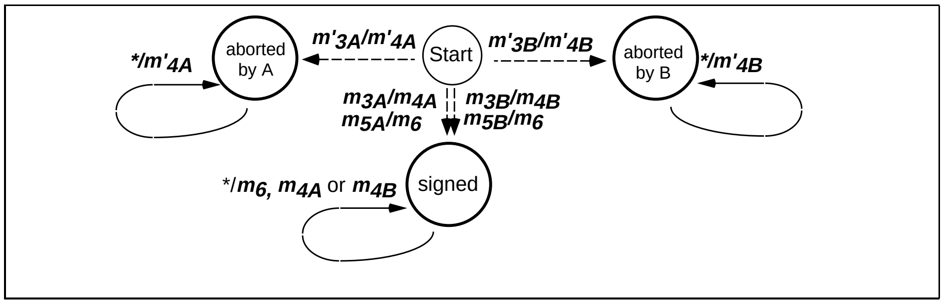


Figure 3.17: Optimistic Behavior of Scheme 3.5.



show/(m_{2B}, m_{2B})
or m_{4A} or m_{4B} or m_6
m_{2B} /m_{2A}+signed
Start
sign/$\boldsymbol{m_{1A}}$
S_{A1}
m_{1B} /m_{2A}
S_{A2}
m_{2B} /signed
signed
m_{1B}&C_A≠C_B or
wakeup/$\boldsymbol{m'_{3A}}$
wakeup/$\boldsymbol{m_{3A}}$
$\boldsymbol{m_{4A}}$ or $\boldsymbol{m_{4B}}$ or $\boldsymbol{m_6}$
/signed
R_A
$\boldsymbol{m'_{4A}}$ /rejected
$\boldsymbol{m'_{4B}}$ /rejected
rejected

Figure 3.18: Signatory, e.g., A, of Scheme 3.5.



*/$\boldsymbol{m'_{4A}}$
aborted by A
$\boldsymbol{m'_{3A}/m'_{4A}}$
Start
$\boldsymbol{m'_{3B}/m'_{4B}}$
aborted by B
*/$\boldsymbol{m'_{4B}}$
$\boldsymbol{m_{3A}/m_{4A}}$
$\boldsymbol{m_{5A}/m_6}$
$\boldsymbol{m_{3B}/m_{4B}}$
$\boldsymbol{m_{5B}/m_6}$
*/$\boldsymbol{m_6}$, $\boldsymbol{m_{4A}}$ or $\boldsymbol{m_{4B}}$
signed

Figure 3.19: Third Party T of Scheme 3.5.

Correct Execution: If both players behave correctly and input identical contracts, each signatory, say A, receives m_{1B} and m_{2B}. Thus, the protocol outputs (signed, tid) on both machines. If the signatories disagree, both execute the "abort" protocol and will output (rejected, tid).

Unforgeability of Contracts: In order to convince a correct verifier, a signatory, say A, needs (m_{2A}, m_{2B}), m_{4A}, or m_{4B}. Since all these messages contain signatures from both signatories, a correct signatory input (sign, A, C, tid) or (sign, B, C, tid), respectively.

Verifiability of Valid Contracts: A signatory, say A, only outputs (signed, tid) after receiving (m_{2A}, m_{2B}), m_{4A}, m_{4B}, or m_6. If it receives m_{4A}, m_{4B}, or m_6 the verifier decides on signed. If A received (m_{2A}, m_{2B}) it can send m_{5A} to the verifier who decides on signed if the third party does not send m'_{4A}. If we now assume that the third party sends m'_{4A}, then T received m'_{3A} while A output signed on receipt of m_{2B}, i.e., A was incorrect.

No Surprises with Invalid Contracts: If a signatory, say A, outputs rejected, it received m'_{4A} or m'_{4B}.

If the verifier would output signed for B upon receipt of m_{4A}, m_{4B}, or m_6, then T would be incorrect since it must not sign these messages together with m'_{4A} or m'_{4B}.

Let us now assume that the correct verifier would output signed upon receipt of m_{5B}, then T sent m_6 instead of its earlier decision m'_{4A} or m'_{4B}, i.e., T would be incorrect.

Termination on Synchronous Network: The protocol terminates in time 2 and 4 messages in case of agreement. On disagreement, it requires 6 messages in time 4.

Limited Trust in T*:* Even if T is incorrect, it cannot forge any signature of the two signatories. Therefore, unforgeability still holds.

Optimistic on Agreement: If two correct signatories input (sign, B, C, tid) and (sign, A, C, tid) and do not input wakeup, a signatory, say A, outputs (signed, tid) after receiving m_{1B}, m_{2B}. In this case, the messages $m_{1A}, m_{1B}, m_{2A}, m_{2B}$ are sent in time 2.

In case of disagreement, the messages $m_{1A}, m_{1B}, m'_{3A}, m'_{3B}, m'_{4A}, m'_{4A}$ are sent and the scheme requires time 4.

■

3.6 Optimistic on Disagreement

We now describe provably time- or message-optimal protocols that are optimistic on disagreement. For this case, we optimize the maximum number of messages and time needed if both signatories are correct. This includes the cases that they agree or disagree on the contract.

We do this by showing how to adapt any optimistic contract signing protocol to be optimistic on disagreement. If the contract signing protocol has only one message at time one, this requires one additional message and one additional round. If the contract signing protocol has two messages at time 1, it requires two additional messages and one additional round.

In order to prove the optimality of the resulting protocols, we show that optimism on disagreement cannot be achieved in the time / messages needed by the optimal protocols presented in Sections 3.4 and 3.5.

3.6.1 Adapting Optimistic Contract Signing

Theorem 3.7
Any optimistic contract signing scheme can be adapted to be optimistic on disagreement with two additional messages in time 1.

If the protocol has only one message at time 1, this simulation requires only one additional message in time 1. □

We prove this by two reductions:

Scheme 3.6
Let $CS = (\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T be a contract signing scheme that is optimistic on agreement.

This scheme is adapted as follows: On input $(\mathsf{sign}, \mathsf{B}, C_A, tid)$, each signatory, say A, sends a message $m'_{1A} := \mathrm{sign}_{\mathsf{A}}(\mathsf{B}, C_A, tid)$ to B. If machine A receives a correct message m'_{1B} with matching parameters, it executes the underlying protocol. Else, it outputs $(\mathsf{rejected}, tid)$. ◇

Lemma 3.8
Scheme 3.6 is a secure contract signing scheme that is optimistic on disagreement. □

Proof. We now show that each requirement holds that is not directly implied by the security of the underlying protocol:

Correct Execution: If both signatories A and B start with inputs $(\mathsf{sign}, \mathsf{B}, C, tid)$ and $(\mathsf{sign}, \mathsf{A}, C, tid)$ and do not input wakeup then they send m'_{1A} and m'_{1B} and then start the underlying protocol with matching parameters. Correct execution then follows from correct execution of the underlying protocol.

If both input different contracts, then m'_{1A} and m'_{1B} are sent and both output $(\mathsf{rejected}, tid)$.

No Surprises with Invalid Contracts: If the underlying protocol output $(\mathsf{rejected}, tid)$, "no surprises" follows from "no surprises" of the underlying protocol. If "no surprises" is output before a party starts the underlying protocol, then "no surprises" follows from the unforgeability for the underlying protocol.

Termination on Asynchronous Network: The input (wakeup, tid) produces an output $(\mathsf{rejected}, tid)$ immediately if the underlying scheme has not been started. Else, termination follows from the termination of the underlying protocol.

Optimistic on Disagreement: If two correct parties input the same contract, optimism on disagreement follows from optimism on agreement of the underlying scheme. If the parties input different contracts, A outputs (rejected, tid) after sending m'_{1A} and receiving m'_{1B} while A outputs (rejected, tid) after sending m'_{1B} and receiving m'_{1A}.

■

Scheme 3.7
Let $CS = (\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T be an optimistic contract signing scheme with only one message m_1 at time 1 of the "sign"-protocol. We assume w.l.o.g. that m_1 is sent from B to A. The scheme is adapted as follows:

On input (sign, B, C, tid) in Round 1, A sends a message $m'_1 := \text{sign}_{\mathsf{A}}(\mathsf{B}, C, tid)$ to B. If machine B disagrees with the contract or if wakeup was input, B sends $m'_2 := \text{sign}_{\mathsf{B}}(\mathsf{abort}, tid)$ and outputs (rejected, tid). If it agrees, it starts the given contract signing protocol by sending m_1 of the original protocol. If A receives m'_2 or an input (wakeup, tid) before m_1, it outputs (rejected, tid). Else, it proceeds with the given contract signing protocol using m_1[22]. Subsequent inputs of wakeup are handled by the underlying protocol. ◇

Lemma 3.9
Scheme 3.7 is a secure contract signing scheme that is optimistic on disagreement. □

Proof. We now show that each requirement holds that is not directly implied by the security of the underlying protocol:

Correct Execution: If both players input identical contracts then m'_1 is sent from A to B who answers with m_1 of the original protocol. Since this protocol is then executed without changes, correct execution for agreement follows from correct execution of the underlying protocol.

If the input contracts differ, then B sends m'_2 and both output (rejected, tid) without starting the underlying protocol.

No Surprises with Invalid Contracts: If the underlying protocol output (rejected, tid), "no surprises" follows from "no surprises" of the underlying protocol. If (rejected, tid) is output before B sends m_1 or A receives it, then "no surprises" follows from the unforgeability for the underlying protocol.

Optimistic on Disagreement: If two correct signatories input (sign, B, C, tid) and (sign, A, C, tid) and do not input wakeup, signatory A sends m'_1 and B starts the protocol. In this case, optimism follows from the fact that the underlying protocol is optimistic on agreement. If both input different contracts, m'_1 and m'_2 are sent and both output (rejected, tid).

■

[22]I.e., if no m_1 or m'_2 is received in the synchronous case, this is handled like a missing m_1.

3.6.2 Optimality Proofs

Any time- or message-optimal contract signing scheme described in Section 3.4 can be adapted to be optimistic on disagreement. We now show that Scheme 3.6 applied to time-optimal schemes results in time-optimal schemes that are optimistic on disagreement. Furthermore, using message-optimal schemes, Scheme 3.7 results in message-optimal schemes that are optimistic on disagreement.

Theorem 3.8
There exists no synchronous optimistic contract signing protocol with three-party verification which is optimistic on disagreement with less than 4 messages in all optimistic cases. □

Proof. Let us assume that such a scheme exists and that the messages are sent in a row m_1, m_2, m_3 (if this is not the case, delaying some messages helps) and that machine A sends the first message.

In case of agreement m_1, m_2, m_3 are sent, where m_1, m_2 is sufficient for A to convince a verifier that a contract C is valid and m_1, m_2, m_3 is sufficient for B.

If we now assume that B did not obtain m_3 in a run with identical contracts, the third party may either revoke the contract m_1, m_2 for A or else decide on signed for B.

If it revokes the contract and A is correct then an incorrect B is able to invalidate the contract of a correct A. This contradicts the verifiability requirement of A (a correct A cannot participate in the recovery protocol since it terminated after time 3 where it sent its last message.)

Therefore, T and B must recover with (m_1, m_2) to signed without contacting A.

Let w.l.o.g. B be the party inputting a different contract $C' \neq C$ in case of disagreement. In this case m_1, m_2', m_3' are sent (primed messages contain the contract $C' \neq C$; some messages may be empty) and m_1, m_2' as well as m_1, m_2', m_3' must lead to an output rejected without contacting T. In this case, however, a cheating B may send m_2' and A terminates with an output rejected. Still, B could recover using a consistent m_2, which contradicts the "no surprises" requirement of A. ■

Theorem 3.9
There exists no synchronous contract signing protocol with three-party verification which is optimistic on disagreement that needs time 2 in all optimistic cases and every scheme in time 3 requires at least 6 messages. □

Proof. If there were a scheme in time 3 with at most 5 messages, the messages could be rearranged into a 3 message protocol in time 3, which does not exist (recall Figure 3.2). ■

Theorem 3.10
There exists no asynchronous optimistic contract signing protocol which is optimistic on disagreement with less than 5 messages in all optimistic cases. □

Proof. Let us assume that there were a scheme with 4 messages. Let us assume that these messages are sent in a row and that A sends the initial message (if not, delaying some messages helps).

In case of agreement between correct signatories a prefix of messages (m_1, m_2, m_3, m_4) (some may be empty) is sent where (m_1, m_3) must lead to a valid contract for B whereas (m_2, m_4) must lead to a valid contract for A and, for unforgeability, at least m_1 and m_2 are non-empty (recall, we ruled out three-party disputes). In case of disagreement between correct signatories, a prefix of messages (m_1, m_2', m_3', m_4') is sent where both (m_1, m_3') and (m_2', m_4') must not be sufficient for B and A, respectively, to convince the verifier. Again, at least m_1 and m_2' are sent in order to enable A to detect disagreement (the scheme was assumed to be optimistic for disagreement and detecting $\neg m_2$ is impossible on asynchronous networks).

Let us assume that B invokes T with (m_1, m_2) for the first time for this *tid* and T decides rejected for B. Then a correct B may have sent m_2 and a correct A may have answered with m_3. In this case, when A later invokes T also, the third party is required to decide on signed for A in spite of its first decision since an incorrect B may nevertheless receive m_3 and thus obtain a contract. This decision, however, would contradict the "no surprises" requirement of a correct B. Therefore, the third party is required to decide on signed for B for the messages (m_1, m_2).

However, as a consequence, a cheating B can recover to signed using (m_1, m_2) after participating in an optimistic execution where it sent m_2' and m_4', which contradicts "no surprises" for A. ■

Theorem 3.11
There exists no asynchronous contract signing protocol which is optimistic on disagreement which needs time 3 in all optimistic cases, and every scheme in time 4 requires at least 8 messages. □

Proof. If there is a scheme in time 3 or a scheme in time 4 with less than 8 messages, this scheme can be used to construct a 4-message scheme in time 4 by shoving messages as in the proof of Theorem 3.5 and Figure 3.2, which contradicts Theorem 3.10. ■

Theorem 3.12
There exists no asynchronous optimistic contract signing protocol with three-party verification, which is optimistic on disagreement with less than 4 messages in all optimistic cases. □

Proof. Let us assume that such a scheme exists and that the messages are sent in a row m_1, m_2, m_3 and that machine A sends the first message (if not delaying some messages helps).

In case of agreement m_1, m_2, m_3 are sent and m_1, m_2 is sufficient to compute a contract for A and m_1, m_2, m_3 is sufficient for B.

If we now assume that B obtained wakeup before receiving m_3 in a run with identical contracts, the third party is required to either revoke the contract for this *tid* or else B must be able to obtain a contract.

If the third party revokes the contract without contacting A (Lemma 3.2), this contradicts “verifiability” of A since A may have output (signed, tid) in the meantime.

If the third party decides on signed for B, B may send m_2' and A terminates with an output rejected. Still, B could recover using a consistent m_2, which contradicts the “no surprises” requirement. ■

Theorem 3.13
There exists no asynchronous contract signing protocol with three-party verification which is optimistic on disagreement in time 2 in all optimistic cases. Every scheme in time 3 requires at least 6 messages. □

Proof. If there were a scheme in time 2 or in time 3 with 5 messages, the messages could be reordered resulting in a scheme with 3 messages in a row. Such a scheme does not exist. ■

Chapter 4

Optimal Efficiency of Optimistic Certified Mail

4.1 Introduction

Certified mail is the fair exchange of a message for a receipt [Blu2 83]. A certified mail scheme contains at least three machines being able to execute two protocols: A sender sends a message to a recipient using the "send"-protocol which produces a receipt if and only if the recipient obtains the message. This receipt can later be used in a receipt verification protocol "show" to convince arbitrary verifiers, such as a court, that the mail was received by the intended recipient.

For issuing receipts, the obvious problem is fairness, i.e., if the sender reveals the message first, the recipient may refuse to sign a receipt. If the receipt is sent first, the sender may refuse to send the message.

Therefore, in all practical schemes, certified mail involves an additional third party. This party is at least to some extent trusted to behave correctly, thus playing the role of a trusted post office in paper-based certified mail.

In order to minimize this involvement while guaranteeing fairness, the notion of "optimistic certified mail" has been introduced [AsSW 97].

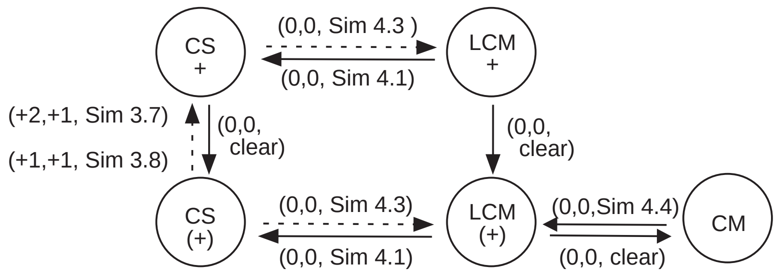
The basic idea of optimistic certified mail is that the third party is not needed in the fault-less case. Only in case of errors, the third party is involved to restore fairness.

4.1.1 Results of this Chapter

This chapter proves tight lower bounds on three kinds of optimistic certified mail for a variety of models. The main service of certified mail is to produce a non-repudiable receipt fixing the sent message. This receipt is produced if and only if the message is sent.

Our three models of certified mail differ in the preconditions for sending a message:

- Labeled optimistic certified mail sends a message and issues a receipt if and only if the sender and the recipient agree on the subject of the message. The subject as well as the message are fixed in the receipt.



Legend:

CM	"Traditional" Certified Mail.
LCM	Labeled Certified Mail.
CS	Contract Signing.
+	stands for optimistic on disagreement.
(+)	stands for optimistic on agreement.
$(m, t, SimX)$	Arrows are labeled with the messages and time needed for the simulation described in Scheme X. Dashed arrows are only applicable to some schemes.

Figure 4.1: Relations between Contract Signing and Certified Mail.

- Unlabeled optimistic certified mail (this is the traditional model) only sends a message and issues a receipt if the recipient is willing to participate in a particular protocol run.
- Simple optimistic certified mail sends the mail and issues the receipt in any case.

At first we analyze labeled certified mail for asynchronous and synchronous networks, and optimism on agreement and disagreement. Unlike contract signing, we do not allow three-party disputes.

In Section 4.3, we show that any labeled certified mail scheme can be used to provide contract signing without additional messages or time. Together with the results on the optimal efficiency of optimistic contract signing from Chapter 3, this results in lower bounds for labeled certified mail. In Section 4.4, we show that all our message-optimal contract signing schemes can be used to provide labeled certified mail without additional messages or time. The message-optimality of the resulting scheme then follows from the simulation described in Section 4.3.

In Section 4.5 we show how to "label" unlabeled optimistic certified mail. This proves that the traditional notion of optimistic certified mail cannot be more efficient than our model of labeled certified mail.

In Section 4.6, we describe the new notion of "simple certified mail", which does not allow the recipient to choose whether to participate in individual protocol runs or not. This leads to a two-message protocol in time two which is time- as well as message-optimal.

An overview of our reductions is given in Figure 4.1. All optimality results are summarized in Table 4.1.

Schemes	Model			Efficiency	Proof in
	Op	C	TP	“(+)”	
Schemes 3.1+3.7+4.3	+	s	2v	$\underline{4m}$ 4t	Th. 4.2
[AsSW 98]+Scheme 3.7+4.3	+	a	2v	$\underline{5m}$ 5t	Th. 4.2
Scheme 3.1+4.3	(+)	s	2v	$\underline{3m}$ 3t	Th. 4.2
[AsSW 98]	(+)	a	2v	$\underline{4m}$ 4t	Th. 4.2
Scheme 4.7	s	a	2v	$\underline{2m}$ $\underline{2t}$	Th. 4.4

Legend:

Op	“+” stands for optimistic on disagreement, “(+)” stands for optimistic on agreement, “s” stands for simple certified mail.
C	Communication Model: “s” for synchronous, “a” for asynchronous.
TP	Properties of the third party: “2v” means that the third party is not allowed to participate in the verification of a receipt.
m	Number of messages in the optimistic case (underlined figures are provably optimal).
t	Time in the optimistic case.

Table 4.1: Optimal Schemes for Optimistic Labeled Certified Mail.

4.1.2 Notations and Assumptions

In addition to the assumptions introduced in Section 3.1.2, we assume that a public-key encryption scheme (E_T, D_T) for the third party T is given. We assume that $c = E_T(r; m)$ encrypts m randomized using a random number r. Besides the usual secrecy of the encrypted plaintext, we require that it is hard to choose two pairs (r, m) and (r', m') with $m \neq m'$ such that $E_T(r, m) = E_T(r', m')$[1].

Such a scheme can be built by encrypting $(r, m, \mathcal{H}(r, m))$ using RSA [RSA 78] where the third party proves the fact that its keys where chosen properly; then any ciphertext has a unique decryption. The resulting scheme is as secure as RSA itself.

The terms “any model” or “all models” only refers to the contract signing models that are considered in this chapter, i.e., it refers to all models without three-party verification as defined in Section 3.2 as well as their counterparts for labeled certified mail as defined in Section 4.2.

For proving generic simulations, we prove the simulation protocols using the requirements of the underlying service. Whenever used, these requirements are listed in square brackets. Usually, our simulations can be used on synchronous and asynchronous networks. Therefore, we assume that synchronous protocols ignore the input of wakeup that may sometimes be caused by our simulations. Since synchronous termination requires termination in a fixed time, this also holds after an input wakeup, i.e., on synchronous networks, synchronous termination implies asynchronous termination.

We denote a machine Z composed from machines X and Y as $\mathsf{Z} = \langle \mathsf{X}, \mathsf{Y} \rangle$. The sub-machines are defined to be local to each other, i.e., communicate using in- and outputs to each other instead of messages.

[1]This enables us to fix a message and other parameters by including a signed ciphertext into the receipt.

We use the identifier of the first machine as the identifier of the composite machine, i.e., protocols and signatures of machine Y inside $\langle \mathsf{X}, \mathsf{Y} \rangle$ will be verified under the identifier X. We assume that messages are automatically dispatched to the appropriate sub-machine and that the identifier of X is sufficient for sending messages to the sub-machine Y as well.

Remark 4.1. The intuitive reason that all sub-machines use the name and address of the super-machine is that in practice the complete machine belongs to a person and uses the address and signature identity of its owner.

Remark 4.2. When identifying the sub-machine using the super-machine's name, we assume that the identifier used by a sub-protocol can be configured.

As our definition of contract signing does not include the name of a machine in its inputs[2], this requires that the underlying scheme is initialized using the identifier of the super-protocol. In practice, however, the contract signing schemes would be extended to include a machine's own name in all inputs, too. ◦

4.2 Definitions

We now define the new notion of labeled optimistic certified mail. In the sequel, we use the network model as defined in Section 3.2.1 and the notion of optimistic contract signing as defined in Section 3.2.2.

The traditional notion of optimistic certified mail will be defined in Section 4.5.1. Simple certified mail will be defined in Section 4.6.1.

4.2.1 Optimistic Labeled Certified Mail

Our new model of certified mail called "labeled certified mail" assumes that mails are sent and accepted under non-secret labels, i.e., that the recipient can define for which kinds of messages or which subjects a receipt should be issued.

The label enables to specify the context in which the message is sent. Consider, e.g., a delivery of a program where the recipient is only willing to receive the program if both parties agree on the licensing conditions. In this application, labeled certified mail makes sure that the program data is only sent if both agree. Furthermore, it fixes the conditions as part of the receipt. Another examples are external references to link the certified mail transmission into a larger commerce context (see Section 11.3 for more information on external references).

Definition 4.1 (Labeled Certified Mail Scheme)
A *labeled certified mail scheme* for a message space M with $|M| \geq 2$, a label space L and a set of transaction identifiers $TIDs$ is a triple $(\mathsf{S}, \mathsf{R}, \mathsf{V})$ of probabilistic interactive machines (such as probabilistic interactive Turing Machines) where V does not keep state between subsequent protocol runs[3]. The algorithm S is

[2]These protocols implicitly assume that the identifier used by each protocol machine is fixed; the reason for this simplification is that the contract signing protocols were not primarily developed for building certified mail on top of them.

[3]If the outcome of a verification may depend on the state of the verifier, a receipt can not be shown to arbitrary verifiers.

called sender, the algorithm R is called recipient, and the algorithm V is called verifier. In addition, the scheme may specify a set AM of auxiliary machines without in- or outputs. The algorithms can carry out two interactive protocols:

Sending Mail (Protocol “send“*):* The sender obtains a local input (send, R, l, m, tid), where send indicates that the “send“-protocol shall be executed, R is the identifier of the recipient, $l \in L$ is the label under which message $m \in M$ shall be sent, and $tid \in TIDs$ is the common unique and fresh transaction identifier which is used to distinguish different protocol runs. The recipient inputs (receive, S, l, tid) to signal that he is willing to receive a message under a given label l from a sender S and acknowledge its receipt. At the end, the sender either gets an output (sent, tid) or (failed, tid) whereas the recipient either gets (received, m, tid) or (failed, tid).

Verification (Protocol “show“*):* This is the receipt verification protocol between a particular verifier V and the sender S only[4]. The sender inputs (show, tid) and the verifier outputs either (received, S, R, l, m, tid) with the identities of the sender and the recipient as well as the label and the message or (failed, tid).

$\diamond$

Intuitively, the input labels are similar to the subject under which the mail is sent, i.e., the recipient fixes the subject under which a mail shall be received and a receipt shall be issued. The outputs (sent, tid) and (received, m, tid) of the “send“-protocol mean that the message has been sent and a receipt has been issued whereas the outputs failed signal failure, i.e., that the message has not been released and no valid receipt has been issued.

Definition 4.2 (Secure Labeled Certified Mail)
A labeled certified mail scheme as defined in Definition 4.1 is called secure if it fulfills the following requirements if all auxiliary parties in AM are correct:

Requirement 4.2a (Correct Execution): Consider an execution of “send“ by a correct sender S and a correct recipient R on input (send, R, l_S, m, tid) to S and (receive, S, l_R, tid) to R with a unique and fresh $tid \in TIDs$, $\mathsf{S}, \mathsf{R} \in id\_space$, $m \in M$, and the labels $l_S, l_R \in L$. If these inputs are made in the same round on synchronous networks and if wakeup is not input on asynchronous networks, the sender outputs (sent, tid) and the receiver outputs (received, m, tid) iff $l_S = l_R$.

If a correct recipient outputs (received, m, tid) on input (receive, S, l, tid) and the sender S is correct, then the sender input (send, R, l, m, tid).

Requirement 4.2b (Unforgeability of Receipts): If (receive, S, l, tid) was not input by a correct recipient R, no correct verifier V will output (received, S, R, l, m, tid) for any m.

Requirement 4.2c (Receipts are Fixed): If a correct recipient outputs (received, m, tid) on input (receive, S, l, tid) then no correct verifier will output (received, S′, R, l', m', tid) for any different S′, m' or l'.

[4]Like in contract signing, we do not consider verification including the sender as well as the recipient.

Requirement 4.2d (Verifiability of Valid Receipts): If a correct sender S outputs (sent, tid) on input (send, R, l, m, tid) and later executes “show“ on input (show, tid) with a particular correct verifier V then this verifier V will output (received, S, R, l, m, tid).

Requirement 4.2e (No Surprises for the Recipient): If a correct recipient R outputs (failed, tid) on input (receive, S, l, tid) then no correct verifier will output (received, S, R, l, m, tid) for any m.

Requirement 4.2f (Secrecy of the Message): If a correct sender outputs (failed, tid) on input (send, R, l, m, tid) then the message $m \in M$ is kept entirely secret.[5]

Furthermore, one of the following requirements must be fulfilled:

Requirement 4.2g (Termination on Synchronous Network): On input (send, R, l, m, tid), a correct sender will either output (sent, tid) or (failed, tid) after a fixed number of rounds.

On input (receive, S, l, tid), a correct recipient will output (received, m, tid) or (failed, tid) after a fixed number of rounds.

Requirement 4.2h (Termination on Asynchronous Network): On input (wakeup, tid) a correct sender will either output (sent, tid) or (failed, tid) after a fixed time.

On input (wakeup, tid) a correct recipient will either output (received, m, tid) or (failed, tid) after a fixed time.

◇

Remark 4.3. Note that wakeup can only be input to the sender and the receiver during the “send“-protocol. Termination of the “show“-protocol is implied by [R. 4.2d].

Remark 4.4. “Correct execution” does not guarantee the output of failed in case of disagreement. But from [R. 4.2g+R. 4.2h] follows that the sender outputs failed if it does not output sent whereas the recipient outputs failed if it does not output received.

Remark 4.5. Intuitively, the fresh tids are only used to associate the inputs at S and R, i.e., to determine that they belong to the same protocol run. If the players input different tids, the protocol usually behaves as if the other party is not present. The protocols guarantee termination without an input wakeup for different labels [R. 4.2a] but not for different tids. ○

An optimistic protocol includes a third party which is not involved if the parties are correct and agree:

Definition 4.3 (Optimistic Labeled Certified Mail)
An *optimistic labeled certified mail scheme* for a message space M, a label space L and a set of transaction identifiers $TIDs$ is a triple $(\mathsf{S}, \mathsf{R}, \mathsf{V})$ and a machine T of probabilistic interactive machines. The scheme must fulfill the following requirements:

[5]This secrecy requirement will be formalized in the traditional way in Section 4.2.2 and in Section 9.2 using our new notion of security for reactive systems.

Requirement 4.3a (Security): $(\mathsf{S}, \mathsf{R}, \mathsf{V})$ with a set $AM := \{\mathsf{T}\}$ with a correct T is a secure labeled certified mail scheme (Def. 4.2) where machine T does not participate in the “show“-protocol.

Requirement 4.3b (Limited Trust in T*):* R. 4.2b, R. 4.2c, and R. 4.2d are fulfilled even if T is incorrect.

Furthermore, one of the following requirements must be fulfilled:

Requirement 4.3c (Optimistic on Agreement on Synchronous Network): If a correct sender S inputs $(\mathsf{send}, \mathsf{R}, l, m, tid)$ and a correct recipient R inputs $(\mathsf{receive}, \mathsf{S}, l, tid)$ in a given round then the third party does not send or receive messages in the “send“-protocol.

Requirement 4.3d (Optimistic on Agreement on Asynchronous Network): If a correct sender S inputs $(\mathsf{send}, \mathsf{R}, l, m, tid)$, a correct recipient R inputs $(\mathsf{receive}, \mathsf{S}, l, tid)$, and both do not input (wakeup, tid), then the “send“-protocol terminates in a fixed time and the third party does not send or receive messages.

An optimistic labeled certified mail scheme with $|L| \geq 2$ is called *optimistic on disagreement* if the third party does not send or receive messages in the “sign“-protocol, even if the sender and the recipient input different labels. $\diamond$

Remark 4.6. Optimism on agreement is only efficient if it is likely that the players have agreed on the label before starting the protocol. In many applications, this will be the case since the execution of the certified mail protocol is, e.g., just one step in a larger electronic commerce transaction.

Remark 4.7. Requirement 4.2a for an incorrect T automatically holds in all optimistic executions. However, if the protocol does not guarantee optimism on disagreement, “correct execution” may no longer be guaranteed.

Remark 4.8. A preliminary analysis of time-optimal labeled certified mail protocol has shown that Requirement 4.2c seems to be the main reason why certified mail requires more time than contract signing in some models. It guarantees that a collusion of the sender and an incorrect third party cannot later change the message contained in a receipt. At the beginning, the recipient has no information about the message. Therefore, it cannot be fixed before time 2. Dropping R. 4.2c from R. 4.3b would therefore enable us to save one round in some cases. In this case, a signature of the third party would be sufficient to fix the message but an incorrect third party together with the sender would be enabled to forge receipts.

Remark 4.9. An alternative to dropping Requirement 4.2c is to assume that the recipient, e.g., knows a commitment fixing the expected message beforehand (it has to know the label anyway). This may enable more efficient protocols as well. However, we do not consider such optimizations in the sequel since we aim at a service that is independent of the internal details (such as commitments) of an actual protocol implementing the service. An advantage of this service-based approach is that implementations of services can be interchanged and that they can be evaluated independently of the protocol using them.

Remark 4.10. Unlike contract signing, in general, we do not consider protocols where the third party participates in the "show"-protocol, even though the resulting revocability of receipts may increase the efficiency in some cases. ○

4.2.2 Traditional Formalization of the Secrecy Requirement

Compared to contract signing, which can be specified using integrity requirements, certified mail also requires secrecy of the message. In Section 3.2.4, we sketched the traditional way to formalize integrity requirements. We now show how to formalize the secrecy of certified mail.

The following definition defines that the adversary must not be able to obtain partial information on the message if no receipt is issued. "Partial information" is modeled similar to the notion of "indistinguishable encryptions" [Gold 93, GoMi 84]: If the adversary is somehow able to determine any feature of the message given the ciphertext, then there are two ciphertexts (e.g., one with and one without this feature) that it is able to distinguish. Therefore, in the following definition, the adversary is asked to choose two ciphertexts and later tries to guess which one was used.

For certified mail, this means that a run of certified mail is executed with a message m_i randomly chosen out of a pair (m_0, m_1) of two messages selected by the adversary. Then, if no receipt was issued, the adversary tries to guess which of the two messages was actually input in this particular run. If the adversary is able to guess the correct message in non-negligibly more than 50% of the runs without receipt, then it was able to obtain some partial information and the scheme is insecure.

Definition 4.4 (Traditional Formalization of "Secrecy of Message")
Let $(\mathsf{S}, \mathsf{R}, \mathsf{V})$ and T be a given labeled certified mail scheme for a message space $M := \{0,1\}^k$, a label space $L := \{0,1\}^l$, and a set TIDs of possible tids. Each machine M obtains a security parameter $n \in \mathbb{N}$ at startup. This is denoted by $\mathsf{M}(n)$.

For the following game, the recipient is replaced by one polynomial time interactive Turing machine R^* that has one additional channel to a message chooser C and to the user-input channel of the sender. Besides the channel with R^*, the chooser has one output channel to the user-input channel of the sender S (see Figure 4.2). The game is defined as follows:

1. The chooser chooses a fresh $\mathit{tid} \in \mathit{TIDs}$ and sends it to the adversary $\mathsf{R}^*(n)$.

2. $\mathsf{R}^*(n)$ interacts arbitrarily with the correct machines $\mathsf{S}(n)$, $\mathsf{T}(n)$, and $\mathsf{V}(n)$ of the labeled certified mail scheme. This includes user-inputs to all correct machines. Inputs to the sender are forwarded by the chooser, if the used tid differs from the chosen tid.

3. $\mathsf{R}^*(n)$ sends two messages m_0 and m_1 and a label l to the chooser C.

4. The chooser verifies that $m_0, m_1 \in M$ and $l \in L$ and chooses a random bit $i \in \{0,1\}$. It then inputs $(\mathsf{send}, \mathsf{R}, l, m_i, \mathit{tid})$ to the sender and stops.

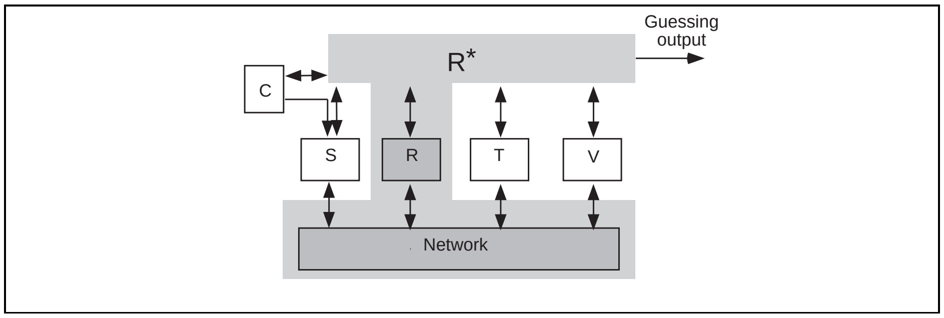


Figure 4.2: Evaluating the Message Secrecy Requirement.

5. $\mathsf{R}^*(n)$ interacts arbitrarily with the correct machines of the labeled certified mail scheme. This includes user-inputs to all correct machines. Inputs to the sender are again forwarded by the chooser, if the used tid differs from the chosen tid.

6. $\mathsf{R}^*(n)$ outputs a guess i^*.

7. Finally, $\mathsf{R}^*(n)$ wins if the sender output (failed, tid) and the guess was correct, i.e., $i^* = i$.

Let $P_f(n)$ be the probability that the sender outputs failed and $P_w(n)$ the probability that the game is won, i.e., that failed was output and the guess was correct.

The scheme provides secrecy of the input message if for any given adversary $\mathsf{R}^*(n)$, the adversary can guess the right messages in about half the runs where the sender output failed, i.e.,

$$\forall p \in poly : \exists n_0 \in \mathbb{N} : \forall n > n_0 : P_w(n) \leq \frac{1}{2}P_f(n) + \frac{1}{p(n)},$$

where $poly$ denotes the set of all polynomials in n. ◇

Remark 4.11. The condition is similar to the condition that $P(wins|failed) \leq 1/2 + 1/p(n)$, except that this condition would lead to problems if P_f is exponentially small.

Remark 4.12. In Section 9.2, we will give an alternative trusted-host specification of certified mail and prove the security of one of our protocols using our new formalism. ○

4.3 Contract Signing using Labeled Certified Mail

We now show how to provide contract signing based on labeled certified mail without any additional messages.

The basic idea of the reduction is to send a signed contract text with labeled certified mail to the recipient while verifying agreement by including the contract text into the label. Then, the received message is defined to be a valid contract for the recipient whereas the receipt on the same message is defined to be the valid contract for the sender. Naturally, this requires that the contract can be used as a label.

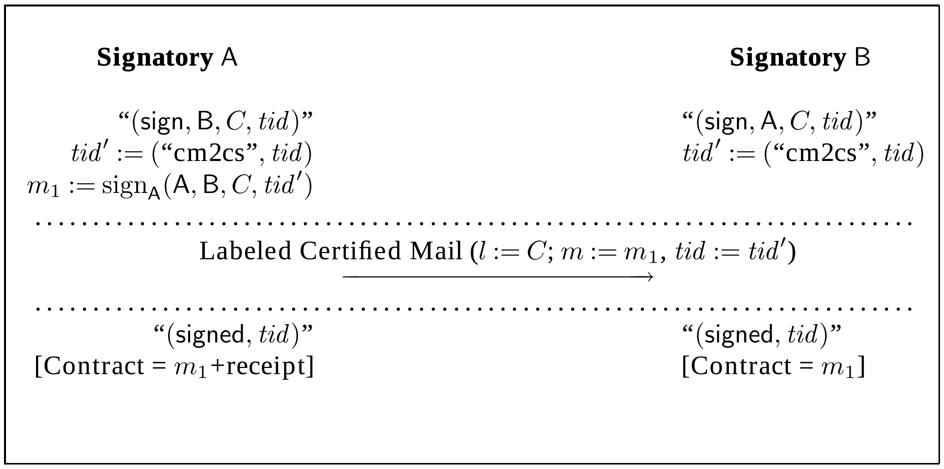

Signatory A
Signatory B
“(sign, B, C, tid)”
$tid' :=$ (“cm2cs”, tid)
$m_1 := \text{sign}_{\mathsf{A}}(\mathsf{A}, \mathsf{B}, C, tid')$
“(sign, A, C, tid)”
$tid' :=$ (“cm2cs”, tid)
Labeled Certified Mail ($l := C$; $m := m_1$, $tid := tid'$)
“(signed, tid)”
[Contract = m_1+receipt]
“(signed, tid)”
[Contract = m_1]

Figure 4.3: Optimistic Behavior of Scheme 4.1 (Contract Signing with Labeled Certified Mail): Sending a signed contract m_1 using the contract text C as the label.

Theorem 4.1 (Contract Signing with Labeled Certified Mail)
Any scheme for labeled certified mail with a label space L and a message space M in time t with m messages can be used to construct a contract signing scheme for a contract space L in time t with m messages. The contract signing scheme is optimistic on agreement / disagreement (Def. 3.4) if the used labeled certified mail scheme is optimistic on agreement / disagreement (Def. 4.3). □

A consequence of this theorem is that no scheme for labeled certified mail can be more efficient than a contract signing scheme for the same model (recall, our lower bounds on contract signing only assumed that $|M| \geq 2$, i.e., the result holds for any valid label space L). We prove the theorem by describing a scheme providing contract signing using labeled certified mail:

Scheme 4.1 (Contract Signing with Labeled Certified Mail)
Let any labeled certified mail scheme $(\mathsf{S_{cm}}, \mathsf{R_{cm}}, \mathsf{V_{cm}})$, $\mathsf{T_{cm}}$ be given.

We define a contract signing scheme $(\langle \mathsf{A}, \mathsf{S_{cm}} \rangle, \langle \mathsf{B}, \mathsf{R_{cm}} \rangle, \langle \mathsf{V}, \mathsf{V_{cm}} \rangle)$ and $\langle \mathsf{T}, \mathsf{T_{cm}} \rangle$. The behavior of the simulation machines A, B, V, and T is defined as follows:

Signing a Contract (Protocol “sign“*; Fig. 4.3):* On input (sign, B, C, tid), machine A derives a fresh and unique transaction identifier $tid' :=$ (“cm2cs”, tid) from the input tid, computes $m_1 := \text{sign}_{\mathsf{A}}(\mathsf{A}, \mathsf{B}, C, tid')$, and inputs (send, B, C, m_1, tid') to $\mathsf{S_{cm}}$. If this algorithm outputs (sent, tid'), A outputs (signed, tid), else it outputs (rejected, tid).

On input (sign, A, C, tid), machine B computes tid' and inputs (receive, A, C, tid') into the labeled certified mail algorithm $\mathsf{R_{cm}}$. If $\mathsf{R_{cm}}$ outputs (received, m_1, tid') with a correct message $m_1 = \text{sign}_{\mathsf{A}}(\mathsf{A}, \mathsf{B}, C, tid')$, B outputs (signed, tid) and (rejected, tid) else.

On input (wakeup, tid), the machines A or B input (wakeup, tid') to $\mathsf{S_{cm}}$ or $\mathsf{R_{cm}}$, respectively.

Verification of a Contract (Protocol “show“*):* On input (show, tid) to machine B, it sends $\mathrm{sign}_{\mathsf{B}}(m_1)$ to V who outputs (signed, A, B, C, tid) if m_1 is correct and (rejected, tid) else.

On input (show, tid) to machine A, it sends $\mathrm{sign}_{\mathsf{A}}(m_1)$ to V and inputs (show, tid') to S_{cm}. If V_{cm} outputs (received, A, B, C, m_1, tid'), machine V verifies that m_1 is correct. If this is the case, the verifier outputs (signed, A, B, C, tid). Else, it outputs (rejected, tid).

◇

Lemma 4.1 (Security of Scheme 4.1)
Scheme 4.1 is a secure contract signing scheme (Def. 3.3). It is optimistic on agreement / disagreement (Def. 3.4) if the used labeled certified mail scheme is optimistic on agreement / disagreement (Def. 4.3). □

Proof. The scheme adheres to Definition 3.2 by construction. We now show that the requirements of Definition 3.3 and 3.4 are fulfilled:

Correct Execution (R. 3.3a) Let us assume that wakeup was not input on asynchronous networks and that both initial inputs were made in the same round on synchronous networks. Then, if both correct machines A and B get the inputs (sign, B, C, tid) and (sign, A, C, tid) with identical tid and C, then B gets an output (received, m_1, tid') from R_{cm} with a correct m_1 [R. 4.2a] and outputs (signed, tid) whereas A receives (sent, tid') from S_{cm} [R. 4.2a] and outputs (signed, tid) as well. If both input different contracts, S_{cm} and R_{cm} output (failed, tid') since the input labels are different which will lead to outputs (rejected, tid) by both [cf. Remark 4.4].

Unforgeability of Contracts (R. 3.3b+R. 3.4b) For successful verification, B needs $\mathrm{sign}_{\mathsf{B}}(m_1)$ even with incorrect T. Since m_1 is signed by A, a correct A received an input (sign, B, C, tid).

Signatory A cannot produce $\mathrm{sign}_{\mathsf{B}}(m_1)$ since the signature scheme is secure. Therefore, for successful verification, the verifier V_{cm} is required to output (received, A, B, C, m_1, tid'). This output only occurs after an input (receive, A, C, tid') by B to R_{cm} even if T is incorrect [R. 4.2b+R. 4.3b]. From the construction of a correct B, this input is only made on input (sign, A, C, tid).

Verifiability of Valid Contracts (R. 3.3c+R. 3.4b) If a correct A output (signed, tid), A received an output (sent, tid') from S_{cm} on input (send, B, C, m_1, tid'). Therefore, the certified mail scheme will output (received, A, B, C, m_1, tid') to V, too [R. 4.2d].

If a correct B output (signed, tid), B received m_1 which will be accepted as a valid contract.

No Surprises with Invalid Contracts (R. 3.3d) If a correct A output (rejected, tid), then (failed, tid') was output to A. Therefore, B is not able to obtain the message m_1 that is otherwise never sent by A [R. 4.2f]. Thus machine B will not be able to pass verification.

If a correct B obtains an output (rejected, tid), either m_1 was incorrect or else, B obtained an output (failed, tid') from R_{cm}. If m_1 was incorrect, this

m_1 is included in the receipt, too [R. 4.2c], and A is not able to pass verification. If B obtained an output (failed, tid'), the certified mail scheme will not output (received, A, B, C, m_1, tid') to the verifier [R. 4.2e].

Termination on Asynchronous Network (R. 3.3f) On input (wakeup, tid), the machines input (wakeup, tid') to the certified mail scheme which gives a definitive output [R. 4.2h]. This output leads to an output of the contract signing scheme within the same time.

Optimistic on Asynchronous Network (R. 3.4d) If both contracts and tid's are equal and none of the signatories receives (wakeup, tid), the scheme does not input wakeup to the underlying scheme and does not contact the third party itself. Therefore, and as equal contracts imply equal labels, the resulting scheme is as optimistic as the underlying scheme [R. 4.3d].

■

4.4 Message-optimal Labeled Certified Mail

We now describe a scheme for message-optimal optimistic labeled certified mail based on message-optimal optimistic contract signing and prove its security and optimality. For easier readability, we first present a simulation that needs additional messages. Then, we optimize the scheme for the message-optimal contract signing schemes as described in Chapter 3. In this case, no additional messages are needed.

Theorem 4.2 (Labeled Certified Mail with Contract Signing)
Let (t, m) be the complexity of a message-optimal optimistic contract signing protocol (Def. 3.4) for the considered models.

Then there exists a message-optimal optimistic labeled certified mail scheme (Def. 4.3) in time t with m messages for the same model. □

We now describe a reduction and its message-optimization and finally prove the theorem by showing that this optimization can be applied to the message-optimal contract signing protocols described in Chapter 3.

The basic idea of the following reduction is that the sender encrypts the message for T, uses the ciphertext as the contract, and finally enables the recipient to verify the encryption by revealing the cleartext (including the message). If, however, the sender refuses to reveal these inputs, the third party decrypts the message if and only if the contract has been signed.

Scheme 4.2 (Labeled Certified Mail with Contract Signing)
Let $(\mathsf{A}_{cs}, \mathsf{B}_{cs}, \mathsf{V}_{cs})$, T_{cs} be an optimistic contract signing scheme. We then define a labeled certified mail scheme $(\langle \mathsf{S}, \mathsf{A}_{cs}\rangle, \langle \mathsf{R}, \mathsf{B}_{cs}\rangle, \langle \mathsf{V}, \mathsf{V}_{cs}\rangle)$ and $\langle \mathsf{T}, \mathsf{T}_{cs}, \mathsf{V}_{cs}\rangle$[6]. The behavior of the simulation machines S, R, V, and T is defined as follows:

Sending Mail (Protocol "send"*; Figure 4.4):* On input (send, R, l, m, tid), the sender S chooses a random r and computes $tid' = (\text{“cs2cm”}, tid)$ and

[6]Recall that the verifier is state-less, i.e., the two copies will show the same behavior.

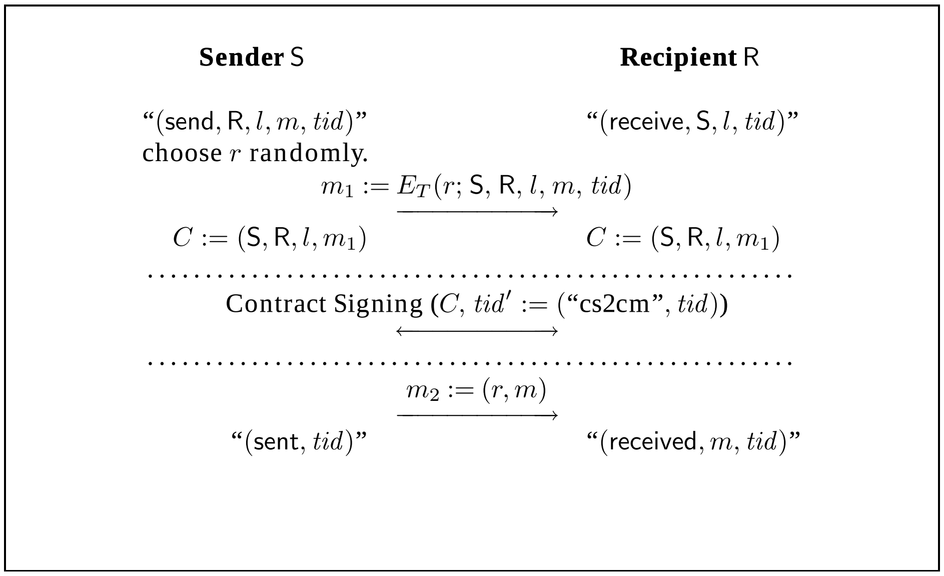


Figure 4.4: Optimistic Behavior of Scheme 4.2 (Labeled Certified Mail with Contract Signing).

sends $m_1 := E_T(r;$ S, R, l, m, $tid)$ to R. Then, in the next round on synchronous networks, it inputs (sign, R, (S, R, l, m_1), tid') to the contract signing scheme. If $\mathsf{A_{cs}}$ outputs (signed, tid'), S sends $m_2 := (r, m)$ to R and outputs (sent, tid). If S receives wakeup, it inputs (wakeup, tid) to $\mathsf{A_{cs}}$. If $\mathsf{A_{cs}}$ outputs (rejected, tid') the sender S outputs (failed, tid) without sending m_2.

If R gets an input (receive, S, l, tid) and receives an m_1, it inputs (sign, S, (S, R, l, m_1), tid') to $\mathsf{B_{cs}}$. If (wakeup, tid) is input before receiving m_1, machine R outputs (failed, tid). If $\mathsf{B_{cs}}$ outputs (signed, tid) and R receives m_2, it checks whether m_2 fits m_1, i.e., whether re-computing E_T with the known inputs produces the ciphertext. If it fits, R outputs (received, m, tid). Else, it starts the sub-protocol "resolve". If (wakeup, tid) is input to R before obtaining an output from $\mathsf{B_{cs}}$, it inputs (wakeup, tid') to $\mathsf{B_{cs}}$. If it receives (wakeup, tid) while waiting for m_2, it starts "resolve". If machine $\mathsf{B_{cs}}$ outputs (rejected, tid'), the recipient outputs (failed, tid).

Recovery from Exceptions (Sub-protocol "resolve"*):* R inputs (show, tid') to $\mathsf{B_{cs}}$. If $\mathsf{V_{cs}}$ outputs (signed, S, R, (S, R, l, m_1), tid') to T, the third party T verifies that the output parameters of the contract signing protocol match the parameters contained in the contract. Then, it decrypts m_1 and checks that it also contains the correct parameters[7].

If the checks succeed, T sends m_2 to R who outputs (received, m, tid). If the checks fail, it sends $m_3 := \mathrm{sign}_{\mathsf{T}}$(aborted, tid) to R who outputs (failed, tid).

[7]Otherwise, the recipient may start another execution of the certified mail protocol while re-using m_1 as the contract. Then, the third party would decrypt the message in "resolve" since it was part of a valid contract. Thus, R would obtain the message even though S output failed. For this reason, we need non-malleability of the encryption scheme [DoDN 91] not only for secrecy but also for integrity.

Verification (Protocol "show"*):* On input (show, tid), the sender sends m_2 to the verifier V and inputs (show, tid') to A_{cs}. If V_{cs} outputs (signed, S, R, $(\mathsf{S}, \mathsf{R}, l, m_1)$, tid'), the verifier V verifies that the output parameters of the contract signing protocol match the parameters contained in the contract. Then, it checks that repeating the encryption using the outputs of the contract verification as well as (r, m) as contained in m_2 in fact produces m_1. If this is the case, the verifier V outputs (received, S, R, l, m, tid). Else, it outputs (failed, tid).

◇

Lemma 4.2 (Security of Scheme 4.2)
Scheme 4.2 is a secure labeled certified mail scheme (Def. 4.2). It is optimistic on agreement/disagreement (Def. 4.3) if the used contract signing scheme is optimistic on agreement/disagreement (Def. 3.4). □

Proof. The scheme adheres to Def. 4.1 by construction. We now show that the requirements in Def. 4.2 and 4.3 hold:

Correct Execution (R. 4.2a): On input (send, R, l, m, tid) to S and (receive, S, l, tid) to R they start the contract signing protocol at the same time with matching parameters and, if none inputs wakeup, the correct sender outputs (sent, tid) after correct execution of the contract signing protocol [R. 3.3a]. The correct recipient receives m_1 and m_2 and outputs (received, m, tid).

If the recipient outputs (received, m, tid), a correct sender sent m_1 on input (send, R, l, m, tid).

Unforgeability of Receipts (R. 4.2b+R. 4.3b): If the verifier outputs (received, S, R, l, m, tid), then V_{cs} output (signed, S, R, $(\mathsf{S}, \mathsf{R}, l, m_1)$, tid') which implies that a correct recipient input (sign, S, $(\mathsf{S}, \mathsf{R}, l, m_1)$, tid'), even if T is incorrect [R. 3.3b+R. 3.4b]. By construction, this input is only made by a correct recipient on input of (receive, S, l, tid).

Receipts are Fixed (R. 4.2c+R. 4.3b): If a recipient output (received, m, tid) on input (receive, S, l, tid), the contract signing scheme output (signed, tid') on input (sign, S, $(\mathsf{S}, \mathsf{R}, l, m_1)$, tid). From [R. 3.3b+R. 3.4b] follows that there cannot be a different output (signed, S, R, C', tid') for V_{cs} even if T is incorrect. Therefore, in order to obtain an output (sent, S, R, l, m', tid) at the verifier, the incorrect sender would be required to show a message $m_2' = (m', r')$ with $m' \neq m$ that encrypts to the same ciphertext m_1 as fixed by the contract. This, however, contradicts the assumptions that it is hard to find two different messages that encrypt to the same ciphertext.

Verifiability of Valid Receipts (R. 4.2d+R. 4.3b): If the scheme outputs (sent, tid) to the correct sender, the contract signing protocol was executed successfully, i.e., the verification protocol of the contract signing scheme will output (signed, S, R, $(\mathsf{S}, \mathsf{R}, l, m_1)$, tid') [R. 3.3c+R. 3.4b] with the parameters input by the correct sender. Furthermore, repeating the encryption using m_2 will result in m_1 as contained in the contract. Therefore, by construction, the verifier will decide on (received, S, R, l, m, tid).

No Surprises for the Recipient (R. 4.2e): If the recipient output (failed, tid) and a contract was produced, this contract does not contain a valid ciphertext m_1 (otherwise, "resolve" would have output the message). Therefore, a correct verifier V will output (rejected, tid).

If no valid contract was produced, i.e., no input (sign, S, (S, R, l, m_1), tid') was made to $\mathsf{B_{cs}}$ or such an input was answered with (rejected, tid') (e.g., since $l_S \neq l_R$ and thus $C_A \neq C_B$), then $\mathsf{V_{cs}}$ will not output (signed, S, R, (S, R, l, m_1), tid') [R. 3.3b or R. 3.3d, respectively] and therefore V will not output (received, S, R, l, m, tid).

Secrecy of Message (R. 4.2f): If a correct sender output (failed, tid), the contract signing scheme output (rejected, tid') and m_2 was not sent. From the security of the encryption scheme follows that the recipient cannot gain knowledge about m with m_1 only. If we now assume that the recipient re-uses m_1 in a protocol run without S, then our assumptions on the encryption scheme guarantee that changing S in the plaintext also changes the ciphertext. Therefore, the third party will not reveal m after decrypting m_1 since the machine S as identified inside m_1 did not participate in the contract signing protocol. In addition, non-malleability guarantees that R is unable to produce a ciphertext m_1^* that will be decrypted by T while revealing information about m.

Furthermore, the recipient is not able to convince the third party to decrypt m_1: The verification of the contract between S and R will not output (signed, S, R, (S, R, l, m_1), tid') for the same tuple (S, m_1, tid') [R. 3.3b+R. 3.3d] since S will not participate in another run with the same tid'.

Termination on Asynchronous Network (R. 4.2h): If a machine gets a wakeup, it either produces an output directly or after receiving an output from the contract signing scheme after an input wakeup [R. 3.3f].

Optimistic on Disagreement/Agreement on Asynchronous Network (R. 4.3d):
If a correct sender S inputs (send, R, l, m, tid) and a correct recipient R inputs (receive, S, l, tid), they input (sign, S, C, tid') and (sign, R, C, tid'), respectively (this is done in the same round on synchronous networks). From [R. 3.3a] follows that this leads to an output (signed, tid') without contacting the third party [R. 3.4d]. Since m_1 and m_2 are correct, this leads to the outputs sent / received without contacting the third party.

If the labels differ, the contracts are different and the underlying scheme will output (rejected, tid'). If the underlying scheme is optimistic on disagreement, this output will be produced without contacting the third party.

■

We now show how to optimize Scheme 4.2 to provide message-optimal labeled certified mail based on the message-optimal contract signing protocols described in Chapter 3. Since the optimization can be applied to message-optimal contract signing schemes and the optimized simulation does not need additional messages or time, the message-optimality directly follows from Theorem 4.1.

Scheme 4.3 (Message Optimization of Scheme 4.2)
Let $(\mathsf{A}_{\mathsf{cs}}, \mathsf{B}_{\mathsf{cs}}, \mathsf{V}_{\mathsf{cs}})$ and T_{cs} be an optimistic contract signing scheme satisfying the following conditions:

1. Machine A_{cs} sends the only message m_{1A} at time 1.
2. There exists a message m_{kA} that is sent by a correct A_{cs} such that:
   - If the signatories are correct and agree, then m_{kA} is always sent.
   - If m_{kA} is sent[8], then A_{cs} will eventually output (signed, tid') (either directly or else after recovering with T).

Then Scheme 4.3 is constructed by concatenating message m_1 of Scheme 4.2 to m_{1A} and concatenating m_2 of Scheme 4.2 to m_{kA}. If m_{kA} is not sent (i.e., in case of disagreement or non-optimistic cases), the protocol proceeds without changes, i.e., the optimization waits for an output of the underlying protocol as well and sends m_2 only if this output is (signed, tid'). ◇

Remark 4.13. A counterexample where A_{cs} cannot be sure that the output will be signed, even after sending its last message of the optimistic phase, can be constructed as follows: Consider a protocol m_1, m_2, m_3, m_4 started by A_{cs} where the last message "opens" the messages m_1 and m_3 that are encrypted for, e.g., machine T. In this case, A_{cs} sends m_3 even if the output will be rejected because m_2 was already wrong internally.

Remark 4.14. In the time-optimal protocols from Chapter 3, both signatories start by sending a signature under the contract to the peer. Therefore, our optimization cannot be applied to these time-optimal schemes. Still, the resulting protocol can be partly optimized by concatenating the second message m_2 sent by A_{cs} to the last message of the protocol.

Remark 4.15. For some models, our message-optimal protocols are not the first optimally efficient protocols. Two protocols for optimism on agreement and two-party disputes have been proposed in [Mica 97] for synchronous networks and [AsSW 98] for asynchronous networks, respectively. Their message-optimality in case of agreement follows from Theorem 4.2.

Remark 4.16. Even though we ruled out three-party disputes, a contract signing protocol with three-party disputes that satisfies the conditions will result in a message-optimal labeled certified mail protocol with three-party disputes as well. An example of a case where this lemma holds is Scheme 3.4, since sending m_3 guarantees that the output to A_{cs} will be signed. ○

Lemma 4.3 (Security of Scheme 4.3)
Scheme 4.3 is an optimistic labeled certified mail scheme. □

Proof. The scheme adheres to Def. 4.1 by construction. We now show that the requirements in Def. 4.2 and 4.3 hold:

Correct Execution (R. 4.2a): Since B_{cm} does not send a message at time 1, A sends two subsequent messages m_{1A} (of the contract signing scheme)

[8]Note that this does not imply that A_{cs} must not send a message in case of rejected but rather that this message can be distinguished from m_{kA}.

and m_1 (of the simulation) without receiving any messages in between. Thus, they may as well be concatenated without changing the behavior on asynchronous networks. Therefore, the reasoning in Lemma 4.2 still holds.

Secrecy of Message (R. 4.2f): If $\mathsf{A_{cs}}$ sends m_{kA} before receiving later messages from $\mathsf{B_{cs}}$, the optimization requires $\mathsf{A_{cs}}$ to send m_2 before receiving these later messages of the contract signing protocol. Thus, $\mathsf{B_{cs}}$ is enabled to obtain the confidential message before $\mathsf{A_{cs}}$ obtains a complete contract. However, condition 2 guarantees that $\mathsf{A_{cs}}$ will eventually output (signed, tid') either directly or after recovery (otherwise, m_{kA} would not be sent). Therefore, S will output (sent, tid), i.e., S obtained a correct receipt.

For all other requirements the reasoning in the proof of Lemma 4.2 holds without changes. ■

Proof. (**Theorem 4.2**)
For proving the theorem, we have to show that the preconditions of Scheme 4.3 are fulfilled for the message-optimal schemes in Chapter 3 and that the optimistic complexity of the resulting protocol is not worse than the complexity of the underlying contract signing protocol.

For the considered model without three-party disputes, the message-optimal schemes (Schemes 3.1, [AsSW 98], and Scheme 3.7) satisfy the condition that $\mathsf{A_{cs}}$ sends the first message. For the schemes that are optimistic on agreement, the last message sent by $\mathsf{A_{cs}}$ in case of agreement satisfies condition 2. For Scheme 3.7, the existence of such a message follows from the fact that it exists in the underlying scheme that is optimistic on agreement.

Let (t, m) be the complexity of the underlying protocol. Let us now assume that the protocol sends m_2 without sending m_{kA} in case of agreement, i.e., the optimization had to wait for an output (signed, tid') of the underlying contract signing scheme thus increasing the time and messages needed. In this case, condition 2 guarantees that the protocol execution was non-optimistic (otherwise m_{kA} would have been sent). Thus, the execution does not increase the optimistic complexity as defined in Def. 3.6. In case of disagreement, the underlying scheme will produce an output $(\mathsf{rejected}, tid')$ at the end and m_2 will not be sent. Thus, the optimization does not increase the time and message complexity of the scheme in case of disagreement, too. ■

4.5 Labeling "Traditional" Certified Mail

In this section, we describe how to label traditional optimistic certified mail without losing efficiency. This disproves the objection that labeled certified mail is less efficient than traditional optimistic certified mail, i.e., that optimistic contract signing can be based on traditional certified mail as well.

4.5.1 Definition

The traditional notion of optimistic certified mail is like labeled optimistic certified mail without labels: Both players still input a tid to signal that their inputs belong together. However, the recipient is no longer able to specify a label under which the message shall be received.

Definition 4.5 (Optimistic Certified Mail)
A (traditional) optimistic certified mail scheme is an optimistic labeled certified mail scheme (Def. 4.3) with only the empty label in the set of labels, i.e., $L := \{\epsilon\}$.
$\diamond$

Remark 4.17. Note that without labels, the distinction between optimistic on agreement and disagreement no longer makes sense. $\circ$

4.5.2 Labeling "Traditional" Certified Mail

We now show how to adapt any optimistic certified mail protocol to labeled certified mail. The basic idea of this simulation is to augment the initial messages with a signature on the input label and tid. If the protocol is executed without changes, this signature is then needed for showing a receipt. One issue that must be solved by the simulation is how to cope with differing labels, i.e., how to securely abort the protocol that has been started by sending the initial messages.

Theorem 4.3 (Labeling Certified Mail Protocols)
Traditional optimistic certified mail protocols can be adapted to provide labeled certified mail with the same efficiency in the optimistic case. $\square$

Proof. We simulate optimistic labeled certified mail using optimistic certified mail depending on the structure of the optimistic certified mail protocol.

In the first round of the optimistic "send"-protocol, the protocol sends at most two messages m_{1S} and m_{1R}. We now present three simulations covering all optimistic protocols: Scheme 4.4 can be applied if m_{1R} and m_{1S} are sent. Scheme 4.5 can be applied if only m_{1S} is sent, and Scheme 4.6 can be applied if only m_{1R} is sent. All simulations do not need additional messages or time.

From the security of these simulations then follows that any optimistic certified mail scheme can be adopted for labeled certified mail without losing efficiency. $\blacksquare$

Scheme 4.4 (Labeled Certified Mail from Certified Mail (1 of 3))
Let $(\mathsf{S}_{\mathsf{cm}}, \mathsf{R}_{\mathsf{cm}}, \mathsf{V}_{\mathsf{cm}}), \mathsf{T}_{\mathsf{cm}}$ be an optimistic certified mail scheme with two messages in the initial round. Let m_{1S} be the message sent by the sender and let m_{1R} be that of the recipient.

We assume without loss of generality that the behaviour of any machine in this scheme for one tid does not depend on results of transactions for other tid's.[9,10]

[9] The assumption is needed because below, R_{cm} is sometimes aborted by R. If, for instance, S_{cm} could store this, it would not need to fulfil "correct execution" in future runs with this R_{cm}, because R_{cm} is not correct in the sense of the underlying scheme.

[10] This is without loss of generality because otherwise, we could adapt the following scheme by always running the simulated R_{cm} with a new identity, i.e., signature key. The machine R using R_{cm} would accompany the first messages of each run with a certificate of this new key, signed with its normal key. Then one is always in the case where R_{cm} has been correct so far. The complexity of the underlying scheme is no lower than its complexity in this case.

We define a labeled certified mail scheme $(\langle \mathsf{S}, \mathsf{S}_{\mathsf{cm}}\rangle, \langle \mathsf{R}, \mathsf{R}_{\mathsf{cm}}\rangle, \langle \mathsf{V}, \mathsf{V}_{\mathsf{cm}}\rangle)$ and $\langle \mathsf{T}, \mathsf{T}_{\mathsf{cm}}\rangle$. If not mentioned otherwise, all simulation machines forward any message output by their sub-machine to the intended recipient sub-machine. The behavior of the simulation machines S, R, V, and T is defined as follows:

Sending a Message (Protocol "send"*):* On input (send, R, l_S, m, tid), the sender inputs (send, R, ϵ, m, tid') with $tid' :=$ ("cm2lcmSR", tid) to S_{cm}. If S_{cm} outputs m_{1S} to be sent to R_{cm}, machine S sends (m_{1S}, m_S) with $m_S := \text{sign}_{\mathsf{S}}(l_S, tid')$ to R instead. If machine S receives (m_{1R}, m_R) with $m_R = \text{sign}_{\mathsf{R}}(l_R, tid')$ and $l_R = l_S$, the protocol continues without changes. Else, or if wakeup was input, S ignores m_{1R} and inputs (wakeup, tid') to the underlying scheme. If any message that arrives in the sequel is augmented with m_R then S stores this m_R. If machine S_{cm} outputs (sent, tid') and m_R has arrived, then S outputs (sent, tid), else (failed, tid).

On input (receive, S, l_R, tid) the recipient inputs (receive, S, ϵ, tid') to R_{cm}. If R_{cm} outputs m_{1R} to be sent to S_{cm}, machine R sends (m_{1R}, m_R) to S instead. If wakeup is input or if R receives (m_{1S}, m_S) with $m_S = \text{sign}_{\mathsf{S}}(l_S, tid')$ and $l_S \neq l_R$, it terminates the algorithm R_{cm} and outputs (failed, tid) without inputting m_{1S}. Else, the protocol continues. If R_{cm} outputs a recovery message m_T to be sent to T_{cm}, the simulation machine R appends (m_S, m_R) to m_T and sends it to machine T. If machine R_{cm} outputs (received, m, tid'), then R outputs (received, m, tid).

If (wakeup, tid) is received by a machine, the machine inputs (wakeup, tid') to the corresponding machine of the underlying certified mail scheme.

If a machine of the underlying scheme outputs (failed, tid'), the scheme outputs (failed, tid).

Recovery with $\langle \mathsf{T}, \mathsf{T}_{\mathsf{cm}}\rangle$*:* The first recovery message from R_{cm} sent to T is only input to T_{cm} if a correct pair (m_S, m_R) of signed messages with matching labels $l_S = l_R$ is appended. Else, all subsequent recovery messages from R are ignored.

Answers to recovery requests from S_{cm} are augmented with m_R if R_{cm} has sent recovery messages for this tid before. Else, T executes recovery of S_{cm} without changes. S accepts messages from T with and without m_R, while it ignores messages from R until it receives m_R.

Verification of a Receipt (Protocol "show"*):* On input (show, tid), S inputs (show, tid') to S_{cm} and sends m_R to the verifier.

The verifier V outputs (received, S, R, l, m, tid) if the underlying scheme outputs (received, S, R, ϵ, m, tid') and the sender is able to present an $m_R := \text{sign}_{\mathsf{R}}(l, tid')$. Else, it outputs (failed, tid).

◇

Lemma 4.4 (Security of Scheme 4.4)
Scheme 4.4 is a secure optimistic labeled certified mail scheme (Def. 4.3). □

Proof. The scheme adheres to Definition 4.1 by construction. We now show that each of the requirements described in Definitions 4.2 and 4.3 is fulfilled:

Correct Execution (R. 4.2a): If (send, R, l, m, tid) is input to S and (receive, S, l, tid) is input to R with identical labels, then the messages m_S and m_R are appended to m_{1S} and m_{1R}, respectively, and both include the same labels. Therefore, the protocol continues without changes. From [R. 4.2a] follows that the protocol then produces the correct outputs.

If a correct recipient output $(\mathsf{received}, m, tid)$ on input $(\mathsf{receive}, \mathsf{S}, l_R, tid)$ then $\mathsf{R_{cm}}$ was not terminated and output $(\mathsf{received}, m, tid')$ on input $(\mathsf{receive}, \mathsf{S}, \epsilon, tid')$. From [R. 4.2a] follows that a matching input $(\mathsf{send}, \mathsf{R}, \epsilon, m, tid')$ was obtained by $\mathsf{S_{cm}}$. This is only input if a correct S obtained an input $(\mathsf{send}, \mathsf{R}, l_S, m, tid)$ for some l_S. If $l_S \neq l_R$ then a correct S would have sent m_S with $l_S \neq l_R$ and R would have terminated with an output (failed, tid), in contradiction to our precondition that it output received.

Unforgeability of Receipts (R. 4.2b+R. 4.3b) and Receipts are Fixed (R. 4.2c+R. 4.3b): If V outputs (received, S, R, l, m, tid) then $\mathsf{V_{cm}}$ output (received, S, R, ϵ, m, tid') and V received $m_R = \mathrm{sign}_{\mathsf{R}}(l, tid')$. The sending of m_R implies that R obtained an input (receive, S$'$, l, tid), and also $\mathsf{S}' = \mathsf{S}$ because we tacitly included the identities in all signed messages. This proves unforgeability of receipts.

If R output (received, m', tid)) upon this input, then $\mathsf{R_{cm}}$ output (received, m', tid') on input (receive, S, ϵ, tid'). Thus it was not terminated and we can apply [R. 4.2c]. Hence m' equals m in the output of $\mathsf{V_{cm}}$ and thus of V. This proves that receipts are fixed.

Verifiability of Valid Receipts (R. 4.2d+R. 4.3b): If a correct sender output (sent, tid) on input (send, R, l, m, tid), then $\mathsf{S_{cm}}$ output (sent, tid') on input (send, R, ϵ, m, tid'). Moreover, by construction S received an m_R that matches its input parameters tid and l. This is sufficient to verify the receipt.

No Surprises for the Recipient (R. 4.2e): If a correct R output (failed, tid), then either $\mathsf{R_{cm}}$ terminated regularly and output (failed, tid') or R did not send any messages except m_{1R}. In the first case, no surprises follows immediately from no surprises in the underlying scheme. In the second case, we show that the sender cannot convince the verifier $\mathsf{V_{cm}}$ for any message m for this tid'. If a decision received of $\mathsf{V_{cm}}$ would be possible with m_{1R} only, a collusion of an incorrect $\mathsf{S_{cm}}$ and $\mathsf{T_{cm}}$ could claim after a run with a correct $\mathsf{R_{cm}}$ that only m_{1R} was sent and thus produce receipts for arbitrary messages. (Recall that $\mathsf{V_{cm}}$ cannot ask $\mathsf{R_{cm}}$ and that m_{1R} cannot fix the message to be received.) This would contradict [R. 4.2c+4.3b].

Secrecy of Message (R. 4.2f): If the sender S outputs (failed, tid) because the underlying $\mathsf{S_{cm}}$ output (failed, tid'), the underlying protocol guarantees secrecy of the message [R. 4.2f]. Since m is not sent in the additional messages, overall secrecy of m follows by construction.

The only other possibility for S to output (failed, tid) is that it does not receive m_R. We now show that if T is correct (which is assumed for this requirement), this also implies that $\mathsf{S_{cm}}$ outputs (failed, tid') and thus secrecy of the message. By construction, all messages without a valid m_R are ignored by S and T (R can only produce a matching pair (m_R, m_S)

for the message m_S signed by the correct S, who does not sign two labels for the given tid'). If we now assume that S_{cm} outputs (sent, tid') while S did not get m_R, there is no causally preceding message from R_{cm} to S_{cm} and T_{cm} for this tid'. In this case, for verifiability and unforgeability with limited trust in the underlying scheme, the output of S_{cm} must be (failed, tid').

Termination on Asynchronous Network (R. 4.2h): On agreement, this follows from [R. 4.2h] of the underlying scheme. On disagreement, R outputs (failed, tid) immediately while S inputs (wakeup, tid') to S_{cm} upon receipt of a non-matching initial message. If S is correct, termination for S then follows from [R. 4.2h] of the underlying scheme.

Optimistic on Asynchronous Network (R. 4.3d): If both input the same label and tid and none of the players receives wakeup, the scheme does not input wakeup to the underlying certified mail scheme. Thus the resulting scheme is optimistic, too [R. 4.3d].

■

Scheme 4.5 (Labeled Certified Mail from Certified Mail (2 of 3))
Let $(\mathsf{S}_{\mathsf{cm}}, \mathsf{R}_{\mathsf{cm}}, \mathsf{V}_{\mathsf{cm}}), \mathsf{T}_{\mathsf{cm}}$ be an optimistic certified mail scheme with one initial message m_{1S} sent by the sender S_{cm}.

We define a labeled certified mail scheme $(\langle \mathsf{S}, \mathsf{S}_{\mathsf{cm}}\rangle, \langle \mathsf{R}, \mathsf{R}_{\mathsf{cm}}\rangle, \langle \mathsf{V}, \mathsf{V}_{\mathsf{cm}}\rangle)$ and $\langle \mathsf{T}, \mathsf{T}_{\mathsf{cm}}\rangle$. If not mentioned otherwise, all simulation machines forward any message output by their sub-machine to the intended recipient sub-machine. The behavior of the simulation machines S, R, V, and T is defined as follows:

Sending a Message (Protocol "send"*):* On input $(\mathsf{send}, \mathsf{R}, l_S, m, tid)$, the sender inputs $(\mathsf{send}, \mathsf{R}, \epsilon, m, tid')$ with $tid' := (\text{"cm2lcmS"}, tid)$ to S_{cm} and sends (m_{1S}, m_S) with $m_S := \mathrm{sign}_{\mathsf{S}}(l_S, tid')$ instead of m_{1S}. If the sender receives an abort message $m'_{2R} = \mathrm{sign}_{\mathsf{R}}(\mathsf{failed}, tid')$ or an incorrect m_{2R}, the sender inputs (wakeup, tid') to S_{cm}. If any message that arrives in the sequel is augmented with m_R then S stores this m_R. Else, the underlying scheme is executed without changes. If the underlying scheme outputs (sent, tid') and m_R has arrived, the sender outputs (sent, tid).

On input $(\mathsf{receive}, \mathsf{S}, l_R, tid)$ the recipient waits for (m_{1S}, m_S). If the label received in m_S is equal to the input one, i.e., $l_S = l_R$, it inputs $(\mathsf{receive}, \mathsf{S}, \epsilon, tid')$ and m_{1S} to R_{cm}. Furthermore, it appends $m_R := \mathrm{sign}_{\mathsf{R}}(l_R, tid')$ to the first message sent, say m_{2R}. Else, if the labels are different or if wakeup is input before receiving (m_{1S}, m_S), the recipient sends $m'_{2R} := \mathrm{sign}_{\mathsf{R}}(\mathsf{failed}, tid')$ and outputs (failed, tid). If R_{cm} outputs $(\mathsf{received}, m, tid')$ then R outputs $(\mathsf{received}, m, tid)$.

If (wakeup, tid) is received by a machine, the machine inputs (wakeup, tid') to the corresponding machine of the underlying certified mail scheme. If a machine of the underlying scheme outputs (failed, tid'), the scheme outputs (failed, tid).

Recovery with $\langle \mathsf{T}, \mathsf{T}_{\mathsf{cm}}\rangle$: The first recovery message from R_{cm} sent to T is only input to T_{cm} if a correct pair (m_S, m_R) of signed messages with matching labels $l_S = l_R$ is appended. Else, all subsequent recovery messages from R are ignored.

Answers to recovery requests from $\mathsf{S_{cm}}$ are augmented with m_R if $\mathsf{R_{cm}}$ has sent recovery messages for this tid before. Else, T executes recovery of $\mathsf{S_{cm}}$ without changes. S accepts messages from T with and without m_R, while it ignores messages from R until it receives m_R.

Verification of a Receipt (Protocol "show"*):* On input (show, tid), S inputs (show, tid') to $\mathsf{S_{cm}}$ and sends m_R to the verifier.

The verifier V outputs (received, S, R, l, m, tid) if the underlying scheme outputs (received, S, R, ϵ, m, tid') and the sender is able to present an $m_R = \mathrm{sign}_{\mathsf{R}}(l, tid')$. Else, it outputs (failed, tid).

◇

Lemma 4.5 (Security of Scheme 4.5)
Scheme 4.5 is a secure optimistic labeled certified mail scheme (Def. 4.3). □

Proof. The scheme adheres to Definition 4.1 by construction. We now show that each of the requirements described in Definitions 4.2 and 4.3 is fulfilled:

Correct Execution (R. 4.2a): If (send, R, l, m, tid) is input to S and (receive, S, l, tid) is input to R with identical labels, then the messages m_S and m_R are appended to m_{1S} and m_{2R}, respectively, and both include the same label. Therefore, the protocol continues without changes. From [R. 4.2a] follows that the protocol then produces the correct outputs.

If the recipient outputs (received, m, tid), then it received (m_{1S}, m_S). A correct sender only sends m_S upon an input (send, R, l, m', tid) for some m' (note that also R is implicitly signed in m_S). Moreover, $\mathsf{R_{cm}}$ must have output (received, m, tid') by construction. Thus m was an input to $\mathsf{S_{cm}}$ for tid' [R. 4.2a] and therefore $m = m'$.

Unforgeability of Receipts (R. 4.2b+R. 4.3b) and Receipts are Fixed (R. 4.2c+R. 4.3b): See proof of Lemma 4.4. (Literally, except that the fact "$\mathsf{R_{cm}}$ was not terminated" is now clear anyway.)

Verifiability of Valid Receipts (R. 4.2d+R. 4.3b): See proof of Lemma 4.4.

No Surprises for the Recipient (R. 4.2e): If a correct R output (failed, tid) because the underlying $\mathsf{R_{cm}}$ output (failed, tid') then the sender cannot convince the verifier [R. 4.2e]. The only other possibility for R to output (failed, tid) is before making any input (receive, S, ϵ, tid') to $\mathsf{R_{cm}}$. Then unforgeability [R. 4.2b] implies that the sender cannot convince the verifier.

Secrecy of Message (R. 4.2f): See proof of Lemma 4.4.

Termination on Asynchronous Network (R. 4.2h): This follows immediately from [R. 4.2h] of the underlying scheme.

Optimistic on Asynchronous Network (R. 4.3d): If both input the same label and tid and none of the players receives wakeup, the scheme does not input wakeup to the underlying certified mail scheme which, in this case, is optimistic too [R. 4.3d].

■

Scheme 4.6 (Labeled Certified Mail from Certified Mail (3 of 3))
Let $(\mathsf{S}_{\mathsf{cm}}, \mathsf{R}_{\mathsf{cm}}, \mathsf{V}_{\mathsf{cm}})$, T_{cm} be an optimistic certified mail scheme with one initial message m_{1R} sent by the recipient R_{cm}.

We define a labeled certified mail scheme $(\langle \mathsf{S}, \mathsf{S}_{\mathsf{cm}}\rangle, \langle \mathsf{R}, \mathsf{R}_{\mathsf{cm}}\rangle, \langle \mathsf{V}, \mathsf{V}_{\mathsf{cm}}\rangle)$ and T_{cm}. If not mentioned otherwise, all simulation machines forward any message output by their sub-machine to the intended recipient sub-machine. The behavior of the simulation machines S, R, and V is defined as follows:

Sending a Message (Protocol “send“*):* On input (receive, S, l_R, tid) the recipient inputs (receive, S, ϵ, tid') to R_{cm} and sends (m_{1R}, m_R) with $m_R := \text{sign}_{\mathsf{R}}(l_R, tid')$ and $tid' := (\text{“cm2lcmR”}, tid)$ instead of m_{1R} to S. If the recipient receives an abort message m'_{2S} from the sender, it inputs (wakeup, tid') to R_{cm}. If machine R_{cm} outputs (received, (m, m_R), tid') with a correct m_R, the recipient R outputs (received, m, tid). If m_R is not as expected, it outputs (failed, tid).

On input (send, R, l_S, m, tid) the sender waits for (m_{1R}, m_R) and verifies that $m_R = \text{sign}_{\mathsf{R}}(l_R, tid')$ and $l_R = l_S$. If this is the case, the sender inputs (send, R, ϵ, (m, m_R), tid') and m_{1R} to S_{cm} and executes the protocol without changes. Else, if the labels are different or if wakeup is received before receiving m_{1R}, the sender sends $m'_{2S} := \text{sign}_{\mathsf{S}}(\text{failed}, tid')$ and outputs (failed, tid). If machine S_{cm} outputs (sent, tid') the sender S outputs (sent, tid).

If (wakeup, tid) is received by a machine, the machine inputs (wakeup, tid') to the corresponding machine of the underlying certified mail scheme. If a machine of the underlying scheme outputs (failed, tid'), the scheme outputs (failed, tid).

Recovery with T_{cm}*:* Recovery of the underlying protocol is not modified.

Verification of a Receipt (Protocol “show“*):* On input (show, tid), S inputs (show, tid') to S_{cm}.

The verifier V outputs (received, S, R, l, m, tid) if the underlying scheme outputs (received, S, R, ϵ, $(m, \text{sign}_{\mathsf{R}}(l, tid'))$, tid'). Else, it outputs (failed, tid).

◇

Lemma 4.6 (Security of Scheme 4.6)
Scheme 4.6 is a secure optimistic labeled certified mail scheme (Def. 4.3). □

Proof. The scheme adheres to Definition 4.1 by construction. We now show that each of the requirements described in Definitions 4.2 and 4.3 is fulfilled:

Correct Execution (R. 4.2a): If (send, R, l, m, tid) is input to S and (receive, S, l, tid) is input to R with identical labels, then the message m_R is appended to m_{1R} and the protocol is executed without changes. From [R. 4.2a] follows that the protocol then produces the correct outputs.

If the recipient outputs (received, m, tid), then R_{cm} output (received, (m, m_R), tid'). From [R. 4.2a] of the underlying scheme follows that (send, R, ϵ, (m, m_R), tid') was input to S_{cm}. By construction, this input is only made by S upon input of (send, R, l, m, tid).

Unforgeability of Receipts (R. 4.2b+R. 4.3b) and Receipts are Fixed (R. 4.2c+R. 4.3b): If the verifier V outputs (received, S, R, l, m, tid) then V_{cm} has output (received, S, R, ϵ, (m, m_R), tid') with $m_R = \text{sign}_{\mathsf{R}}(l, tid')$. Such an m_R was only signed if R got an input (receive, S, l, tid). (Recall that S is tacitly assumed to be signed in m_R.) This implies unforgeability.

If R output (received, m', tid) on this input, R_{cm} must have output (received, (m', m_R), tid'). From [R. 4.2c+R. 4.3b] follows that m' equals m in the output of V_{cm} and thus of V.

Verifiability of Valid Receipts (R. 4.2d+R. 4.3b): If a correct sender output (sent, tid) then the underlying scheme output (sent, tid') for the message (m, m_R). From [R. 4.2d+R. 4.3b] follows that on input (show, tid') to S_{cm}, the verifier V_{cm} will output (received, S, R, ϵ, $(m, \text{sign}_{\mathsf{R}}(l, tid'))$, tid') which leads to an output (received, S, R, l, m, tid) of V.

No Surprises for the Recipient (R. 4.2e): If a correct R output (failed, tid) on input (receive, S, l, tid), then R_{cm} either output (failed, tid') on input (receive, S, ϵ, tid'), or it output (received, (m, m_R^*), tid') with a wrong m_R^*.

In the first case, the verifier V_{cm} does not output (received, S, R, ϵ, m', tid') for any m' [R. 4.2e], and V will not output (received, S, R, l, m, tid) by construction.

In the second case, the verifier V_{cm} will only output (received, S, R, $\epsilon, (m, m_R^*), tid'$) for the same m_R^* [R. 4.2c]. If $m_R^* \neq \text{sign}_{\mathsf{R}}(l^*, tid')$ for any l^* then the verifier V does not output received. Otherwise, $l^* = l$ would hold because R only signs one m_R for this tid'. But this contradicts the assumption of this second case.

Secrecy of the Message (R. 4.2f): If the sender outputs (failed, tid), then either the underlying certified mail protocol output (failed, tid') and guarantees secrecy of the message [R. 4.2f] or else, failed was output after receiving m_{1R}. In this case the message is never input to the underlying scheme. Since the sender does not send the message in any simulation message, the message is kept secret.

Termination on Asynchronous Network (R. 4.2h): This follows from [R. 4.2h] of the underlying scheme.

Optimistic on Asynchronous Network (R. 4.3d): If both input the same label and tid and none of the players receives wakeup, the scheme does not input wakeup to the underlying certified mail scheme which, in this case, is optimistic too [R. 4.3d].

■

4.6 Simple Certified Mail

We now present a variant of certified mail which is considerably more efficient than labeled certified mail. We show that one is able to provide non-repudiation with 2 messages in time 2 if the recipient is not allowed to reject to participate in individual runs of the non-repudiation protocol.[11]

4.6.1 Definition

The basic difference between labeled and simple certified mail is that simple certified mail neither includes labels nor transaction identifiers, i.e., the recipient cannot decide whether to participate in a particular run (identified by the tid) or not.

Definition 4.6 (Simple Certified Mail)
A *simple certified mail scheme* for a message space M is a triple $(\mathsf{S}, \mathsf{R}, \mathsf{V})$ of probabilistic interactive machines (such as probabilistic interactive Turing Machines) where V does not keep state between subsequent protocol runs. The algorithm S is called sender whereas the algorithm R is called recipient. In addition, the scheme may specify a set AM of auxillary machines without in- or outputs. The algorithms can carry out two interactive protocols:

Sending Mail (Protocol "send"*):* The sender obtains a local input (send, R, m), where send indicates that the "send"-protocol shall be executed, R is the intended recipient, and $m \in M$ is the message to be sent. After the protocol execution, the sender either gets an output (sent, R, m) or (failed, R, m) whereas the recipient may output (received, S, m).

Verification (Protocol "show"*):* This is the non-repudiation of receipt verification protocol between the sender S and a particular verifier V[12]. The sender inputs (show, R, m) and the verifier outputs either (received, S, R, m) or (failed, R, m).

It is called secure if it fulfills the following requirements if all machines in AM are correct:

Requirement 4.6a (Correct Execution): Consider an execution of "send" by a correct sender S and recipient R on input (send, R, m) to S. Then, if the sender does not input (wakeup, R, m) the "send"-protocol outputs (sent, R, m) to the sender and (received, S, m) to the recipient.

Requirement 4.6b (Verifiability of Valid Receipts): If a correct sender S outputs (sent, R, m) on input (send, R, m) and later executes "show" on input (show, R, m) with a particular correct verifier V then V will output (received, S, R, m).

Requirement 4.6c (Secrecy of Message): If a correct sender outputs (failed, R, m) on input (send, R, m) then the recipient will not be able to obtain information about the message m.

[11] Note that this restriction corresponds to a german service of registered mail where a mail is defined to be delivered even if the recipient does not pick up his mail within a certain time.

[12] Note that if the recipient would be allowed to participate, a protocol sending the message during "show" would be a correct scheme.

Requirement 4.6d (No Surprises for the Recipient): If a correct verifier outputs $(\mathsf{received}, \mathsf{S}, \mathsf{R}, m)$ then a correct recipient R outputs $(\mathsf{received}, \mathsf{S}, m)$[13].

Requirement 4.6e (Termination on Synchronous Network): A correct sender will either output $(\mathsf{sent}, \mathsf{R}, m)$ or $(\mathsf{failed}, \mathsf{R}, m)$ after a fixed number of rounds.

Requirement 4.6f (Termination on Asynchronous Network): On input $(\mathsf{wakeup}, \mathsf{R}, m)$, a correct sender will either output $(\mathsf{sent}, \mathsf{R}, m)$ or $(\mathsf{failed}, \mathsf{R}, m)$ after a fixed time.

◇

We again define an optimistic simple certified mail by introducing a third party T that participates in case of exceptions:

Definition 4.7 (Optimistic Simple Certified Mail)
A secure simple certified mail protocol with $AM := \{\mathsf{T}\}$ is called "optimistic" if for correct R and a correct S who does not input $(\mathsf{wakeup}, \mathsf{R}, m)$, the "send"-protocol terminates after a fixed time and the third party does not send or receive messages. ◇

4.6.2 Protocol with Optimal Efficiency

We now sketch a simple certified mail scheme which is time- as well as message-optimal. Figure 4.5 depicts the optimistic behavior whereas Figure 4.6 depicts the detailed behavior of the machines.

Scheme 4.7 (Optimal Simple Certified Mail Scheme)
The scheme for simple certified mail for a message space M is a triple (S, R, V) and a third party T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Sending Mail (Protocol "send"; Figure 4.6): On input $(\mathsf{send}, \mathsf{R}, m)$, the sender sends $m_1 := \mathrm{sign}_{\mathsf{S}}(m)$ to R. The recipient then sends $m_2 := \mathrm{sign}_{\mathsf{R}}(m_1)$ to the sender and outputs $(\mathsf{received}, \mathsf{S}, m)$. Upon receipt of m_2, the sender outputs $(\mathsf{sent}, \mathsf{R}, m)$.

On input $(\mathsf{wakeup}, \mathsf{R}, m)$, the sender sends $m_3 := \mathrm{sign}_{\mathsf{S}}(\mathsf{R}, m)$ to T who sends $m_{4R} := m_3$ to R and $m_{4S} := \mathrm{sign}_{\mathsf{T}}(m_3)$ to S who then outputs $(\mathsf{sent}, \mathsf{R}, m)$.[14]

Verification (Protocol "show"): On input $(\mathsf{show}, \mathsf{R}, m)$ to S, S sends m_2 or m_{4S}, respectively. The verifier outputs $(\mathsf{received}, \mathsf{S}, \mathsf{R}, m)$ if the verification of the signatures was successful and $(\mathsf{failed}, \mathsf{R}, m)$, else.

◇

[13] Since the recipient must not participate in "show", this output is required to occur during "send".

[14] Note that an incorrect S may send a message $m' \neq m$, which may lead to a second receipt. This, however, does not contradict the requirements since the receiver and the third party will only produce receipts for messages that are received by the recipient.

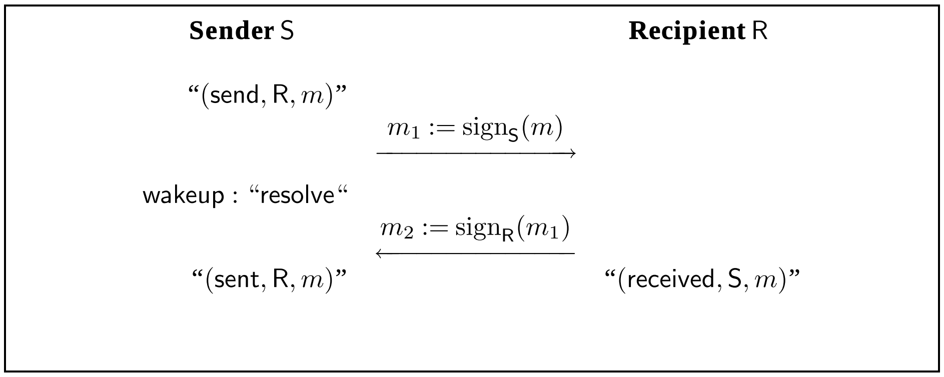


Figure 4.5: Optimistic Behavior of the Simple Certified Mail Scheme 4.7

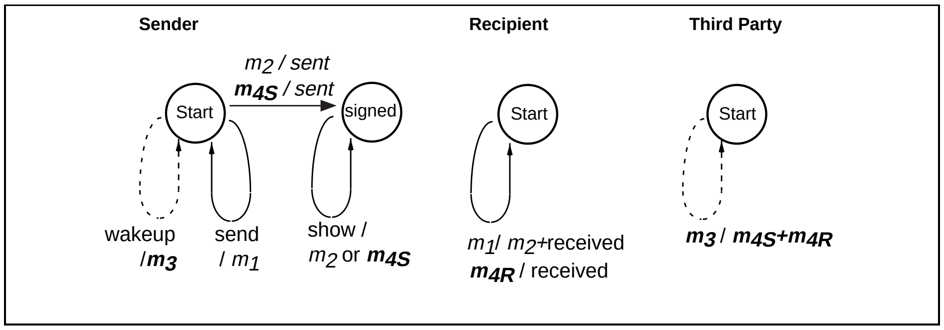


Figure 4.6: Machines of Scheme 4.7.

Lemma 4.7 (Security of Scheme 4.7)
Scheme 4.7 is a secure simple certified mail scheme. □

Proof. The scheme adheres to the protocols and their in- and outputs by definition. We now show that the requirements of Definition 4.6 and 4.7 are fulfilled:

Correct Execution: A correct receiver outputs (received, S, m) after receiving m_1 and sending m_2. After receiving m_2, a correct sender outputs (sent, R, m).

Verifiability of Valid Receipts: If a sender output (sent, R, m), it has received m_2 or m_{4S}, which are both accepted as a receipt by the verifier.

Secrecy of Message: A correct sender never outputs failed.

No Surprises for the Recipient: If a correct verifier outputs received, R either sent m_2 and obtained m or eventually obtains m as part of m_{4R}.

Termination on Asynchronous Network: On input (wakeup, R, m), the sender outputs (sent, R, m) after time 2.

Optimism: If both players are correct and the sender does not input (wakeup, R, m), then m_1 and m_2 are sent and the “send“-protocol terminates without T sending or receiving messages.

■

It is obvious that the following theorem holds:

Theorem 4.4 (Optimality of Scheme 4.7)
Scheme 4.7 is message- as well as round-optimal in the optimistic case.

This holds even if the third party is allowed to participate in the “show“-*protocol.* □

Proof. If the optimistic protocol would have only one round or one message, the receipt would be independent of the message. Therefore, the sender may convince a verifier that it sent any message. ■

Chapter 5

Conclusion and Outlook

In this part, we have analyzed the efficiency of two particular instances of optimistic fair exchange, namely contract signing and certified mail.

With respect to the optimal efficiency, we focused on the (hopefully) most likely case where both participants are correct. In this case, the efficiency of our *optimistic* protocols is better than non-optimistic fair exchange (cf. [PfSW1 98]). Therefore, *optimistic* protocols should be chosen for most applications if a high percentage of faulty protocol executions seems unlikely[1]. Another reason for using optimistic protocols is that they provide a higher level of availability, and more secrecy against the third party.

Furthermore, we have shown by means of mutual simulation that the message-complexity of both instances of fair exchange in case of two-party disputes is identical for optimism on disagreement. Obviously, our schemes resulting from several simulations are not efficient with respect to computation and memory. However, optimizing particular protocols is easily possible without additional messages or time.

A preliminary analysis of the time-complexity of time-optimal certified mail has shown that in case of optimism only on agreement, optimistic labeled certified mail needs more time than optimistic contract signing. For optimism on disagreement optimistic labeled certified mail needs as much time as contract signing. However, formal proofs of these observations are still in progress.

Even though we published some results towards multi-party fair exchange [ABSW 98, ABSW 99, AsSW2 96], the optimal efficiency of multi-party fair exchange under different assumptions is still unclear in most cases. In [ABSW 99], we have sketched how to apply multi-party contract signing to implement the secure and publicly verifiable multi-party commit of an atomic transaction. It would be interesting to further extend this approach towards a general notion of optimistically secure atomic transactions between mutually distrusting parties.

[1]Since a cheating party does not gain anything, this should be the most likely case.

Part II

Security of Reactive Systems

Chapter 6

Introduction

Most practically relevant cryptographic systems are *reactive:* For instance, in electronic cash systems users *interact* with the system by withdrawing, spending, receiving and depositing e-cash, and at any point in time the system has to *remember* the amount each party owns. Following the terminology in distributed systems we call any system that can interact with its environment multiple times and needs to keep state between two interactions reactive.

Some systems are obviously reactive, like electronic payments, but also secret-key agreement, public-key certification, or certified mail. At first glance more surprising examples are digital signatures and message encryption secure against active attacks: such attacks are sequences of interactions between the adversary and the system, and thus these systems are naturally defined reactively.

We present a computational model for reactive systems, and investigate how to define the security of such systems following the *simulation paradigm:* Security of a system is defined by means of an ideal system that has all the desired security properties by construction, but usually makes unrealistic assumptions, e.g., that a machine unconditionally trusted by all parties is available (called "trusted host"). A real system is secure if it is "at least as secure as" this ideal system: Anything that can be achieved by an adversary attacking the real system can be achieved as well, i.e., simulated, by an adversary attacking the ideal system.

6.1 Our Results

Our main two contributions are to show how the simulation paradigm can be applied to reactive and real-world systems. Previous work either considered only practical protocols for specific types of systems, or focused on non-reactive evaluation of functions.

One of the new aspects is that we model the users explicitly as arbitrary stateful machines. These users use the system as well as its simulation. This is particularly natural if one reactive system is used as a sub-system of another one while keeping its own state.

Our model compares the security (i.e., integrity and secrecy) of any two systems. We then use the concept of *trusted hosts* for specifying the intended service of a system. A trusted host is, essentially, a single and non-corruptible machine, which provides the desired service for all parties. A reactive system is secure if it is at least as secure as the trusted host specifying its service.

As we will show in Section 6.2, applying the concept of trusted hosts in its original sense (i.e., only one machine that offers the same service to attackers and honest users) leads to the problem that in many cases, no efficient system can fulfill such a specification. Nevertheless, one may want to use an efficient system and one should be able to define and prove precisely what security such a system offers. We solve this problem by the new concept of "real-world" trusted hosts: First, such a trusted host differs from a normal one by offering specific services to the adversary, which define the accepted vulnerabilities of the system in an abstract way. Secondly, it is then useful to accept that there is not one, but a collection of trusted hosts in one specification.

6.2 "Real-world" Trusted Hosts Specifying Accepted Vulnerabilities

To illustrate why we need specific "real-world" trusted hosts offering special services to adversaries, we now sketch some common vulnerabilities that are accepted for efficiency reasons in many situations, and why they cannot be modeled in a usual one-for-all trusted host.

6.2.1 Accepted Secrecy Leaks

As a first example, consider encryption. For instance, in [Gold 99], Page 8, Goldreich says that an encryption scheme is considered secure if it simulates an ideal secret channel between the parties, and that this means that an eavesdropping adversary gains nothing over a user which does not tap this channel. This is indeed the most desirable service and easily specified by a trusted host that simply relays messages. However, it implies that an adversary does not even see whether a message was sent or not. Thus any real system as secure as this trusted host must hide this fact, too, e.g., by sending meaningless ciphertexts at all other times (see [Abad 98]). This may indeed be the way to go in a few applications, but it costs a lot of bandwidth. In many applications, one is therefore satisfied with encrypting the real messages only. Nevertheless, one may want to have a precise definition of what one has achieved in the form of the simulation of a trusted host. For this, we introduce an abstraction of the accepted vulnerability into the trusted host. For encryption, the trusted host will tell the adversaries the traffic pattern in the form of one "busy" bit per pair of honest participants and round (see Section 9.1).

The same situation with observable traffic patterns occurs in many other systems. Our approach enables us to separate cryptographic vulnerabilities of insecure implementations from the unavoidable vulnerability of the service given certain ressource limitations.

6.2.2 Multi-Round-Rushing Adversaries

While, once noticed, the traffic analysis vulnerability may seem clear, and so does a similar vulnerability that the adversary can usually destroy messages in transit, it may be less obvious that even after these changes to a trusted host, no real implementation will be as secure as it. (The same problem occurs with the ideal model of encryption, i.e., it is not a result of introducing vulnerabilities at all.)

The problem is that adversaries can react several rounds faster than honest participants.[1] We first present it in the synchronous model and then discuss why it is not only a technical difficulty of this model.

The synchronous model is that all correct machines switch once per round, and their outputs are available as inputs to other machines in the next round. Thus, if an honest user sends a message using the real encryption system, the message goes via the sender’s machine and the recipient’s machine. The corresponding timing would be modeled in the trusted host, i.e., the trusted host would deliver messages after two switching steps. Thus, with the trusted host, an honest user cannot get an answer referring to his message earlier than four rounds after his message. However, if the recipient in the real system is corrupted, he can save the two rounds where his own machine would handle the message by taking it immediately from the line, decrypting it (he knows his own decryption key), and composing and encrypting the answer, all in one round. Thus an answer from a dishonest recipient can arrive two rounds faster. In this case, we see no realistic way at all to improve the real system so that it corresponds to the same trusted host for honest and dishonest recipients.

We therefore also model this accepted vulnerability in the trusted host, so that anyone basing an application on the trusted host must be aware of it. For this, we distinguish access points of honest users and adversaries to the trusted host. The service offered to the adversary is faster, corresponding to the minimal timing the adversary can achieve in real systems.

It is reasonable to ask whether this problem would disappear in asynchronous models, and whether asynchronous models aren’t more realistic anyway. (A similar approach would be to relax the timing requirements in the comparison, i.e., not to require that events in the ideal and the real system must happen in the same rounds.) First, however, many cryptographic protocols are designed for synchronous systems and it should be possible to define their security.[2] Secondly, in real life the problem occurs whenever the users (not the machines of the system!) may have access to real time. This seems unavoidable because we cannot define that cryptographic systems should never be used in real-time applications or that human users should not look at their watches. Thus, in a real system an adversary may indeed be faster (e.g., by bridging network delays), and honest users can notice this. This will usually not have a very bad effect,[3] but contradicts real-life indistinguishability of the real system and the trusted host. Hence we believe that this fact must be modeled in the trusted host.

[1]This is different from the well-known problem of rushing adversaries within a round, which is described in Section 8.1.6.

[2]Of course, the synchrony is typically virtual, i.e., derived from loosely synchronized clocks and bounds on message delays after which a message will be considered lost.

[3]However, one can construct examples where users tell secrets to someone who can react very fast, e.g., because they believe that the person must already have known the secrets before.

6.3 Overview

Chapter 7 discusses related work.

Chapter 8 presents our formal model in detail. A reactive system is basically a set of synchronously interacting probabilistic state machines. “At least as secure as” means that for each static adversary in the real reactive system there is an adversary in the ideal reactive system that achieves essentially the same effect for each honest user. This means that the adversary attacking an ideal system can computationally or perfectly simulate the (reactive) behaviour of the real system towards the honest users.

Chapter 9 shows how to apply our model to practical reactive systems. The first example (Section 9.1) is *secure channels*, i.e., secret and authenticated message transmission. As we will see, already this seemingly trivial example exhibits serious definitional problems. The second example (Section 9.2) is *optimistic labeled certified mail*. This example illustrates our alternative to the traditional style of definition as used in Section 4.2. In both cases we define trusted hosts, present a real system, and prove the security.

Chapter 7

Related Work

Early research (e.g., [GoMi 84, Yao1 82]) formalized the security of non-reactive cryptographic primitives against passive attacks. For completeness, we sketch some of these early results but then focus on definitions based on the simulatability paradigm. Early sketches of general security definitions for reactive systems were described in [PfWa 94, GMW 87]. In [PfWa 94], a system was defined to be secure if for any adversary and user-machine behavior, there exists a behavior of a virtual user accessing the corrupted ports of the trusted host such that the behavior to the correct users is indistinguishable. The environment was already modeled explicitly by introducing user-machines.[1]

7.1 Direct Cryptographic Security Definitions

Simulatability is not the main way to define cryptographic security: For many systems the desired properties can be defined directly, without comparing the real system with some ideal system.

In general, all *integrity* properties can be directly defined by requiring that an adversary cannot destroy the desired properties (cf. Section 3.2.4). Unfortunately, *secrecy* properties are more difficult to capture with such definitions [GoMi 84][2] since they require a more detailed model of the attack and a clear definition of the secret that shall be guessed (cf. Section 4.2.2), i.e., a new secrecy definition is required for each new kind of system. This hampers an evolution of a well-understood security formalism in parallel to the evolution of new services.

This is probably the reason why for multi-party computations trusted host definitions are the dominant way of defining security. Note that a trusted host necessarily defines both, secrecy and integrity.

A disadvantage of simulatability-based definitions as compared to direct ones is that a trusted host usually over-specifies the intended service: It fixes all details *completely* and fixes also details that are not relevant for security. For instance, a trusted host may fix the exact timing of a protocol even though it would not make any difference for security.

[1]The concept of user-machines was also described in [PfWa3 98].

[2]In [GoMi 84], an encryption system was defined to be semantically secure if any output of an adversary given the ciphertext can also be computed without it.

7.2 General but Non-reactive Simulatability Definitions

The first definition of secure multi-party computation of a function was proposed in [Yao 82]. It defines that a protocol provides secrecy if each player does not gain knowledge about the other inputs to the function, except the computed value. Integrity is defined as being able to detect cheating players. This definition does not require that the inputs are kept secret in case a player is cheating. This was fixed in later definitions of secure function evaluation by defining secrecy including integrity [Beav5 91, Cane 96, GoLe 91, MiRo 92, Yao 82]. In [GoLe 91], for example, the simulation paradigm was used for defining secrecy and integrity: A computation including faulty players was defined to be secure if any protocol execution including faulty players can be simulated by a 'legal version" of the protocol. In such a legal version, the faulty players first compute their inputs, then perform all computations correctly, and finally make an output that is computed from their view. This is similar to the definition of non-adaptive secure computation in [Cane 96]. In order to introduce adaptiveness, [Cane 96] then allows the adversary to input corruption requests in the legal version of the protocol.

In [GoMR 89], a similar approach has been used to define the notion of zero-knowledge: In a zero-knowledge proof, a prover P proves the fact that $x \in L$ for a language L to a verifier V. Intuitively, secrecy means that the verifier learns nothing except the fact that $x \in L$. This is formalized by the requirement that any dialogue between a prover P and an incorrect verifier V' can be simulated by a simulator $\mathsf{M}(\mathsf{V}')$.

7.3 Reactive but Non-General Simulatability Definitions

We now sketch some specific reactive security definitions that follow a simulatability approach comparable to ours.

In [GeMi 95], the security of verifiable secret sharing was defined based on existing simulatability definitions for multi-party computation of a function. This definition splits the reactive system into a sharing and a recovery phase. Each of these phases is then specified by a function. A scheme is defined to be a secure verifiable secret-sharing scheme if it securely computes these two functions.

[BeCK1 98] define the security of authentication and key exchange protocols. They compare an adversary attacking a protocol on an idealized authenticated network model (called AM) with an adversary attacking a real-world protocol on an unauthenticated network (called UM).

A so-called authenticator converts an idealized non-authenticated protocol for a authenticated network model AM into an authenticated protocol for an unauthenticated network UM. It is defined as being secure if for any adversary for the unauthenticated network there exists an adversary for the authenticated network, such that the output distributions for any given input vector are computationally indistinguishable.

Next they analyze key exchange protocols for implementing authenticators now directly comparing the behavior of an idealized trusted host[3] and a real protocol. In both models the machines output the secret keys, but the adversary does not obtain any information about them (i.e., there are two different kinds of outputs) unless he corrupts a user. The adversary is scheduling everything.

The weakness that the adversary does not obtain any information about the keys is observed in [Shou 99]. The basic model for key exchange is the same as in [BeCK1 98] but more explicit (i.e., the trusted host is really defined as a machine). The user revealing (partial) information is modeled by letting the adversary choose functions f which can be applied to the global state (which includes all secret keys). The adversary then obtains the output of the chosen function. The collection of all functions f corresponds to our user machine H modeling the users of the system.

Note that [BeCK1 98, Shou 99] do not consider users as stateful machines but rather quantify over the input sequences of correct users. This is sufficient for the analysis of authentication protocols but does not extend to secrecy of other protocols.

Remark 7.1. In [Pfit8 96] such user-machines were generally defined, but only in the context of a cryptographic semantics for temporal-logic formulas, i.e., integrity requirements. However, there it was easy to prove that the user machines could be joined with the adversary machine. Intuitively this means that the honest users are fully controlled by the adversary. But in general, i.e., with secrecy requirements, this is not possible: if A comprises the honest users, there are no "secrets" that could be protected – A would know everything the honest users would know. ○

Another definition of a particular reactive system is [Beav 96] that defines the security of oblivious transfer against adaptive adversaries, i.e., adversaries that may decide to corrupt any player while the protocol is running. This is done by allowing the adversary to send "corrupt requests" to the players at any time.

7.4 Reactive and General Simulatability Definitions

An early reactive specification was the notion of "mental games" as sketched in [GMW 87]. It specifies the game to be played by means of a single turing-machine and then requires that the (distributed) game produces the same outputs and that each view of a player can be simulated using its inputs and the outcome of the game.

One way to apply the simulation paradigm to the definition of secure reactive systems that has been proposed is to consider a reactive system as the sequential composition of individual function applications. Each distributed function evaluation can then be compared to the idealized function using the existing notion of simulatability for multi-party computation of a function (see Section 7.2).

[3]The described trusted host defines commands like "establish key", "corrupt session", or "corrupt player" that can be invoked by the adversary.

Indeed, often one can identify individual sub-protocols (sending one message, making one payment, withdrawing money, etc.), and one could hope that they play the role of these functions. However, both the system and its users may keep state between different executions, e.g., if the system is a secure database the results of a "read" operation clearly depends on earlier "write" operations. In particular with the system, this is much better modeled if one also defines the ideal system in a reactive way (i.e., with an "ideal state") and defines the comparison of reactive systems.

For instance, in an electronic cash system the success of an attempted payment depends on the fact whether a party has enough "electronic coins." These coins in their cryptographic form should certainly not be an output of the withdrawal function in the ideal system used for the definition because they are implementation-dependent. Nor can we simply let the ideal system output something like "you now have seven coins" in withdrawal and let the user input this again in payment because an adversary might input an arbitrary number. Instead, the ideal system should internally have a state that contains the correct number of coins any party would currently have.

A quite detailed definition for reactive systems using this approach is given in [HiMa 97, FiHM 99]: They define the system by straight-line programs, with inputs and outputs (but without loops). A protocol is secure iff for any adversary A corrupting a subset of the players of the protocol, there exists an ideal-model adversary A_0 and a fixed subset of the machine outputs of the protocol, such that the distribution of the subset of the outputs of the correct machines (excluding T) in the protocol is identical to the distribution of all outputs of the correct machines in the ideal model. The definitions are only given for the information-theoretic model which simplifies the definition of user behavior but also results in relatively complicated definitions for the views that must be simulatable. The definitions do not easily generalize to the computational case (later, for each concept, we will explain why).

Another approach extending multi-party function evaluation is to consider a reactive system as a single state transition machine. Under the assumption that each party outputs its state after each round, the execution of such a machine can then be modeled as a sequence of multi-party computations of the state transition function [Gold 98].

An approach to specifying general reactive systems that is similar to ours was sketched in [Cane 98]. It is an extension of the model for secure function evaluation proposed in [Cane 96]. The basic idea of the extension is to introduce an environment that keeps state between subsequent invocations of the reactive system. In [CaGo 99], this approach was applied to threshold encryption where an arbitrary protocol π that uses the cryptosystem plays the role of this environment.

Chapter 8

Security of Reactive Systems

This section describes our formal notion of the security of reactive systems. In Section 8.1, we define the model of systems, adversaries, users, and ideal systems specifying a service based on trusted hosts. In Section 8.2, we define what it means that a system is "as secure as" another system. In Section 8.3, we prove some properties of our model.

8.1 Building Blocks of a Model of Reactive Systems

Intuitively, when a reactive system is running, there are some correct machines, adversaries, and honest users, i.e., an environment to which we want to guarantee a service, see Figure 8.1. We will call this a configuration. In the following subsections, we first briefly describe the machine model. Then we define correct machines and how a system prescribes them, adversaries and honest users, and how a configuration runs. In particular, we need a special switching model allowing for "rushing users". Finally, we define ideal systems, i.e., trusted hosts, in a way that allows them to contain accepted vulnerabilities.

8.1.1 Machines

All our machines, including adversaries and users, are modeled as probabilistic interactive state machines. In each switching step, such a machine reads its inputs and a random value (assumed to come from a random source private to this machine), changes its local state, and makes outputs. For ease of concrete

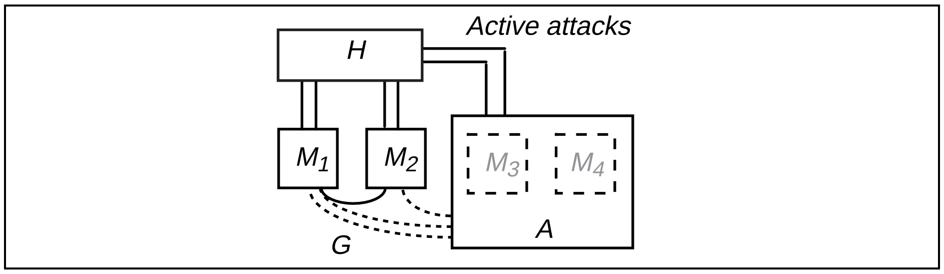


Figure 8.1: A Configuration of a Reactive System

specifications we allow machines to have several so-called input and output *ports*; there can be one input or output per port in each round. For each machine M, let ports(M) denote its ports (formally pairs of a name and a Boolean value “in” or “out”); similarly for sets of machines. Machines start with one “initial input”, i.e., the starting state is parameterized.

We call a machine *polynomial-time* if the time of its computation is polynomially bounded in the length of its initial inputs.[1] The typical computational model is interactive Turing machines as in [GoMR 89]; each port is modeled by a communication tape.[2]

8.1.2 Structures

A structure is a triple (M, G, s). The first component is a set $M = \{\mathsf{M}_1, \ldots, \mathsf{M}_\mathsf{n}\}$ of machines, called the *correct machines*. Intuitively, these are the machines that execute a prescribed protocol. We assume that both the machines and all their ports have a unique name within any structure.

The second component, G, is a graph called the *connection graph*. Its nodes are ports, let nodes(G) be the set of them. Intuitively, this graph contains both the connections between different machines of M and all the connections that these machines offer to their environment, i.e., honest users and adversaries. We require $\mathsf{nodes}(G) \supseteq \mathsf{ports}(M)$ and define $\mathsf{free}(G, M) = \mathsf{nodes}(G) - \mathsf{ports}(M)$. Each output port is connected to one or more input ports, and each input port is connected to exactly one output port. Hence G describes unidirectional multicast channels, and all ports are connected. Each multicast connection must contain at least one port from ports(M).

The third component, s, called *specified ports*, is an arbitrary subset of $\mathsf{free}(G, M)$. Formally, it will only play a role in the definition of “as secure as”. There the ports in s are intuitively those whose service must be guaranteed.

8.1.3 Systems

Typically, a system will be described by an “intended” structure (M, G, s) for the case without attacks, and the other allowed structures are derived from it given a so-called trust model for both machines and channels.

However, our model is not specific to any such derivation rules: We simply define that a *system* Sys is a set of structures (M, G, s).

Remark 8.7. Recall that we currently only model static adversaries. Hence each of our structures can contain a fixed set of correct machines. Otherwise we would have to provide specific ports where the adversary can make “corruption requests”. ◦

[1]The simpler definition that the state-transition function is polynomial-time is not sufficient. For instance, a polynomial-time machine could then always make an output twice as long as its latest input. Two interacting machines of this type would have exponential power.

[2]Actually, we are not aware of a precise formalization for a multi-party scenario (i.e., how the Turing machines that continually run on a bit-by-bit basis are synchronized or scheduled into global message-by-message communication), but this can certainly be done and is not specific to our paper. (It is also needed for multi-party function evaluation and any specific multi-party reactive systems with computational bounds.)

8.1.4 Honest Users

As mentioned in the introduction, a special feature of our model is that we model the honest users, i.e., the environment of the correct machines and the adversary, as normal probabilistic machines. The reason is that in the real world, the users' inputs will depend in some unknown way on previous in- and outputs. Moreover, real users may also communicate outside the given system; for instance, user X might give some outputs to user Y so that Y's input can depend on them, even if X makes no inputs himself. We therefore model the set of all honest users as a single probabilistic machine named H.

A compatible user for a structure (M, G, s) is an arbitrary machine H whose port names are disjoint from $\mathsf{ports}(M)$. Typically, $\mathsf{ports}(\mathsf{H}) \cap \mathsf{nodes}(G)$ is non-empty, i.e., the user uses some of the free ports offered by the structure. Intuitively, these ports are used for commands from the users to the system, such as "send message m to R" or "pay amount x to S", and to obtain the system's reaction, like "message m received" or "amount x paid".

Remark 8.8. Using an arbitrary machine H gives the strongest definition: an arbitrary probabilistic user machine represents the adversary's arbitrary a-priori knowledge about the behavior of the honest users. However, in computational security definitions, we will also restrict H to computationally feasible behavior.

Remark 8.9. We currently do not allow restrictions on honest users (such as "don't do this before doing that"), corresponding to the view that a system specification used in a simulateability definition should contain provisions for all possible user behaviors, e.g., error messages.

Remark 8.10. In a system with subprotocols like individual payments, our model automatically covers that subprotocols can be interleaved, i.e., that different users may make payments etc. at the same time. If one does not want this, it must be excluded on the system side, e.g., by ignoring inputs at certain times. ○

8.1.5 Adversaries

A compatible adversary for a structure (M, G, s) and user H is an arbitrary machine A whose ports are disjoint to those of M and H and that contains all ports of G that are still free, i.e., $\mathsf{ports}(\mathsf{A}) \cap \mathsf{nodes}(G) = \mathsf{free}(G, M) - \mathsf{ports}(\mathsf{H})$.

The remaining ports of H and A are for communication between the honest users and the adversary. Intuitively, these connections are used for active attacks. One example is that a user forwards outputs of a machine $\mathsf{M_i}$ to the adversary, e.g., the received plaintexts in a chosen-ciphertext attack on an encryption scheme. Another example is that a user forwards certain inputs from the adversary to a machine $\mathsf{M_i}$, as in a chosen-message attack on a signature scheme. We describe the connections by a graph G_{AH}, whose set of nodes $\mathsf{nodes}(G_{AH})$ is $\mathsf{ports}(\mathsf{H}) \cup \mathsf{ports}(\mathsf{A}) - \mathsf{nodes}(G)$. Like G, it may connect an output port with several input ports, while an input port is connected only once. Furthermore, connections from H to itself are not allowed. We call this a compatible graph.

A *configuration* is defined as a tuple $(M, G, s, \mathsf{H}, \mathsf{A}, G_{AH})$ where (M, G, s) is a structure, and H, A, and G_{AH} are a compatible user, adversary, and

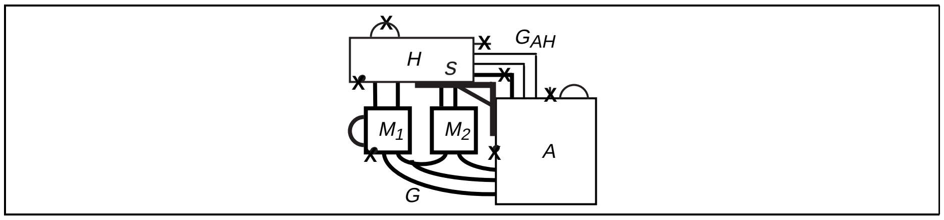


Figure 8.2: A Configuration. The structure is shown in bold lines. While Figure 8.1 showed a "normal" case, this figure shows some more exotic possibilities; those that we have excluded are crossed out with an X.

graph, see Figure 8.2.[3] We denote the set of all configurations for a given system Sys by $\mathsf{Conf}(Sys)$, and those with polynomial-time adversary and user by $\mathsf{Conf}_{\mathsf{poly}}(Sys)$. We omit "poly" if it is clear from the context.

Remark 8.11. The fact that we allow either all or all polynomial-time adversary machines in configurations guarantees that we can later freely modify and combine adversaries without risking that the new adversary is not valid.

Remark 8.12. We currently model a single adversary who co-ordinates all malicious behavior in the configuration. This results in the strongest security definitions. Introducing multiple adversaries only changes the model if we limit the communication and computation power of some of them: Otherwise they can always agree on a common strategy and behave like a single adversary. Hence a structure would contain precise interfaces for the different adversaries, and possibly classes of allowed machines for them. Multiple and non-cooperating adversaries of different power are considered in [HiMa 97, FiHM 99]. A practical example are wallet-with-observer payment systems [ChPe1 93]; they offer privacy for the payer only if a corrupt bank cannot communicate directly with a corrupt observer inside the payer's personal wallet. ◦

8.1.6 Dynamic Behaviour with Rushing Users

We now define the dynamic behaviour of a configuration, i.e., of connected correct machines, adversary and user.

All machines, including A, receive the same initial input, also called global input. For simplicity we assume that it is of the form 1^k, where $k \in \mathbb{N}$ is a security parameter.

As mentioned above, we currently use a synchronous model. Normally, i.e., in systems without specific adversaries and users, this means that the execution proceeds in *rounds*. In each round, all messages from the previous round are reliably transported from the output ports to the connected input ports. Then each machine switches (based on its current local state, all messages from its input ports, and its current random input), producing a new local state and messages for its output ports. (If an input port does not provide any message, the machine reads a special "empty" symbol ϵ.)

The well-known concept of "rushing adversaries" (the first references we are aware of are [BrDo 84] for the concept and [ChDw 89] for the name) can be

[3]Without loss of generality, one can make a similar definition with only one graph, but then one needs additional notation to retrieve the structure from the configuration.

described as follows: In a fully synchronous system, even the adversary would respect the round structure, i.e., consider only outputs of the previous round as inputs. However, this is not realistic in most implementations of rounds: A real adversary can "rush" and compute its outputs of a given round based on the outputs of the correct players of the *same* round.[4] Hence one switches adversaries and correct machines alternately in the model.[5]

We now have the same problem with the users: Even an honest human user might consider a "message" as soon as it pops up on his or her display.[6] Since we do not want to restrict the user behavior, we define "rushing users" in addition to "rushing adversaries". Furthermore, in real life, the adversary and the honest users may interact with each other several times in one round before producing their inputs for the correct machines. This would mean that in each round, we first switch the correct machines and then allow an arbitrary dialogue between the user machine H and the adversary A before they produce their inputs for the correct machines in the next round. Fortunately, as we show in Section 8.3.2, the following model with only four subrounds of each round $[i]$ is equivalent:

- In Subround $[i.1]$, the correct machines in M and the user H switch.
- In Subround $[i.2]$, the adversary A switches.
- In Subround $[i.3]$, the user H switches again.
- In Subround $[i.4]$, the adversary A switches again.

We can assume w.l.o.g. that H and A make outputs on connections where all recipients are in M only in Subrounds $[i.3]$ and $[i.4]$, respectively, because a machine $\mathsf{M_j}$ can only consider them in Round $[i + 1.1]$. Otherwise we define that $\mathsf{M_j}$ obtains the concatenation of the two outputs as an input. Given this switching model, the runs (or executions, or traces) of a configuration are well-defined. Hence for each global input (i.e., starting states), one obtains a probability space on such executions, given by all random inputs to all machines.

The *view* of a set M' of machines in a run consists of the global input and, for each round, of the random input and the list of all messages sent or received by the machines in M'. Such a view, like any other function of a run, is a random variable in the probability space of the runs. We call it $view_{conf,k}(M')$, where k is the initial input and $conf$ the configuration, and omit $conf$ if it is clear from the context. Let $view_{conf}(M')$ denote the ensemble of random variables $(view_{conf,k}(M'))_{k\in\mathbb{N}}$.

Remark 8.13. The subrounds also hide differences between correct and attacked connection graphs: Channels where the adversary can modify the messages are routed through the adversary. In a fully synchronous model, the messages

[4]If such an attack is not modeled, two players can send two random numbers r_1 and r_2 to each other and use $r_1 \oplus r_2$ as a secure distributed coin-flip (cf. [Blu1 83]). In practice, however, such a scheme is insecure since a dishonest player may send r_2 after learning r_1.

[5]Recall that this is another problem than the multi-round-rushing adversaries explained in Section 6.2.2, which will be covered by the specification of the trusted hosts.

[6]We could prevent this by letting the machines hold back user outputs until the beginning of the next round. However, the next problem occurs in any case.

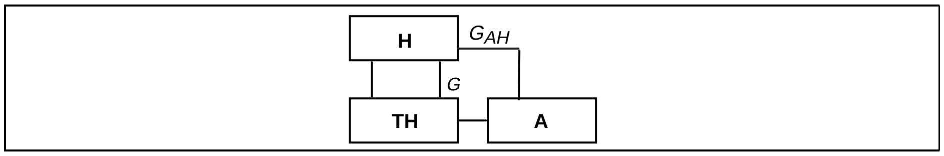


Figure 8.3: Configuration of an Ideal System

would be delayed as A switches them, while the uncorrupted channel would deliver them directly. With subrounds, every message arrives in the next full round in both cases.

○

Lemma 8.1
The following general facts are true about compositions of machines:

a) We can consider several machines of the same type (i.e., users, adversaries, or correct machines) as one machine without modifying the probability space of executions of the system.

b) Such a composition of polynomial-time machines is polynomial-time.

c) In the same sense, a user machine and correct machines can be combined into a user machine.

d) W.l.o.g. the view of a combined machine M^* *can be identified with the view of the set of individual machines* $\mathsf{M_i}$ *in it. This implies that we can also say w.l.o.g. that the view of each* $\mathsf{M_i}$ *is a part of the view of* M^**.*

□

Proof. The composition of several machines in Part a) means that the transition function of a composed machine is defined by letting internal machines and internal connections switch just like external ones. Then one can easily show associativity of such compositions. The number of steps of the composition is the sum of the steps of the two machines, plus an overhead for internal switching linear in the length of the messages written. All this is polynomial in k. Part c) is clear from Part a) because correct machines switch like a special case of user machines. For Part d), recall that the view of M^* consists of all the random inputs and all messages sent and received by M^*. From this, the local states and messages sent and received internally between machines $\mathsf{M_i}$ can be uniquely reconstructed. ■

8.1.7 Ideal Systems with Accepted Vulnerabilities

As we allow accepted vulnerabilities even in the ideal system (recall Section 6.2), our ideal systems are technically not special and we do not make a specific definition for them. Typically, an ideal system Sys contains only structures of the type $(\{TH\}, G, s)$ (see Figure 8.3), i.e., with only one correct machine. It is called *trusted host*. In other models, i.e., without accepted vulnerabilities, the trusted host has no special connections to the adversary, and one is not so flexible in allowing different trusted hosts in different structures.

8.2 Simulateability Definition of Secure Reactive Systems

We now define what it means that one system Sys_1 is "at least as secure as" another system Sys_2, abbreviated as $Sys_1 \geq_{sec} Sys_2$. Usually Sys_2 will be an ideal system and Sys_1 a real system supposed to implement the same service. An overview is given in Figure 8.4.

Roughly we want to make the following requirement: Whatever an adversary can achieve in a structure $(M_1, G_1, s_1) \in Sys_1$ with an honest user H, another adversary could achieve in a structure $(M_2, G_2, s_2) \in Sys_2$ with the same honest user. However, we must somehow fix which structures from Sys_2 are suitable for the same user. We therefore assume a "suitability" mapping f from Sys_1 to the powerset of Sys_2. Usually, this mapping is derived from an intended structure for each system and a trust model. However, in the general definition, f can be almost arbitrary.

We will now also use the sets s of specified ports given in the structures. Primarily, however, the distinction between the user interface and the adversary interface of a system is implicit in the definition of f: If structures (M_1, G_1, s_1) and $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ have free ports in common, and a user H uses them in the first system, the same ports of H are automatically connected to the corresponding ports of (M_2, G_2, s_2). Hence at these ports, the first structure must offer a service indistinguishable from the second one. However, H may have additional ports that occur in $\mathsf{free}(G_2, M_2)$, but not in $\mathsf{free}(G_1, M_1)$. If such a port is in s_2, i.e., a specified port, we will indeed connect H with (M_2, G_2, s_2) at this port, but otherwise we will not. In the latter case, we exclude this user machine H. For clarity, we also exclude the case $\mathsf{free}(G_1, M_1) \cap (\mathsf{free}(G_2, M_2) - s_2) \neq \emptyset$ in the mapping f, i.e., structures that are compared should not have the same name for ports that are not supposed to be compared.[7] See Figure 8.5 below for the general case.

The notion that the second adversary achieves the same results as the first is defined as follows: The views of H in the two configurations must be indistinguishable, see Figure 8.4.[8] We recall the definition of indistinguishability (in the form from [Gold1 95], first introduced in [BlMi 82, GoMi 82, Yao1 82]):

Definition 8.1 (Indistinguishability)
Two ensembles $(\mathsf{var}_k)_{k \in \mathbb{N}}$ and $(\mathsf{var}'_k)_{k \in \mathbb{N}}$ of random variables (or probability distributions) are called

- perfectly indistinguishable ("=") if they are identical,
- computationally indistinguishable ("$\approx_{poly}$") if for any probabilistic polynomial-time algorithm D (the distinguisher), the differences of the outcome of D on the two distributions are negligible (in k):

$$|P(D(\mathsf{var}_k, 1^k) = 1) - P(D(\mathsf{var}'_k, 1^k) = 1)| \leq \frac{1}{\mathsf{poly}(k)}.$$

[7]Now the omission of machines H with ports from $\mathsf{free}(G_2, M_2) - s_2$ is w.l.o.g.: We can take an identical H' except that these port get completely new names. As these ports are not in $\mathsf{free}(G_1, M_1)$, no corresponding renaming in (M_1, G_1, s_1) is necessary.

[8]Note that this includes the views of the adversary since it may communicate its view to H.

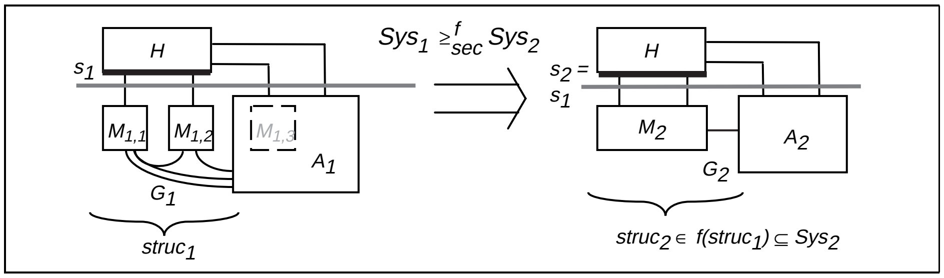
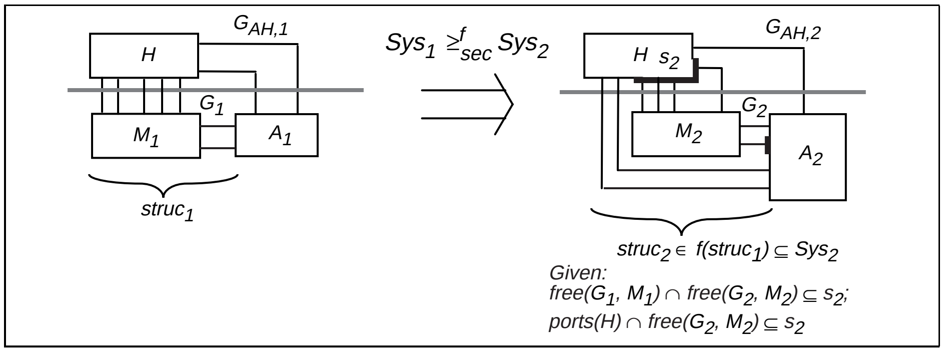


Figure 8.4: Simple Case of the Simulateability Definition for Reactive Systems. The grey line delimits the view of the honest users, which must be indistinguishable. The sets of specified ports on both sides are the same and precisely the interface between M and H.



$G_{AH,1}$
H
G_1
M_1
A_1
$struc_1$
$Sys_1 \geq^f_{sec} Sys_2$
$G_{AH,2}$
H s_2
G_2
M_2
A_2
$struc_2 \in f(struc_1) \subseteq Sys_2$
Given:
$free(G_1, M_1) \cap free(G_2, M_2) \subseteq s_2$;
$ports(H) \cap free(G_2, M_2) \subseteq s_2$

Figure 8.5: General Case of the Simulateability Definition for Reactive Systems.

Hence D gets an element chosen according either var_k or var'_k and the security parameter as input and makes a Boolean output. The notation $g(k) \leq 1/\mathsf{poly(k)}$ ("negligible") for a function g means that for all polynomials p, $\exists k_0 \forall k \geq k_0$: $g(k) \leq 1/p(k)$.

We write $\approx$ if we want to cover both cases. $\diamond$

Definition 8.2 ($Sys_1 \geq_{sec} Sys_2$)
Let Sys_1, Sys_2 be two systems, and let a mapping f from Sys_1 to the powerset of Sys_2 be given. We require that $\mathsf{free}(G_1, M_1) \cap \mathsf{free}(G_2, M_2) \subseteq s_2$ for all $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$, i.e., comparable structures only contain equal free port names if they are supposed to correspond to each other.

- We call Sys_1 **perfectly at least as secure as** Sys_2 with respect to f and write

$$Sys_1 \geq^{f,perf}_{sec} Sys_2$$

if for any configuration $conf_1 = (M_1, G_1, s_1, \mathsf{H}, \mathsf{A}_1, G_{AH,1}) \in \mathsf{Conf}(Sys_1)$, there exists a configuration $conf_2 = (M_2, G_2, s_2, \mathsf{H}, \mathsf{A}_2, G_{AH,2}) \in \mathsf{Conf}(Sys_2)$ with $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ (and the same H) such that

$$view_{conf_1}(\mathsf{H}) = view_{conf_2}(\mathsf{H}),$$

unless H has ports from $\mathsf{free}(G_2, M_2) - s_2$.[9]

[9]Recall the explanation of the use of s_2 at the beginning of this section.

- We call Sys_1 **computationally at least as secure as** Sys_2 with respect to f and write
$$Sys_1 \geq_{sec}^{f,poly} Sys_2$$
if for any configuration $conf_1 \in \mathsf{Conf}_{\mathsf{poly}}(Sys_1)$, there exists a configuration $conf_2 \in \mathsf{Conf}_{\mathsf{poly}}(Sys_2)$ with $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ (and the same H) such that
$$view_{conf_1}(\mathsf{H}) \approx_{poly} view_{conf_2}(\mathsf{H}),$$
unless H has ports from $\mathsf{free}(G_2, M_2) - s_2$.

In both cases, we call $conf_2$ an "indistinguishable configuration" for $conf_1$ and write
$$conf_2 \in Indist_f(conf_1).$$

◇

Where the difference between perfect and computational security is irrelevant, we simply write $\geq_{sec}$ or $\geq_{sec}^f$.

Remark 8.14. In most cryptographic examples (including the examples described in Chapter 9), a function f with $s_2 = s_1$ for all $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ is sufficient. This corresponds to the fact that the ideal system provides the same specified ports than its implementation. ○

8.3 Properties of our Definition

8.3.1 Transitivity

An important lemma is that $\geq_{sec}$ is transitive, as one would expect.

Lemma 8.2
If $Sys_1 \geq_{sec}^{f_1} Sys_2$ and $Sys_2 \geq_{sec}^{f_2} Sys_3$, then $Sys_1 \geq_{sec}^{f_3} Sys_3$, where $f_3 := f_2 \circ f_1$ is defined in a natural way: $f_3(M_1, G_1, s_1)$ is the union of the sets $f_2(M_2, G_2, s_2)$ with $(M_2, G_2, s_2) \in f_1(M_1, G_1, s_1)$. □

Proof. The preconditions imply that for any configuration $conf_1 \in \mathsf{Conf}(Sys_1)$, there exists a configuration $conf_2 \in \mathsf{Conf}(Sys_2)$ with the same user H such that $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$, and $conf_3 \in \mathsf{Conf}(Sys_3)$, still with the same user H, such that $view_{conf_2}(\mathsf{H}) \approx view_{conf_3}(\mathsf{H})$ and $(M_2, G_2, s_2) \in f_1(M_1, G_1, s_1)$ and $(M_3, G_3, s_3) \in f_2(M_2, G_2, s_2)$. Thus $(M_3, G_3, s_3) \in f_3(M_1, G_1, s_1)$. Moreover, $view_{conf_1}(\mathsf{H}) \approx view_{conf_3}(\mathsf{H})$ holds because indistinguishability is transitive. This is trivial for perfect indistinguishability (equality). For computational indistinguishability and any distinguisher D, the sequence of probability differences between the views with index 1 and 3 is at most the sum of the two given sequences, and the sum of two negligible functions is negligible again. ■

One would also expect $\geq_{sec}$ to be reflexive with the identity function f, i.e., a system is as secure as itself if one compares each structure with itself. This does hold for the strictest type of comparison where each port is considered specified, i.e., all structures are strengthened to $(M, G, \mathsf{free}(G, M))$. Otherwise, however, we have to rename all ports in $\mathsf{free}(G, M) - s$ in one copy of the system to fulfill the condition that ports that are not supposed to be compared do not "by chance" have the same names.

8.3.2 Dialogues Between Honest Users and Adversary

In Section 8.1.6 we claimed that splitting rounds into four subrounds is sufficient, although in reality A and H can engage in a multi-round dialogue before producing their outputs for the correct machines. We now show that this is true, i.e., the notions defined in Def. 8.2 are not changed if we replace the three last subrounds in the switching model by an arbitrary multi-round dialogue.

To distinguish both models, we call the model with arbitrary dialogues the dialogue model and our standard model the quarter-round model. As the difference does not concern the structures, we can consider the same systems in both models. We denote the set of configurations of a system in the dialogue model by $\mathsf{Conf}_\mathsf{d}(Sys)$.

Lemma 8.3
Sys_1 is at least as secure as Sys_2 w.r.t. a mapping f in the dialogue model if and only if this holds in the quarter-round model. This is true for both computational and perfect security. □

Proof. Let Sys_1 and Sys_2 be two systems, and let f be a valid mapping. We can treat the perfect and the computational case together; in the second case all given adversaries and users are polynomial-time; this will imply that so are the constructed ones, similar to Lemma 8.1. We have to show that the following two statements are equivalent:

(1) For any configuration $conf_{d,1} = (M_1, G_1, s_1, \mathsf{H_d}, \mathsf{A_{1,d}}, G_{AH,1,d}) \in \mathsf{Conf}_\mathsf{d}(Sys_1)$, there exists a configuration $conf_{d,2} = (M_2, G_2, s_2, \mathsf{H_d}, \mathsf{A_{2,d}}, G_{AH,2,d}) \in \mathsf{Conf}_\mathsf{d}(Sys_2)$ with $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ (and the same dialogue user $\mathsf{H_d}$) such that

$$view_{conf_{d,1}}(\mathsf{H_d}) \approx view_{conf_{d,2}}(\mathsf{H_d}),$$

unless $\mathsf{H_d}$ has ports from $\mathsf{free}(G_2, M_2) - s_2$.

(2) For any configuration $conf_1 = (M_1, G_1, s_1, \mathsf{H}, \mathsf{A_1}, G_{AH,1}) \in \mathsf{Conf}(Sys_1)$, there exists a configuration $conf_2 = (M_2, G_2, s_2, \mathsf{H}, \mathsf{A_2}, G_{AH,2}) \in \mathsf{Conf}(Sys_2)$ with $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ (and the same H) such that

$$view_{conf_1}(\mathsf{H}) = view_{conf_2}(\mathsf{H}),$$

unless H has ports from $\mathsf{free}(G_2, M_2) - s_2$.

(1) implies (2): The configuration $conf_1$ is also valid in the dialogue model. Thus (1) implies that there is an indistinguishable configuration $conf_{d,2} \in$

$\mathit{Indist}_f(\mathit{conf}_1)$, but with a dialogue adversary $\mathsf{A}_{2,\mathsf{d}}$. Since H processes messages from $\mathsf{A}_{2,\mathsf{d}}$ only once in each subround $[i.1]$ and $[i.3]$, all messages sent by $\mathsf{A}_{2,\mathsf{d}}$ to H between these points in time can be concatenated into one message, respectively. This results in an equivalent quarter-round adversary A_2.

(2) implies (1): Let the given configuration in the dialogue model be $\mathit{conf}_{d,1} = (M_1, G_1, s_1, \mathsf{H}_\mathsf{d}, \mathsf{A}_{1,\mathsf{d}}, G_{AH,d,1})$. We first construct a quarter-round user H and a quarter-round adversary A_1 that compress the multi-round dialogue between H_d and $\mathsf{A}_{1,\mathsf{d}}$ into quarter-rounds (see Figure 8.6):

- In Quarter-round $[i.1]$, H does nothing.
- In Quarter-round $[i.2]$, machine A_1 forwards the inputs obtained from M_1 to the user H.[10]
- In Quarter-round $[i.3]$, machine H takes the inputs from M_1 and A_1 and internally simulates the dialogue between $\mathsf{A}_{1,\mathsf{d}}$ and H_d until they produce their outputs to M_1.[11] Machine H sends the resulting outputs to M_1 and to A_1.
- In Quarter-round $[i.4]$, machine A_1 uses the outputs received from H as alleged outputs of $\mathsf{A}_{1,\mathsf{d}}$ to the correct machines.

Let $\mathit{conf}_1 = (M_1, G_1, s_1, \mathsf{H}, \mathsf{A}_1, G_{AH,1})$ be the resulting quarter-round configuration. Now (2) implies that there exists an indistinguishable configuration $(M_2, G_2, s_2, \mathsf{H}, \mathsf{A}_2, G_{AH,2}) \in \mathit{Indist}_f(\mathit{conf}_1)$ with a quarter-round adversary A_2.

We now reverse the compression by combining $\mathsf{A}_{1,\mathsf{d}}$ and A_2 into a dialogue machine $\mathsf{A}_{2,\mathsf{d}}$ that naturally interacts with the original dialogue user H_d. The views of H in the configurations with $\mathsf{A}_1$and A_2 are indistinguishable. The views of H_d within H, i.e., in the right column of Figure 8.6, can be regarded as part of the view of H (see Lemma 8.1d), hence they are also indistinguishable. By construction, the views of H_d are identical in the two configurations in each row of Figure 8.6. Hence they are also indistinguishable in the two configurations in the left column. ■

Remark 8.15. For this proof, it makes no difference whether H also switches in Quarter-round $[i.1]$ or not. ○

8.3.3 Output of Guess by the Adversary

In addition to $\mathit{view}(\mathsf{H})$, one can consider a final output made by the adversary (e.g., like the guessing-outputs in definitions of semantic security [GoMi 84]), or all outputs the adversary makes to some unconnected result port. This intuitively corresponds to the idea that secrecy is captured by "whatever the *adversary* can do must be simulateable," while "whatever the *honest users* do and see is simulateable" captures integrity.

The definitions are modified such that each adversary has a distinguished output port guess (which is typically unconnected in the graph $G_{AH,g}$), and the pair of the user H_g's view and the events at this port must be indistinguishable.

[10]Note that there is no secrecy problem in this direction, i.e., there is no reason for A to keep secrets from H.

[11]W.l.o.g., this may be signaled by an end-of-dialogue symbol.

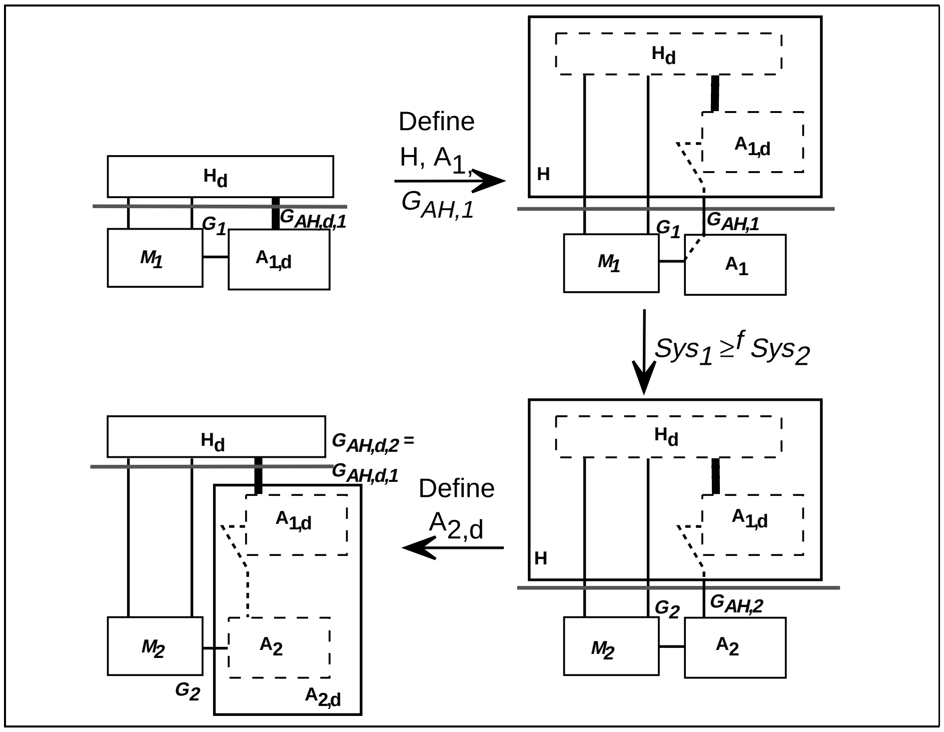


Figure 8.6: Simulateability by Compressing Dialogues to Quarter-Rounds. (A dialogue is depicted as a bold line.)

We show that this addition results in a definition that is equivalent to our general notion of secrecy. (We only need to allow replacement of a system by one with bijectively renamed ports in certain cases.)

Lemma 8.4
Sys_1 is at least as secure as Sys_2 with respect to f in the model with guessing output iff this holds in our standard model. This is true for both computational and perfect security. □

Proof. As with Lemma 8.3, we can treat the perfect and the computational case together. The proof structure is also similar.
(1) implies (2): Let Sys_1 be as secure as Sys_2 with guessing outputs, and let a configuration $conf_1$ for Sys_1 in the standard model be given. We obtain a valid configuration with guessing output if we augment A_1 by a port $guess$ (w.l.o.g., this name does not occur yet in (M_1, G_1, s_1, H, A_1, $G_{AH,1}$) and Sys_2) where it does not make any outputs. Thus (1) implies that there is an indistinguishable configuration $conf_{g,2}$ with guessing output. If one deletes the guessing output (e.g., by defining an adversary A_2 containing $A_{g,2}$ but ignoring its guess), this is also a valid configuration in the standard model, and indistinguishability of the pair of H's view and the guessing output clearly implies indistinguishability of H's view alone.
(2) implies (1): Now let Sys_1 be as secure as Sys_2 in the standard model, and consider a configuration (M_1, G_1, s_1, H_g, $A_{g,1}$, $G_{AH,g,1}$) with guessing output. Let $guess'$ be a port name that does not occur in (M_1, G_1, s_1, H, A_1, $G_{AH,1}$) and

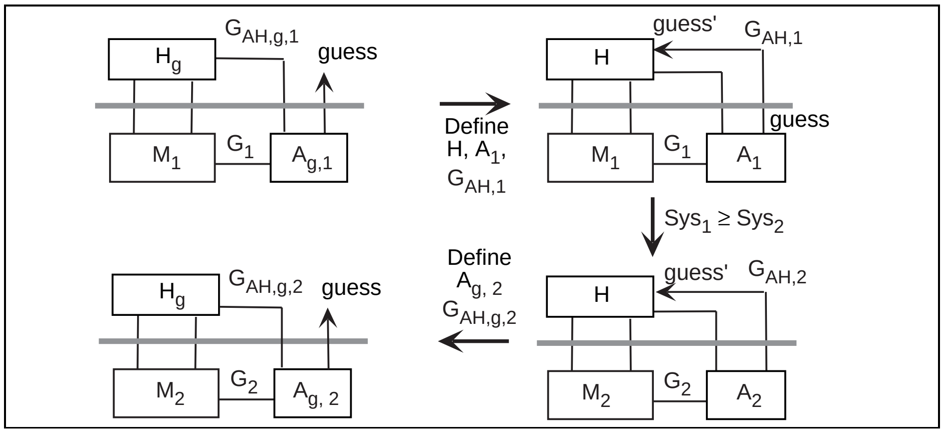


Figure 8.7: Machines with and without Guessing Outputs

Sys_2. We construct a related configuration $(M_1, G_1, s_1, \mathsf{H}, \mathsf{A}_1, G_{AH,1})$ where the former guessing output belongs to the user's view as follows (see Figure 8.7): The adversary A_1 equals $\mathsf{A}_{g,1}$, and H is like H_g, except that it has the new input port $guess'$, where it ignores all inputs. The graph $G_{AH,1}$ is $G_{AH,g,1}$ augmented by an edge connecting the port $guess$ with $guess'$. Let $(M_2, G_2, s_2, \mathsf{H}, \mathsf{A}_2, G_{AH,2}) \in Indist_f(M_1, G_1, s_1, \mathsf{H}, \mathsf{A}_1, G_{AH,1})$ be an indistinguishable configuration.

We retransform this into a configuration $(M_2, G_2, s_2, \mathsf{H}_g, \mathsf{A}_{g,2}, G_{AH,g,2})$ with guessing output: If $G_{AH,2}$ contains a connection from A_2 to the port $guess'$, the corresponding output port of A_2 is renamed into $guess$ (by the precondition, no port in the structure and of H_g has this name, and a port of A_2 could be renamed), and this connection is omitted in $G_{AH,g,2}$. (Recall that H is the same and ignored these inputs anyway.)

There is always such a connection from A_2: We did not allow unconnected ports, nor connections from H to itself. If $guess'$ were connected to an output port of M_2, this port would have to be in s_2, i.e., $guess'$ would not be new.

It is clear that the pair of the view of H_g and the guessing output is identical to the view of H in the upper row in the figure, and that the views of H in the right column of the figure are indistinguishable. By construction, the view of H_g and the guessing output is identical to the view of H in the lower row, i.e., the views of H_g in the left column of the figure are indistinguishable, too. ∎

8.3.4 Fixing a User Interface in the Structure

In particular in the cryptographic examples, we naturally expected honest users of a structure (M, G, s) to use precisely the set s of specified ports and to leave the other free ports to the adversary. We now show that this gives an equivalent definition in the case where $s_1 = s_2$ for all $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$. Hence we consider the restricted set $\mathsf{Conf_s}(Sys)$ of configurations with $\mathsf{ports(H)} \cap \mathsf{free}(G, M) = s$. We call this the fixed-user-interface model.

Lemma 8.5
Sys_1 is at least as secure as Sys_2 w.r.t. a mapping f with $s_1 = s_2$ for all $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ in the fixed-user-interface model if and only if this holds

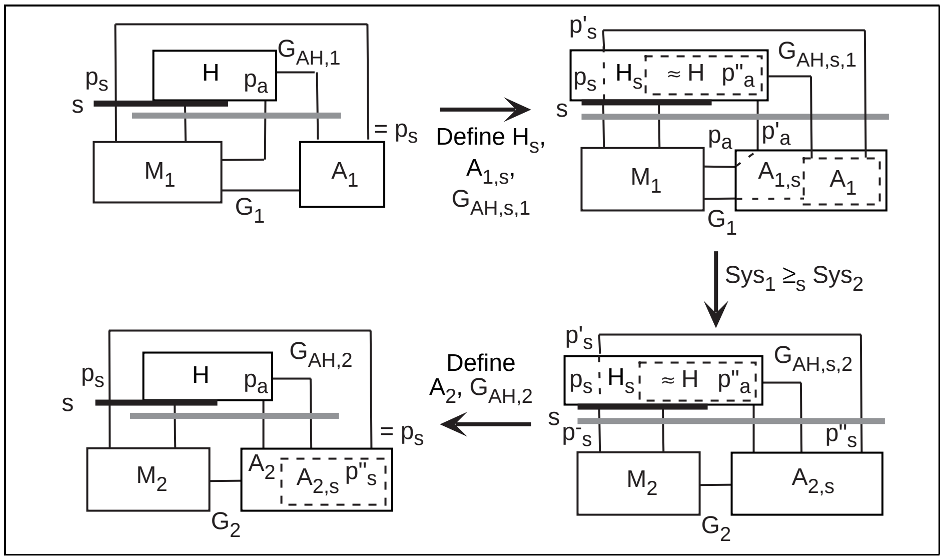


Figure 8.8: Fixed-User-Interface Model Implies Standard Model if $s_2 = s_1 = s$.

in the standard model. This is true for both computational and perfect security. □

Within the proof, the dialogue model will be useful. We then need the equivalent of Lemma 8.3 for the fixed-user-interface case:

Lemma 8.6
Sys_1 is at least as secure as Sys_2 w.r.t. a mapping f with $s_1 = s_2$ for all $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ in the fixed-user-interface model if and only if this holds in the model with fixed user interface and dialogues. This is true for both computational and perfect security. □

The proof is the same as that of Lemma 8.3, except that one has to show that the construction of the "compressing" user machine H respects the interface condition. This is clear because $\text{ports}(\mathsf{H}) \cap \text{free}(G_1, M_1) = \text{ports}(\mathsf{H_d}) \cap \text{free}(G_1, M_1)$.

Proof. (Lemma 8.5)

As usual, let Statement (1) be security in the new model, here the fixed-user-interface model, and Statement (2) be security in our standard model.

(1) implies (2): Let a configuration $(M_1, G_1, s_1, \mathsf{H}, \mathsf{A}_1, G_{AH,1}) \in \text{Conf}(Sys_1)$ in the standard model be given. We construct a related dialogue user $\mathsf{H_s}$ with the fixed user interface, i.e., $\text{ports}(\mathsf{H_s}) \cap \text{free}(G_1, M_1) = s_1$, see Figure 8.8, and a suitable adversary $\mathsf{A}_{1,s}$ and graph $G_{AH,s,1}$: Any port $p_s \in s_1 - \text{ports}(\mathsf{H})$ becomes a port of $\mathsf{H_s}$, and $\mathsf{H_s}$ simply forwards messages between this port and a corresponding new port p'_s connected to $\mathsf{A}_{1,s}$. Here we need the dialogue property: The forwarding is done in extra sub-rounds before and after each $[i.1]$. Any port $p_a \in \text{ports}(\mathsf{H}) \cap (\text{free}(G_1, M_1) - s_1)$ becomes a port of $\mathsf{A}_{1,s}$ instead, and $\mathsf{A}_{1,s}$ simply forwards messages between this port and a corresponding new port p'_a connected to $\mathsf{H_s}$. We call the endpoint at $\mathsf{H_s}$ port p''_a; $\mathsf{H_s}$ uses it just like H used p_a.

Now (1) and Lemma 8.6 imply that there is an indistinguishable configuration (in the dialogue and fixed-user-interface model) $conf_{s,2} = (M_2, G_2, s_2, \mathsf{H_s}$, $\mathsf{A_{2,s}}$, $G_{AH,s,2})$ with the same user $\mathsf{H_s}$ and $\mathsf{ports}(\mathsf{H_s}) \cap \mathsf{free}(G_2, M_2) = s_2$. Moreover, our precondition implies $s_2 = s_1$, hence in Figure 8.8 it is simply called s.

We now transform it into a configuration $(M_2, G_2, s_2, \mathsf{H}, \mathsf{A_2}, G_{AH,2}) \in \mathsf{Conf}(Sys_2)$: We must use the original H. One difference to $\mathsf{H_s}$ is that it does not have the ports called p_s and p'_s above. We simply want to rejoin the connections there (as in the simple case shown in Figure 8.8, but we have to check that all conditions on the graphs are fulfilled.

- Case 1: p_s is an input port (and thus p'_s an output port): Only one port can be connected to p_s. Several ports p''_s may be connected to p'_s, but they must all belong to $\mathsf{A_{2,s}}$ because $\mathsf{H_s}$ has no connections to itself, and no ports connected to M_2 outside s by the fixed-user-interface model (and $p'_s \notin s$).

  Hence we give $\mathsf{A_2}$ the port p_s; internally it forwards inputs there to all ports p''_s of its submachine $\mathsf{A_{2,s}}$.

- Case 2: p_s is an output port (and thus p'_s an input port): Now only one port p''_s can be connected to p'_s, and as above it belongs to $\mathsf{A_{2,s}}$, while p_s can be connected to several ports p_s^-, which may belong to both M_2 and $\mathsf{A_{2,s}}$.

  We give $\mathsf{A_2}$ the port p_s. If p''_s is a unicast port, it remains internal to $\mathsf{A_{2,s}}$ and $\mathsf{A_2}$ simply forwards its outputs to p_s. Otherwise p''_s also becomes a port of $\mathsf{A_{2,s}}$, and $\mathsf{A_2}$ internally duplicates the corresponding outputs from $\mathsf{A_2}$ to p_s and p''_s.

The other difference between H and $\mathsf{H_s}$ is that the ports called p_a were renamed. We want to give them their old names again. (And thus also rename them in the attached connections.) This presupposes that these names do not occur in the configuration yet. If they do and are in $\mathsf{free}(G_2, M_2) - s_2$, we need not fulfill any indistinguishability condition. They cannot be in $s_2 = s_1$. Otherwise, they are in $G_{AH} \cap \mathsf{ports}(\mathsf{A})$. Then we rename those other ports and retain a valid configuration.

Finally, $\mathsf{A_{2,s}}$ may be a real dialogue adversary. However, as H and M_2 only react on inputs from $\mathsf{A_2}$ once in the normal quarter-rounds $[i.1]$ and $[i.3]$, $\mathsf{A_2}$ can concatenate all outputs from $\mathsf{A_{2,s}}$ and output them once at the end of quarter-rounds $[i.2]$ and $[i.4]$.

Now the views of H in the two configurations in each row of Figure 8.8 are equal by construction except for port renaming, and the views of $\mathsf{H_s}$, which comprise those of H, in the right column. Hence the two views of H in the left column are also indistinguishable.

(2) implies (1): Now let a configuration in the fixed-user-interface model, $(M_1$, G_1, s_1, $\mathsf{H_s}$, $\mathsf{A_{1,s}}$, $G_{AH,s,1}) \in \mathsf{Conf_s}(Sys_1)$, be given. It is clearly also in $\mathsf{Conf}(Sys_1)$. Let $(M_2$, G_2, s_2, $\mathsf{H_s}$, $\mathsf{A_2}$, $G_{AH,2})$ be an indistinguishable configuration as it exists by (2). We have to show that it fulfills $\mathsf{ports}(\mathsf{H_s}) \cap \mathsf{free}(G_2, M_2) = s_2$. The general precondition on users implies $\mathsf{ports}(\mathsf{H_s}) \cap \mathsf{free}(G_2, M_2) \subseteq s_2$, and the fixed-user-interface model $\mathsf{ports}(\mathsf{H_s}) \cap \mathsf{free}(G_1, M_1) = s_1$. Together with $s_2 = s_1$ this implies $\mathsf{ports}(\mathsf{H_s}) \supseteq s_2$, and thus the claim. ■

8.3.5 Integrity Specifications

We now consider a modular usage of a trusted-host specification: One can show that the trusted host fulfills various requirements, e.g., safety requirements expressed in temporal logic. This may be done by formal and even automatic model checking if the trusted host is simple enough. At least it does not contain cryptographic operations, which would make the same task infeasible for the real system. Now we want to interpret these statements for the real system. Hence we want to show that the real system also fulfills these requirements in a certain cryptographic sense, i.e., even if parts of the system are under control of an adversary, but possibly only for polynomial-time adversaries and negligible error probabilities. Such a notion was defined (not quite as rigorously as here) in [Pfit8 96].

Clearly this can only hold for requirements that are formulated in terms of in- and outputs of the trusted host at the specified ports, and not its internal state, because the security definition only means that the real and the ideal system interact with their users in an indistinguishable way.

As a rather general version of integrity requirements (independent of the concrete formal language in which they will be formulated) we consider those that have a linear-time semantics, i.e., that correspond to a set of allowed traces of in- and outputs.

Definition 8.3 (Integrity Requirements)
By an integrity requirement Req for a system Sys, we mean a function that assigns a set of finite traces of in- and outputs at the ports in s to each set s with $(M, G, s) \in Sys$.

- The system Sys is said to fulfill Req perfectly if for any $conf = (M, G, s, \mathsf{H}, \mathsf{A}, G_{AH}) \in \mathsf{Conf}(Sys)$, the restrictions to s of all finite runs of this configuration lie in $Req(s)$.[12]
- The system Sys is said to fulfill Req computationally if for any $conf = (M, G, s, \mathsf{H}, \mathsf{A}, G_{AH}) \in \mathsf{Conf}_{\mathsf{poly}}(Sys)$, the following holds: The probability that the restriction to s of a run of this configuration (until the time when H stops) does not lie in $Req(s)$ is negligibly small (in the security parameter k).

$\diamond$

Lemma 8.7 (Conservation of Integrity Properties)
Let a system Sys_2 be given that fulfills an integrity requirement Req perfectly, and let a system Sys_1 be perfectly or computationally as secure as Sys_2 with respect to f with $s_1 = s_2$ for all $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$.

In the first case, Sys_1 also fulfills Req perfectly. In the second case, Sys_1 also fulfills Req computationally if membership in the set Req can be decided in polynomial time. $\square$

Note that Req is indeed also defined on Sys_1 under the preconditions: No $f(M_1, G_1, s_1)$ can be empty for security, and thus all sets s_1 also occur as a set s_2.

[12]Unlike our general model, this excludes availability requirements.

Proof. The basic idea is that if Sys_1 did not fulfill the requirement while Sys_2 does, this would offer a possibility to distinguish them, in contradiction to the definition of "$\geq_{sec}^{f}$".

Assume that a configuration $conf_1 = (M_1, G_1, s_1, \mathsf{H}, \mathsf{A}_1, G_{AH,1}) \in \mathsf{Conf}(Sys_1)$ exists that contradicts the claimed statement. Let H cover s completely (restricting ourselves to such user machines is equivalent for $s_1 = s_2$; see Lemma 8.5).

Let the systems be perfectly as secure as and let $conf_2 \in \mathsf{Conf}(Sys_2)$ with $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ be the configuration such that

$$view_{conf_1}(\mathsf{H}) = view_{conf_2}(\mathsf{H})$$

that exists by Definition 8.2. Then, the perfect indistinguishability of these views contradicts the assumption that $conf_1$ does not fulfill $Req(s)$ while all $conf_2$ do.

Let the systems be computationally as secure as and let $conf_2 \in \mathsf{Conf}_{\mathsf{poly}}(Sys_2)$ with $(M_2, G_2, s_2) \in f(M_1, G_1, s_1)$ be the configuration such that

$$view_{conf_1}(\mathsf{H}) \approx_{poly} view_{conf_2}(\mathsf{H})$$

that exists by Definition 8.2. We define a distinguisher Dist that distinguishes H's views: Given the view of machine H in Sys_1 or Sys_2, the distinguisher can extract the trace at s_1 that we consider. It verifies if this trace lies in $Req(s)$. If not, Dist outputs 1 (meaning that it believes H interacted with system Sys_1), otherwise 0.

This distinguisher is polynomial-time (in the global inputs of the system) because the view of H is polynomial-length, and membership in Req was assumed to be polynomial-time decidable. The distinguisher has a significant advantage in distinguishing because the probability that it outputs 0 if H interacted with Sys_2 is one, while the probability that it outputs 1 if H interacted with Sys_1 is non-negligible. ∎

Chapter 9

Some Provably Secure Reactive Systems

In order to illustrate how our new notion of reactive security can be applied to practical systems, we provide two examples: a reactive system for secure channels, and a reactive system for certified mail. For each example we present a specification using an ideal system and an implementation and show that the implementation is at least as secure as the ideal system.

9.1 Secure Channels

Our notion of secure channels offers secrecy and authenticity of transmitted messages. This means that an adversary must not obtain knowledge on the contents of messages exchanged between correct players and must not be able to send seemingly correct messages in the name of a correct sender.

9.1.1 An Ideal System

The goal of the ideal system for secure channels is to transmit messages between any two users while keeping messages delivered between correct users secret and unchanged. The accepted attacks for secure channels are:

Deleting Messages: The adversary shall be enabled to suppress messages sent between correct machines (because authentication does not guarantee availability). This is modeled by a $suppress$-matrix input by the adversary and specifying which messages should be deleted in a specific round.

Seeing Encrypted Network Traffic: The adversary may learn which correct players sent messages to whom, i.e., we assume that traffic will be produced on the network if a user inputs a message. This is modeled by a $busy$-matrix output to the adversary.

Multi-Round Rushing: The adversary can receive and send messages without the delay of the transport caused by using the actual network (see Section 6.2.2). To model this, the trusted host serves the adversary without delays. For instance, a message from a correct player to the adversary is output in the same round, even though messages to other correct players take more rounds.

Scheme 9.1 (Ideal System for Secure Channels)
Let a set $Msg \subseteq \{0,1\}^{\leq len}$ of messages of length at most len be given with $\epsilon \in Msg$, where ϵ is the empty word and stands for "no message". Let a number $n \in \mathbb{N}$ of intended participants be given and $M := \{1, \ldots, n\}$, and let $rnds \in \mathbb{N}$ be the intended number of rounds.

An ideal system Sys^{chan} for secure channels is then defined as the set of all structures

$$(\{\mathsf{TH}(H)\}, G_{TH}(H), s^*(H))$$

for any set $H \subseteq M$ of correct users[1], where $\mathsf{TH}(H)$, $G_{TH}(H)$, and $s^*(H)$ are defined as follows:

Ports and Graph: Let $A = M - H$ be the set of the corrupted players. The trusted host has an input port $in(u)$ and an output port $out(u)$ for each correct user $u \in H$. Furthermore, it has output ports adv_out and $busy$ and input ports adv_in and $suppress$ for the adversary machine A. We define $s^*(H) := \{in'(u), out'(u)\}_{u \in H}$, i.e., each in- and output port of a correct user has a corresponding port in $s^*(H)$. (This distinction of non-primed and primed ports is only needed to define $s^*(H)$ as a subset of the free ports of $G_{TH}(H)$; thus, we usually omit this distinction in the sequel and use $(in(u), out(u))$ for the ports of TH as well as H.)

The graph $G_{TH}(H)$ connects $in'(u)$ to $in(u)$ and $out(u)$ with $out'(u)$ and duplicates each of the four adversary ports by defining a primed variant and connecting it with the non-primed one; see Figure 9.1. (In the sequel, we again subsume the primed and non-primed adversary ports under the non-primed port to be connected to the adversary.)

In Round 0, TH does nothing. (This is time reserved for initialization in the real system.) Now we consider any Round $i > 0$.

Inputs: The overall inputs that TH accepts from honest users are denoted by a matrix $in_i : H \times M \to Msg$, where each row $(in_i(s, r) | r \in M)$ is the input at port $in(s)$. (If a user H makes other inputs, they are set to ϵ in TH.) This means that every user s may send one message to every other user r in each round. At the adversary ports, the trusted host expects the input of matrices $adv\_in_i : A \times H \to Msg$ and $suppress_i : H \times H \to \{0, 1\}$.

Computations: For all $s, r \in H$, the trusted host computes the following values:

$$out_i(s, r) := \begin{cases} \epsilon & \text{if } suppress_i(s, r) = 1 \\ in_{i-1}(s, r) & \text{else;} \end{cases} \quad (9.1)$$

$$busy_i(s, r) := \begin{cases} 0 & \text{if } in_i(s, r) = \epsilon \\ 1 & \text{else.} \end{cases} \quad (9.2)$$

[1]In the sequel, we use the identifier H for the *set* of names of the correct users whereas H denotes the corresponding user-*machine*. Similarly, A will denote corrupted users and A the adversary machine.

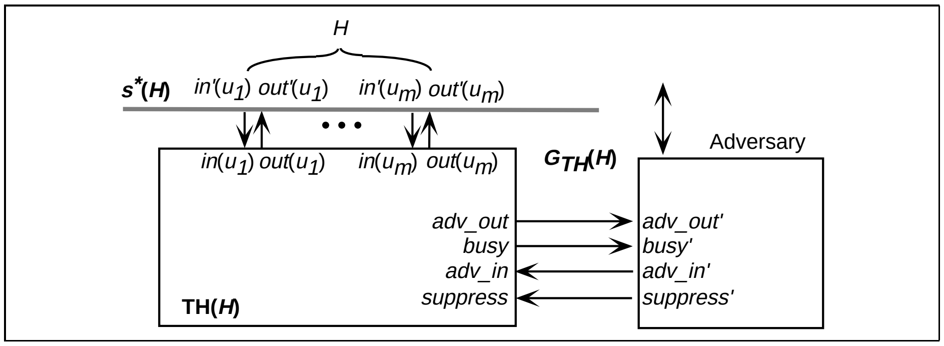


Figure 9.1: Structure of the Ideal System for Secure Channels

This means that messages between honest users take one round, while the adversary immediately sees that such a message is in transit. For $a \in A$, $u \in H$, the trusted host computes

$$out_i(a,u) := adv\_in_i(a,u); \tag{9.3}$$
$$adv\_out_i(u,a) := in_i(u,a). \tag{9.4}$$

This means that messages to and from the adversary are delivered immediately.

Outputs: The matrices $adv\_out_i : H \times A \to Msg$ containing messages for corrupted users and $busy_i : H \times H \to \{0,1\}$ containing activity flags are output at the corresponding adversary ports. Each column $(out_i(s,r)|s \in M)$ of the matrix $out_i : M \times H \to Msg$ is output at the port $out(r)$.

$\diamond$

Remark 9.1. Note that this trusted host specifies that messages must be authenticated, i.e., that an adversary cannot send messages in the name of other users. A trusted host for secrecy without authenticity would allow the adversary to input an extended matrix adv_out into the trusted host. It contains for each s and r (both may be correct) either a message to be sent on behalf of s to r, or a flag indicating whether the original message shall be suppressed or delivered.

Remark 9.2. For simplicity, we assume that the adversary can suppress messages sent from a party to itself, i.e., we do not handle these messages differently. For the implementation, this implies that if a user wants to send a message to itself, this is done using the insecure network. $\circ$

9.1.2 A Provably Secure System

We now define an actual implementation for secure encryption and authentication of messages.

Let len be the maximum length of the messages to be transmitted, let $rnds$ be a number of rounds, and let $M = \{1, \ldots, n\}$ be defined as in Def. 9.1.

In the following, the algorithms $(\mathsf{genS}, \mathrm{sign}, \mathrm{test})$ denote a secure digital signature scheme [DiHe 76, GoMR 88] for the message space $MsgS := Msg \times$

$\{1, \ldots, rnds\} \times M$ with $Msg = \{0,1\}^{\leq len}$.[2] We write $(\text{sign}_{\mathsf{u}}, \text{test}_{\mathsf{u}}) \leftarrow \mathsf{genS}(1^k)$ for the generation of a signature key sign_{u} and a verification key test_{u} based on a security parameter k input in unary representation. By $sig \leftarrow \text{sign}_{\mathsf{u}}(msg)$, we denote a signature on the message $msg \in MsgS$, including msg itself.[3] We denote the resulting signature space by $sig\_space(k)$ (the length of the signatures is polynomially bounded in k). The verification $\text{test}_{\mathsf{u}}(sig)$ returns msg if the signature is valid with respect to the included msg, else false.

By $(\mathsf{genE}, \mathsf{E}, \mathsf{D})$, we denote an encryption scheme secure against adaptive chosen-ciphertext attacks (e.g., [CrSh1 98]).[4] We write $(\mathsf{E_u}, \mathsf{D_u}) \leftarrow \mathsf{genE}(1^{k^*(k)})$ for the generation of an encryption key $\mathsf{E_u}$ and a decryption key $\mathsf{D_u}$. In this key generation, $k^*(k)$ denotes a security parameter that allows to encrypt messages in the message space $M \times sig\_space(k)$ for the given k.[5] By $c \leftarrow \mathsf{E_u}(m)$, we denote the (probabilistic) encryption of the message m using the key $\mathsf{E_u}$. The ciphertext is decrypted to the cleartext m by the function $m \leftarrow \mathsf{D_u}(c)$ using the key $\mathsf{D_u}$.

Scheme 9.2 (Secure Channels)
Let a set $Msg \subseteq \{0,1\}^{\leq len}$ of messages of length at most len be given with $\epsilon \in Msg$, where ϵ is the empty word and stands for "no message". Let a number $n \in \mathbb{N}$ of intended participants be given and $M := \{1, \ldots, n\}$, and let $rnds \in \mathbb{N}$ be the intended number of rounds. We mainly describe a set of machines $\{\mathsf{M_u} | u \in M\}$.

The system Sys is the set of all structures

$$(M(H), G(H), s^*(H))$$

for any set $H \subseteq M$ of correct users, where $M(H) = \{\mathsf{M_u} | u \in H\}$, $G(H)$ and $s^*(H)$ are defined as follows:

Ports and graph: Let $A = M - H$ be the set of the corrupted players. Each machine $\mathsf{M_u}$ has a pair $(in(u), out(u))$ of in- and output ports to a user (called user u), one pair $(in_0(v,u), out_0(u,v))$ of in- and output ports for exchanging keys for each $v \in M$ in Round 0, and a pair $(netw\_in(v,u), netw\_out(u,v))$ of in- and output ports for exchanging messages with $\mathsf{M_v}$ for each $v \in M$; see Figure 9.2. We define $s^*(H) := \{in'(u), out'(u)\}_{u \in H}$.

The connection graph defines authentic channels for key exchange. All other connections are left to the discretion of the adversary. We therefore define $G(H)$ as follows: Each key exchange output port $out_0(u,v)$ is connected to a free port $out'_0(u,v)$ as well as to the corresponding input port $in_0(v,u)$, i.e., the adversary can listen to, but not modify the key exchange channels between correct players. All other ports $netw\_out(s,r)$

[2] By "for a message space" we mean that the generated keys are appropriate to allow for this message space.

[3] We even assume more concretely that it is a pair of the message and the actual signature which can be uniquely decomposed. Then sig uniquely determines msg independent of the key. Otherwise more complicated message formats would be needed below, which is a waste of space in the standard case.

[4] In Section 9.1.3 we will define security against chosen-ciphertext attacks in more detail.

[5] We do not require all input messages to be of the same length. However, security for this message space implies that also the message length is hidden by the encryption.

and $netw\_in(s,r)$ are connected to ports $netw\_out'(s,r)$ and $netw\_in'(s,r)$ that are in $\mathsf{free}(G(H), M(H))$. Similarly, $G(H)$ connects $in'(u)$ to $in(u)$ and $out(u)$ with $out'(u)$. As before, we omit the primes in the sequel.

The behavior of each machine $\mathsf{M_u}$ is defined as follows (see Figure 9.4 for the exact timing of the most important message subsequence):

Round 0 *(Initialization)* Machine $\mathsf{M_u}$ generates four keys that are appropriate for the domains of the signature and encryption schemes, respectively:

$$\begin{aligned}(\mathrm{sign}_{\mathsf{u}}, \mathrm{test}_{\mathsf{u}}) &\leftarrow \mathsf{genS}(1^k)\\ (\mathsf{E_u}, \mathsf{D_u}) &\leftarrow \mathsf{genE}(1^{k^*(k)}).\end{aligned}$$

It outputs $(\mathrm{test}_{\mathsf{u}}, \mathsf{E_u})$ to $out_0(u,v)$ for $v \in M$.

Now we consider an arbitrary round $i > 0$.

Inputs Machine $\mathsf{M_u}$ accepts a vector

$$(in_i(u,r)|r \in M)$$

from its user input port. If $in_i(u,r) \notin Msg$ we again set $in_i(u,r) := \epsilon$. If $i = 1$, it also accepts two public keys $(\mathrm{test}_{\mathsf{v}}, \mathsf{E_v})$ at port $in_0(v,u)$. If $i > 1$, it reads one element $netw'_{i-1}(s,u)$ from each network input port.

Computations Machine $\mathsf{M_u}$ computes a network message $netw_i(u,r)$ for each $r \in M$ as follows:

$$netw_i(u,r) \leftarrow \begin{cases} \mathsf{E_r}(u, \mathrm{sign}_{\mathsf{u}}(in_i(u,r), i, r)) & \text{if } in_i(u,r) \neq \epsilon,\\ \epsilon & \text{else.}\end{cases} \tag{9.5}$$

If $i > 1$, it also decomposes[6] each received message $netw'_{i-1}(s,u)$ as follows: Let

$$(s_{i-1}(s,u), sig_{i-1}(s,u)) \leftarrow \mathsf{D_u}(netw'_{i-1}(s,u))$$

or, if the decryption or decomposition fails, both components be ϵ. Then

$$out_i(s,u) := \begin{cases} m & \text{if } s_{i-1}(s,u) = s\\ & \text{and } \mathrm{test}_{\mathsf{s}}(sig_{i-1}(s,u)) = (m, i-1, u)\\ \epsilon & \text{else.}\end{cases} \tag{9.6}$$

Outputs Machine $\mathsf{M_u}$ outputs the vector $(out_i(s,u)|s \in \{1,\ldots,n\})$ to its user. Furthermore, it outputs each $netw_i(u,r)$ on the corresponding network output port $netw\_out(u,r)$.

◇

Remark 9.3. The format of the network message may look complicated. However, simpler formats are not always secure. In particular, omitting the identity u in the encryption, i.e.,

$$netw_i(u,r) \leftarrow \mathsf{E_r}(\mathrm{sign}_{\mathsf{u}}(in_i(u,r), i, r))$$

[6]We assume that tuples are represented such that their decomposition is unambiguous.

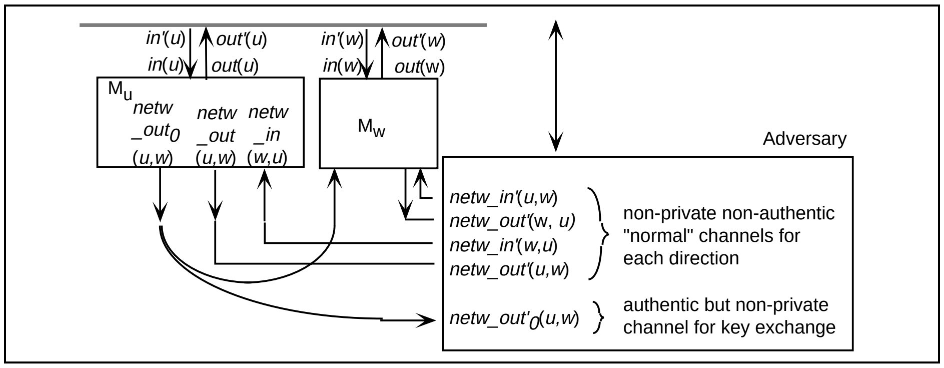


Figure 9.2: Two Machines in a Structure of the Real System for Secure Channels.

is insecure although one may feel that the signature inside determines the identity: An adversary can choose one of his own test keys test_a for the signature system equal to one of a correct machine, say test_s, because M_s switches in Round $[0.1]$ and A in Round $[0.2]$. Later it can then take a network message $netw_i(s,r)$ sent between two honest participants and also use it as its own network message $netw'_i(a,r)$ (again this works because A switches after M_s in Round i). This message then passes the test similar to Equation 9.6, and thus the message $in_i(s,r)$ is output to the user H also as $out_{i+1}(a,r)$. This is an effect that the adversaries on the trusted host cannot achieve. It is also dangerous in practice, because H, believing that it obtained this message from M_a, might freely send parts of it back to M_a in a reply.

Remark 9.4. The attack on network messages in this form as in Remark 9.3 can easily be avoided by verifying that all public keys are different. However, this does not imply provable security given the normal definition of a signature system: It is not forbidden that an adversary can choose a key related to the key of a correct machine such that signatures made with sign_s are also acceptable as signatures with respect to test_a.

Remark 9.5. First encrypting and then signing does not automatically work either, e.g.,

$$netw_i(v,r) \leftarrow \text{sign}_v(i, r, E_r(in_i(v,r)))$$

has the same problem even more obviously: The adversary can take the ciphertext $c = E_r(in_i(v,r))$ from such a message and also send it in a message of his own as $netw'_j(a,r) \leftarrow \text{sign}_a(j,r,c)$.

○

9.1.3 Security Proof

We now show that the implementation is as secure as the trusted host. Our simulation is blackbox and uniform in the user machine, i.e., for each set $H \subseteq M$ of correct machines, we define one simulator Sim, which can use any given adversary A for the real system as a blackbox sub-machine.

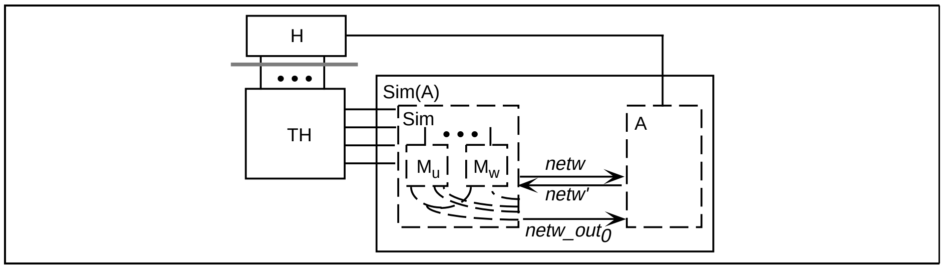


Figure 9.3: Set-up of the Simulation.

Scheme 9.3 (Simulator Sim)
Let a set $H \subseteq M$ of correct participants be given. Let $A := M - H$ be the names of the corrupted players. The interface between the "real" adversary A and the system was shown on the right-hand side in Figure 9.2.

We now define a corresponding adversary Sim(A) on the trusted-host system for any given adversary A of the real system (see Figure 9.3). It consists of a simulator Sim interacting with A.[7] The ports of Sim(A) were shown in Figure 9.1. Internally, it more or less simulates the behavior of the real machines $\{\mathsf{M_u} | u \in H\}$ using the information obtained from machine TH in order to give the simulated A its expected environment. See Figure 9.4 and 9.5 for the timing of the protocol and its simulation.

In the initial round, the simulator Sim simulates the correct machines without changes:

Round $[0.1]$ The trusted host TH and the user H switch. (Recall that adversaries, and thus Sim, only switch in even quarter-rounds.)

Round $[0.2]$ Sim simulates the key generation without changes: It first produces all the outputs corresponding to simulated correct machines; then it switches A. No changes are made to the connection between A and H.

Round $[0.3]$ The user H switches.

Round $[0.4]$ The simulator switches A only.

All subsequent rounds are simulated as follows:

Round $[i.1]$ The trusted host TH and the user machine H switch.

Round $[i.2]$ The simulator simulates the outputs of the correct machines:

a) Sim prepares all inputs to A: From TH, Sim obtains matrices $busy_i$ and adv_out_i.
   For each message sent from a correct sender $s \in H$ to a corrupted recipient $r \in A$, A expects to see a network message $netw_i(s,r)$ on the simulated lines. This is computed as in Equation 9.5 using $adv\_out_i(s,r)$ instead of $in_i(s,r)$.
   For each message sent between correct machines $s, r \in H$ signaled by the trusted host by an output $busy_i(s,r) = 1$, the adversary also

[7]One may be tempted to write Sim(A) := Sim × A. However, composition would mean that messages between Sim and A are only delivered in the next round, while Sim will preprocess inputs, switch its subprogram A and postprocess outputs all in one round.

expects to see an enciphered network message $netw_i(s, r)$ on the simulated lines. This is computed as in Equation 9.5 using any fixed message m_{sim} instead of $in_i(s, r)$.[8] We use a fixed message m_{sim} since the simulator cannot obtain the actual message input at the port $in(s)$ (it is kept secret by TH).

b) Sim switches A: A may make network outputs $netw'_i(s, r)$ for all $s \in M$ and $r \in H$ and outputs to H, which Sim simply passes on.

Round $[i.3]$ The user H switches.

Round $[i.4]$ The simulator computes its outputs to TH:

c) Sim switches A: I.e., A reads the inputs from H and produces its outputs $netw'_i(s, r)$ to the correct machines. Note that some of these outputs may have been written when switching A in Round $[i.2]$. Outputs from A to H are just forwarded.

d) Sim derives its external outputs: Sim converts the received network messages into inputs $suppress_{i+1}$ and adv_in_i to the adversary ports of TH:

For $s, r \in H$ and if $busy_i(s, r) = 1$, the trusted host expects to receive $suppress_{i+1}(s, r)$ indicating whether the adversary destroyed the message in transit. Therefore the simulator Sim verifies A's output $netw'_i(s, r)$ according to Equation 9.6. It sets $suppress_{i+1}(s, r) := 1$ if the output of the decomposition is ϵ and $suppress_{i+1}(s, r) := 0$ else. If the output of the decomposition is a message $msg' \neq msg_{sim}$ with $msg' \neq \epsilon$, or if it is any message $msg' \neq \epsilon$ although $busy_i(s, r) = 0$, the simulator stops.[9]

For $s \in A$ and $r \in H$, the message $netw'_i(s, r)$ is decrypted and verified as in Equation 9.6, and the result is output as adv_in_{i+1}.

$\diamond$

Note that the simulated correct machines switch at the beginning of Rounds $[i.2]$ and at the end of Rounds $[i.4]$, while real correct machines switch in Rounds $[i.1]$. This is no problem because the simulated A is clocked by the simulator: For A, the switching of the simulated correct machines at the end of Round $[i - 1.4]$, then TH, and again the simulated correct machines together looks like a normal Round $[i.1]$.[10]

We now show in Theorem 9.1 that the simulation is computationally correct given computationally secure public-key encryption and signature schemes. We use the well-known definition of secure digital signatures [GoMR 88]. For the security of the encryption scheme as needed in the second part of this proof, we use a variant of "message-restricted chosen-ciphertext attacks" [RaSi 92] as

[8] Intuitively, the security of the encryption scheme will later ensure that this message cannot be distinguished from $in_i(s, r)$.

[9] In the real system, msg' would be output to the recipient, but the simulator has no way of making TH do this. Hence the simulation would become distinguishable and can as well be stopped. However, it will be seen in the proof that this case presupposes that A has broken the signature scheme, and thus this case is negligible.

[10] This argument implicitly assumes that synchronous machines have no internal real-time clocks.

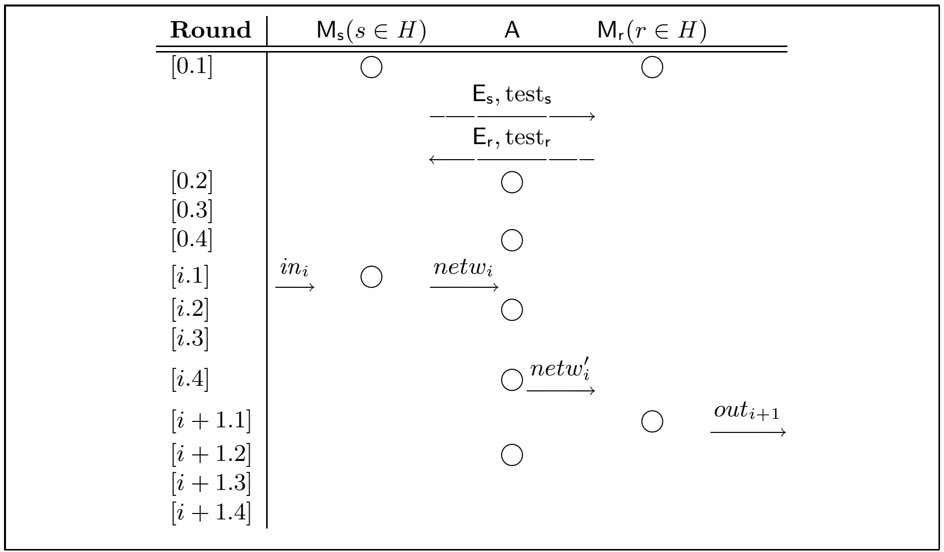


Figure 9.4: Exact Timing of Scheme 9.2 for a Message Between Two Correct Machines. (◯ denotes relevant switching of the machine in this column; H switches in [i.1] and [i.3].)

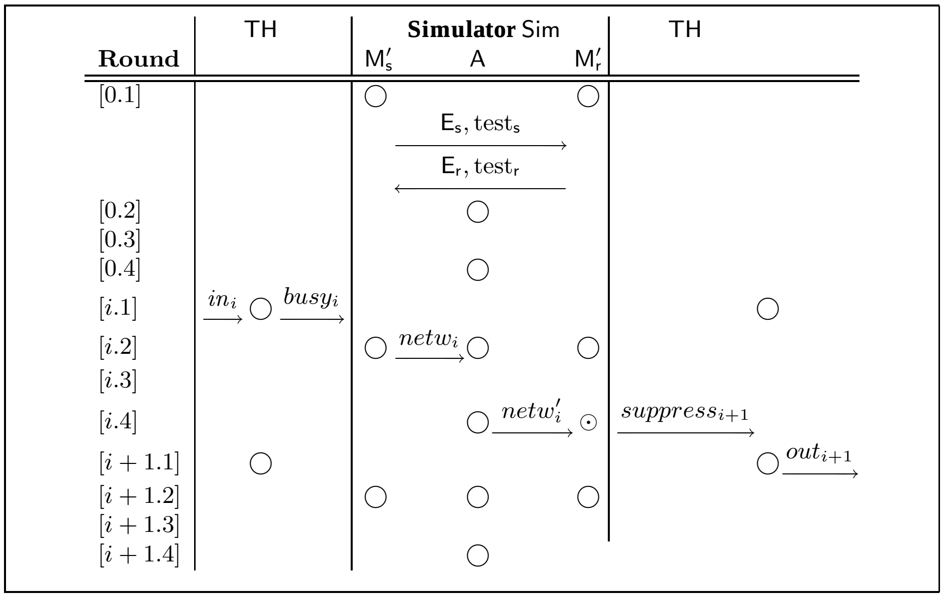


Figure 9.5: Simulation of Scheme 9.2 for a Message Between Two Correct Machines. (⊙ denotes additional state transitions not in the protocol.)

sketched in [CrSh1 98] and formalized as "IND-CCA2" in [BDPR 98]. The basic idea of this definition is that the adversary $\mathsf{A}_{\mathsf{enc}}$ plays the following game with a decryption oracle Dec:

1. The decryption oracle generates a key pair (E, D) for the given security parameter k and sends the public key E to $\mathsf{A}_{\mathsf{enc}}$.
2. $\mathsf{A}_{\mathsf{enc}}$ may ask Dec to decrypt any polynomial number of messages.
3. $\mathsf{A}_{\mathsf{enc}}$ sends two messages m_0 and m_1 to Dec, which randomly chooses a bit b and returns the encryption c of m_b.
4. $\mathsf{A}_{\mathsf{enc}}$ may then asks Dec to decrypt a polynomial number of ciphertexts c' with $c' \neq c$, i.e., Dec refuses to decrypt the given ciphertext c.
5. $\mathsf{A}_{\mathsf{enc}}$ outputs a bit b^*.

The attack succeeds if the probability of a correct guess $b^* = b$ is $1/2 + \epsilon_k$ for a sequence ϵ_k that is non-negligible in the given security parameter k.

Theorem 9.1 (Computational Security)
Let a set $Msg \subseteq \{0,1\}^{\leq len}$ of messages of length at most len be given with $\epsilon \in Msg$, where ϵ is the empty word and stands for "no message". Let a number $n \in \mathbb{N}$ of intended participants be given and $M := \{1, \dots, n\}$, and let $rnds \in \mathbb{N}$ be the intended number of rounds.

Then

$$Sys \geq_{sec}^{f,poly} TH$$

holds for Sys as defined in Scheme 9.2, TH as defined in Scheme 9.1, and

$$f((M(H), G(H), s^*(H))) := \{(\{\mathsf{TH}(H)\}, G_{TH}(H), s^*(H))\}.$$

□

Proof. We have to show that for any given configuration $(M(H), G(H), s^*(H), \mathsf{H}, \mathsf{A}, G_{AH})$ of the real system, there exists a configuration $(\mathsf{TH}(H), G_{TH}(H), s^*(H), \mathsf{H}, \mathsf{Sim}(\mathsf{A}), G_{AH})$ such that the families of distributions of the view of H are polynomially indistinguishable unless H has ports from $\mathsf{free}(G_{TH}(H), \mathsf{TH}(H)) - s^*(H)$.

We actually show that the joint views of A and H in the two cases are indistinguishable.[11] This is stronger because the view of H is contained in the joint view (see Lemma 8.1). In other words, we now consider H and A connected by G_{AH} that interact either with the correct machines $M(H)$ of the real system, or else with the combination of TH and Sim, which we abbreviate as $\mathsf{TH} + \mathsf{Sim}$. This combination is defined such that TH and Sim now all switch in Rounds $[i.1]$. Figure 9.3 illustrates that the given Sim never modifies messages between A and H. Another consequence of the simulation of the complete view of A and H is that we do not restrict H to ports of $s^*(H)$ (even though for simplicity H occupies $s^*(H)$ in our figures).

Intuitively, there are only two possible sources of errors in the simulation: in Round $[i.2]$ a message m_{sim} is encrypted instead of a real message, and a

[11]The view of A is well-defined because Sim runs A without rewinding. Since A is also clocked by Sim as it would be in the real system, the relative timing of H and A is also correct.

simulation may stop prematurely in Round $[i.4]$ due to what we already saw intuitively to be a successful signature forgery by A.

Signatures: We first show that the probability that the simulation stops prematurely is negligible. Let us assume the contrary, i.e., that there exists two machines A and H such that the simulation stops with a non-negligible probability. Then A and H can be converted into a successful adversary $\mathsf{A}_{\mathsf{sig}}$ against the underlying signature system[12]: $\mathsf{A}_{\mathsf{sig}}$ randomly chooses $u \in H$ and starts simulating TH and Sim interacting with the given A and H. It does everything as prescribed except that it uses the given public key test as $\mathrm{test}_{\mathsf{u}}$, and hence uses the given signing oracle whenever it has to execute $\mathrm{sign}_{\mathsf{u}}$. If the simulation stops prematurely (see Item (d) of Round $[i.4]$), a "network message" $netw'_{i-1}(s,r)$ for $s, r \in H$ has been received that is correct according to Equation 9.6 but contains a message $msg' \neq msg_{sim}$ or any message although $busy_{i-1}(s,r) = 0$. Then, if $s = u$, the algorithm $\mathsf{A}_{\mathsf{sig}}$ outputs the second component sig of $\mathsf{D}_{\mathsf{r}}(netw'_{i-1}(s,r))$ as its forgery.

We first show that if $\mathsf{A}_{\mathsf{sig}}$ makes any output, it is a successful forgery: By Equation 9.6, $\mathrm{test}_{\mathsf{u}}(sig) = (msg', i-1, r)$. By construction of Sim, the only possibility that a message with the components $i-1, r$ (here we use the assumed unique decomposition of the representation of message triples) would be signed by the simulated M_{u} (and thus $\mathsf{A}_{\mathsf{sig}}$ would have called the signature oracle for it) is in Round $[i-1.2]$ when constructing $netw_{i-1}(u,r)$. However, this happens only if $busy_{i-1}(u,r) = 1$, and with msg_{sim} as the first component. Hence the proposed forgery is in fact on a new message.

Now, we have to show that the probability that the constructed $\mathsf{A}_{\mathsf{sig}}$ outputs a valid signature is still non-negligible: It is clear that $\mathsf{A}_{\mathsf{sig}}$ perfectly simulates M_{u}, since the only change was to replace signature key generation and signing by the given oracles. If Sim stops prematurely, a signature of one of the $|H|$ players has been forged. Since the keys of the machines are chosen randomly and independently and $|H|$ is independent of k, the probability that the forgery succeeds for the given key $\mathrm{test}_{\mathsf{u}}$ used by a randomly chosen player is a fixed fraction of the overall probability of success and thus still non-negligible.

As a consequence, we can abstract from the negligible failures of the signature scheme in the sequel: This is formalized by considering machines $\mathsf{TH}' + \mathsf{Sim}'(\mathsf{A})$ that do not stop in Item d) of Step [i.4] but output the forged message msg' instead. It follows from the proof so far that $\mathsf{TH}' + \mathsf{Sim}'(\mathsf{A})$ and $\mathsf{TH} + \mathsf{Sim}(\mathsf{A})$ are indistinguishable if the signature scheme is secure.

Encryption: Now, for the encryption part, we make a reactive version of the typical "hybrid arguments", e.g., known from [GoMi 84]: Intuitively, we show that to distinguish the overall views, the distinguisher must be able to distinguish at least one particular encryption. For this, we consider hybrid machines $\mathsf{Hyb}_{i,s,r}$ for $i := 1, \ldots, rnds$ and $s, r \in H$ that treat the inputs $in_j(s', r')$ for $s', r' \in H$ as in the real system up to the triple (i, s, r) (in lexicographic order) and afterwards as in the simulation.[13] This means that up to Round $i-1$

[12]Given a public key and a signing oracle signing arbitrary messages, the goal of $\mathsf{A}_{\mathsf{sig}}$ is to output a valid signature on any message that has not been signed by the oracle with non-negligible probability [GoMR 88].

[13]It is not trivial in general to define hybrid reactive systems, even if one only wants to switch between the two configurations after a certain round: If the configurations are probabilistic and have memory, it may not be clear how to initialize the memory of configuration 2 such that it is consistent with the execution of configuration 1 so far. Hence we make the particular construction

the real system is run, except that the matrix in_{i-1} is stored. In Round $[i.1]$, for pairs $(s', r') \leq (s, r)$ with $s', r' \in H$, the network message $netw_i(s', r')$ is also computed as in the real system. For $(s', r') > (s, r)$ with $s', r' \in H$ and all subsequent rounds, it is computed as $netw_i(s', r') := \epsilon$ if $in_i(s', r') = \epsilon$ or else by applying Equation 9.5 to m_{sim}. Note that this is exactly the joint effect of $\mathsf{TH}' + \mathsf{Sim}'(\mathsf{A})$ since it outputs forgeries as well. For $s' \notin H$ or $r' \notin H$ there is no difference between $M(H)$ and $\mathsf{TH} + \mathsf{Sim}$ anyway.

Finally, let Hyb_0 be $\mathsf{TH} + \mathsf{Sim}$. Clearly, $\mathsf{Hyb}_{t_{max}}$ with $t_{max} := (rnds, s, r)$ and with maximum s and r is $M(H)$.

Let us assume that a distinguisher Dist exists that distinguishes Hyb_0 from $M(H)$ (i.e., $\mathsf{Hyb}_{t_{max}}$) with a non-negligible sequence of probabilities ϵ_k as defined in Def. 8.1. With $p_t(k)$, we denote the probability of an output 1 for Hyb_t. With $\mathsf{pred}(t)$, we denote the predecessor of t in the lexicographic ordering of the indices. Since

$$\left| \sum_{t=1}^{t_{max}} (-p_{\mathsf{pred}(t)}(k) + p_t(k)) \right| = |p_{t_{max}}(k) - p_0(k)| \geq \epsilon_k$$

and $p_t(k) \geq 0$, there is at least one t so that $|p_t(k) - p_{\mathsf{pred}(t)}(k)|$ is non-negligible, say ϵ'_k. Let $t = (i, s, r)$ be this index.

We assume that Dist outputs 0 to identify the simulation while it outputs 1 to identify the real system, i.e., that $p_t(k) \geq p_{\mathsf{pred}(t)}(k) + 1/poly(k)$ for a polynomial $poly(k)$ and infinitely many k.[14]

We now construct an adversary A_{enc} against the encryption scheme. It uses H, A and Dist as blackboxes and simulates either Hyb_t or $\mathsf{Hyb}_{\mathsf{pred}(t)}$ as follows; an overview is given in Figure 9.6.

1. First A_{enc} obtains a public key E from Dec. It uses this key in the place of $\mathsf{E_r}$, and generates all the other public keys for correct participants itself.

2. Then it simulates the machines $M(H)$ in interaction with the given blackboxes until directly before constructing $netw_i(s, r)$. Where decryptions with the unknown key $\mathsf{D_r}$ are needed, it uses the decryption oracle.

3. Now it sends the two messages $m_0 := (s, \text{sign}_{\mathsf{s}}(m_{sim}, i, r))$ and $m_1 := (s, \text{sign}_{\mathsf{s}}(in_i(s, r), i, r))$ to Dec. (If $in_i(s, r) = \epsilon$, it does nothing.) This is the message that is simulated in the hybrid system $\mathsf{Hyb}_{\mathsf{pred}(t)}$ but not in Hyb_t. As a response, Dec chooses $b \xleftarrow{\mathcal{R}} \{0, 1\}$ and sends the ciphertext $c := \mathsf{E_r}(m_b)$. Our A_{enc} then uses c as $netw_i(s, r)$.

4. From now on, it simulates $\mathsf{TH}' + \mathsf{Sim}'$. Again, if it needs a decryption with $\mathsf{D_r}$, it uses the decryption oracle. Now, however, the oracle refuses to answer if the query equals c.

   The only ciphertexts that A_{enc} has to decrypt with $\mathsf{D_r}$ are the possibly modified network messages $netw'_j(s', r)$, and the only usage it makes of the result is to compute $out_{j+1}(s', r)$ according to Equation 9.6.

   If $netw'_i(s, r) = c$ (intuitively, the adversary has not modified the network message $netw_i(s, r) = c$), our A_{enc} sets $out_{i+1}(s, r) = in_i(s, r)$.

explicit.

[14]Else, we consider a Dist′ that outputs the inverse of Dist.

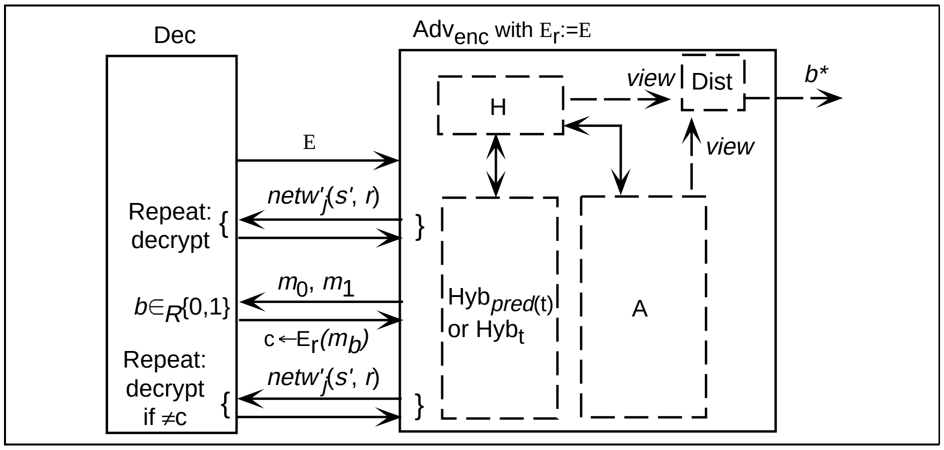


Figure 9.6: Adversary for the Encryption Scheme

If $netw'_j(s', r) = c$ for any other pair (j, s') (intuitively a replay attack) it sets $out_{j+1}(s', r) := \epsilon$.

5. At the end, A$_{enc}$ inputs the view of the simulated H $\times$ A to Dist. It uses the output bit b^* of Dist as its own output.

We now show that the success probability of A$_{enc}$ is equal to that of the distinguisher and thus non-negligible.

First it is clear by construction that if Dec chooses to encrypt m_0, then H and A interact with Hyb$_{\mathsf{pred}(t)}$, otherwise with Hyb$_t$, with only one possible difference: how $out_{j+1}(s', r)$ is computed if $netw'_j(s', r) = c$ for the given ciphertext c.

- Case $(j, s') = (i, s)$ and $b = 0$: Then machine Hyb$_{\mathsf{pred}(t)}$ would decrypt c to m_0, set $suppress_i(s, r) = 0$ and therefore its internal TH would output $out_{i+1}(s, r) = in_i(s, r)$, which is also what A$_{enc}$ does.
- Case $(j, s') = (i, s)$ and $b = 1$: Then machine Hyb$_t$ would decrypt c to m_1 and also obtain $out_{i+1}(s, r) = in_i(s, r)$ just like A$_{enc}$.
- Case $(j, s') \neq (i, s)$ and $b = 0$: Then machine Hyb$_{\mathsf{pred}(t)}$ would decrypt c to $m_0 = (s, sig)$. It first tests that $s = s'$. Then, it verifies that $\mathrm{test}_{\mathsf{s}}(sig) = (m', j, r)$ for any $m' \in Msg$ and the given (j, r). As we assumed that the signature verification outputs the signed message, this implies that $j = i$. Since $(j, s') \neq (i, s)$, the verifications fail. It sets $suppress_j(s', r) = 1$ and the output of TH is $out_{j+1}(s', r) = \epsilon$ just like A$_{enc}$'s output.
- Case $(j, s') \neq (i, s)$ and $b = 1$: Then machine Hyb$_t$ would decrypt c to $m_1 = (s, sig)$. It first tests that $s = s'$. Then, it verifies that $\mathrm{test}_{\mathsf{s}}(sig) = (m', j, r)$ for any $m' \in Msg$ and the given (j, r). As we assumed that the signature verification outputs the signed message, this implies that $j = i$. Thus, since $(j, s') \neq (i, s)$, the verifications fail and the output is $out_{j+1}(s', r) = \epsilon$ just like A$_{\mathsf{enc}}$.

Hence the success probability of Dist within A$_{enc}$ is the same as in its normal setting. Whenever Dist outputs a guess $b^* = b$ while interacting with Hyb$_{\mathsf{pred}(t)}$ or Hyb$_t$, the same correct guess b^* is output by A$_{enc}$, too.

Let $p_t(k) \geq p_{\mathsf{pred}(t)}(k) + 1/poly(k)$ for a polynomial $poly(k)$ and infinitely many k as described above. Since $b = 1$ and $b = 0$ with probability $1/2$, we can compute the success probability of A_{enc} as

$$\begin{aligned} p_{enc}(k) &= \frac{1}{2}p_t(k) + \frac{1}{2}(1 - p_{\mathsf{pred}(t)}) \quad \text{(i.e., } b = 1, b^* \neq 0 \text{ and } b = 0, b^* = 0) \\ &= \frac{1}{2} + \frac{1}{2}(p_t(k) - p_{\mathsf{pred}(t)}) \\ &\geq \frac{1}{2} + \frac{1}{2poly(k)}. \end{aligned}$$

Hence, if Dist can in fact distinguish the views, then A_{enc} is indeed a successful adversary against the encryption system that succeeds with non-negligible probability, and we have reached the desired contradiction. ■

9.2 Labeled Certified Mail

We now apply the formalism to labeled certified mail. We define an ideal system specifying a service of reactive labeled certified mail. We then present an actual implementation and prove that it is as secure as this ideal system.

It turns out that we have to modify the certified mail protocols from Chapter 4, i.e., most of them do not fulfill the strong reactive security definition based on simulatability. (This does not seem to be a matter of how we define the ideal system, but a more fundamental question related to the general composability that follows from this strong definition.) Of course, the protocols from Chapter 4 can nevertheless be used in any application for which the definition from Chapter 4 is sufficient. The main technical modification is that, instead of using encryption to fix the message, we now use a so-called chameleon commitment scheme. However, the resulting certified mail scheme does not fulfill the requirement "limited trust in the third party" (R. 4.2c of R. 4.3b) from Chapter 4 since this chameleon commitment enables an incorrect third party to later forge receipts. Thus, the resulting scheme is not "optimistic" in our sense as defined in Chapter 4.

9.2.1 An Ideal System

We now describe an ideal system for labeled certified mail. This ideal system defines the service as well as the *maximum* information that we allow an adversary to obtain (in fact, the "service" to the adversary is considerably better than the vulnerabilities of our protocol). This ideal system assumes that sending a message and showing a receipt requires a fixed time. It defines the following attacks to be acceptable:

Suppressing a Run The adversary may suppress an execution of a certified mail protocol as long as the outcome is still fair.

This is a stronger model than in Part I: It models that the channels between normal correct machines can be modified (only those including the verifier or the third party cannot).

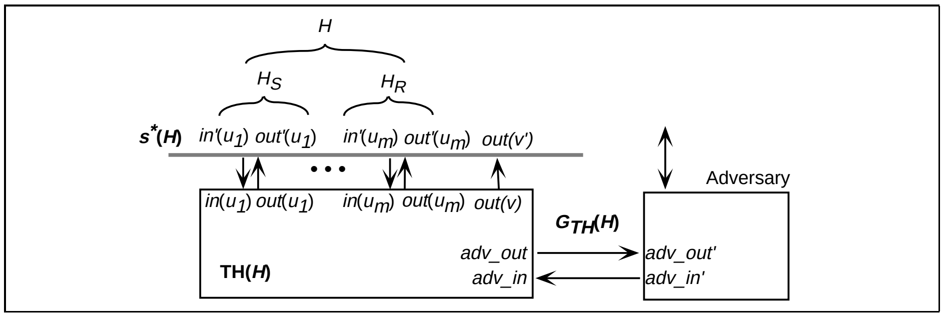


Figure 9.7: Structure of an Ideal System for Certified Mail

Maximum Information The adversary obtains all parameters except the message immediately. If a run is successful, the message can be obtained as well.

This corresponds to the fact that the network traffic need not be encrypted, i.e., the message may be transmitted in clear.

Adversary may Show Receipts The adversary may show any valid receipt.

This corresponds to the fact that we do not require authenticity of the channel to the verifier, i.e., not only the sender can show a valid receipt.

As mentioned in Section 7.1, trusted-host-based specifications tend to over-specify the intended service. An example of such an over-specification is that the trusted host fixes the timing of the protocol for correct users and that the trusted host ignores inputs with unfresh tids, while in Part I the responsibility for freshness was with the users and the behavior for the other case was unspecified.

Scheme 9.4 (Trusted Host for Labeled Certified Mail)
Let a message space $Msg \subseteq \{0,1\}^{\leq len}$, a label space $L \subseteq \{0,1\}^{\leq len}$, a set of transaction identifiers $TIDs \subseteq \{0,1\}^{\leq len}$, and a round number $rnds \in \mathbb{N}$ be given. Furthermore, let numbers $n_S, n_R \in \mathbb{N}$ of intended senders and recipients be given and let $M_S := \{1, \ldots, n_S\}$ and $M_R := \{n_S + 1, \ldots, n_S + n_R\}$. Let $v := n_S + n_R + 1$ so that the verifier can be treated in a similar way. Let $n := v$ and $M := \{1, \ldots, n\}$. Let Δ_{send} be the fixed intended time to send a message from a correct sender to a correct recipient and to issue a receipt.

An ideal system TH^{LCM} for labeled certified mail is then defined as the set of all structures

$$(\{\mathsf{TH}(H)\}, G_{TH}(H), s^*(H))$$

for any set $H \subseteq M$ of correct users with $v \in H$. The corrupted players are $A := M - H$. We abbreviate the sets of correct senders and recipients by $H_S := M_S \cap H$, and $H_R := M_R \cap H$, and the corrupted ones by $A_S := M_S \cap A$ and $A_R := M_R \cap A$. Machine $\mathsf{TH}(H)$, $G_{TH}(H)$ and the set $s^*(H)$ are defined as follows:

Ports and graph: Each user $u \in H$, $\mathsf{TH}(H)$ has pair $(in(u), out(u))$ of in- and output ports, except that no input port for the verifier v is needed. For the adversary, there are two ports: adv_in for inputs and adv_out for signaling whatever the adversary is allowed to know. The graph $G_{TH}(H)$ primes each port of $\mathsf{TH}(H)$, i.e., connects it to a free port with the same (but primed) name (see Figure 9.7). We define $s^*(H) := \{in'(u), out'(u)\}_{u \in H}$, i.e., each in- and output port of a correct user has a corresponding port in $s^*(H)$ (We again identify primed and non-primed ports in the sequel).

Overall structure of TH*:* Let

$$Slots := M_S \times M_R \times \{1, \ldots, rnds\}.$$

As the trusted host will allow each user to start a "send" protocol with each other user in each round, this corresponds to the "slots" or opportunities for such protocol runs. The trusted host contains matrices

$$\begin{aligned} tidstate &: H \times TIDs \rightarrow \{\epsilon\} \cup Slots; \\ slotstate &: Slots \rightarrow Localstate. \end{aligned}$$

Initially, $tidstate$ contains ϵ everywhere. Generally it denotes which correct participants have used which tids in which slot. $slotstate$ initially contains the starting states s_0, r_0, and sr_0 for $(s, r) \in H_S \times A_R$ or $A_S \times H_R$ or $H_S \times H_R$, respectively. Generally, $slotstate(s, r, i)$ is the state of a sub-machine $\mathsf{th}(s, r, i)$ for this slot. If the slot is not used, i.e., neither s nor r start a "send" protocol with the other in this round, the starting state always remains. After the successful end of a "send" protocol with label l and message m, the state is $(\mathsf{received}, l, m, tid)$ and $\mathsf{th}(s, r, i)$ can be used by the sender or the adversary to "show" the receipt.

We first describe how TH dispatches global inputs to the machines $\mathsf{th}(s, r, i)$, then the programs of the machines $\mathsf{th}(s, r, i)$, and finally how their outputs are transformed into global outputs. The trusted host does nothing for $i = 0$. We now consider an arbitrary round i with $i > 0$:

Global inputs: At each input port $in(s)$ with $s \in H_S$, TH accepts an input vector $(inS_i(s, r) | r \in M_R \cup \{v\})$. An element $inS_i(s, r)$ with $r \in M_R$ is empty (i.e., ϵ) or a command (send, r, l, m, tid) with $l \in L$, $m \in Msg$, and $tid \in TIDs$. An element $inS_i(s, v)$ is empty or (show, tid) with $tid \in TIDs$. (Other inputs are set to ϵ in TH.)

At each input port $in(r)$ with $r \in H_R$, TH accepts an input vector $(inR_i(r, s) | s \in M_S)$ whose elements are either ϵ or a command (receive, s, l, tid) with $l \in L$ and $tid \in TIDs$.

At the port adv_in, the trusted host accepts a matrix adv_in_i over $Slots$. Each element $adv\_in_i(s, r, t)$ is a set of commands from the adversary for the sub-machine in the given slot. A command may be one of the following values: adv_receive signals that the adversary is willing to receive the message handled by this sub-machine. $(\mathsf{adv\_send}, m)$ enables the adversary to send a message m. adv_suppress suppresses the given run if possible. adv_show shows the receipt.

Input dispatching: The trusted host first considers all inputs from the adversary:

The adversary inputs $adv\_in_i(slot)$ with $slot \in Slots$. The trusted host inputs each correct command in $adv\_in_i(slot)$ to $\mathsf{th}(slot)$. (Note that user-inputs such as send or receive are ignored since they are no correct adversary commands.)

After dispatching all adversary inputs to the sub-machines, all sub-machines are switched once for each command (in alphabetical order), i.e., adversary inputs are handled with a higher priority than user-inputs. (Note that these are only "virtual" submachines, i.e., this does not contradict the synchronous model, it is only a way to keep the state-transition figures simpler.)

Then, it processes the input elements from the correct users in the order as listed above.[15] [16]

For any input element ϵ, it does nothing.

For $inS_i(s,r)$ = $(\mathsf{send}, r, l, m, tid)$, it verifies that tid is fresh for s, i.e., $tidstate(s, tid) = \epsilon$. If yes, it sets $tidstate(s, tid) := (s, r, i)$ and inputs $(\mathsf{send}, l, m, tid)$ to the sub-machine $\mathsf{th}(s, r, i)$.[17]

For $inS_i(s,v)$ = (show, tid), it looks up $slot := tidstate(s, tid)$. If this is not ϵ, it inputs show to the sub-machine $\mathsf{th}(slot)$.

For $inR_i(r,s)$ = $(\mathsf{receive}, s, l, tid)$, it checks whether tid is fresh for r, i.e., $tidstate(r, tid) = \epsilon$. If yes, it sets $tidstate(r, tid) := (s, r, i)$ and inputs $(\mathsf{receive}, l, tid)$ to $\mathsf{th}(s, r, i)$.

The trusted host inputs stop to all machines $\mathsf{th}(s, r, i - \Delta_{send})$ to trigger the output of their results after Δ_{send} rounds.

Sub-machines: The state diagrams of the machines $\mathsf{th}(s, r, i)$ are shown in Figures 9.8 to 9.10. There are three types of machines, one each for $(s, r) \in H_S \times A_R$, $A_S \times H_R$ or $H_S \times H_R$. Inputs that are not explicitly shown in a state are ignored. Recall that stop is always input by TH after the fixed time Δ_{send}.

The diagrams are complete except that the local variables are not shown in the states. The states s_0, r_0, sr_0, and failed are without parameters. State sr_3 stores two possibly different tids, one input by the sender and one input by the recipient. All other states have variables l, m, and tid. Variables are initially ϵ and then set by the first input with the same name; later ones are interpreted as comparisons, i.e., incoming messages with different parameters are ignored.

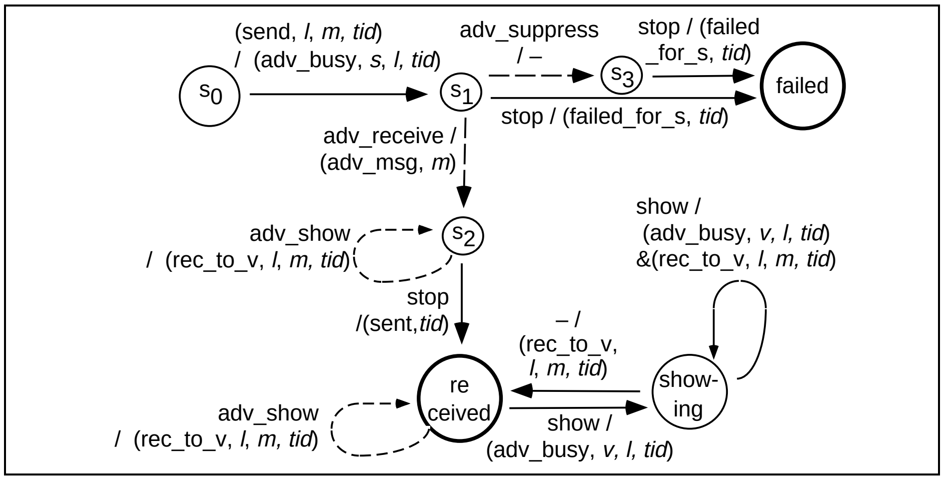
The basic idea of the machines in Figure 9.8 and 9.9 is that the adversary acting on behalf of the incorrect participant is notified that a run has been started. If the adversary then makes the required input, the protocol run succeeds and else fails.

For two correct parties, more cases need to be distinguished (see Figure 9.10): State sr_3 corresponds to the case where the parties input different parameters or the adversary suppressed the run. In state sr_4 both

[15]The only non-determinism resulting otherwise would be if one user inputs two commands with the same tid in the same round.

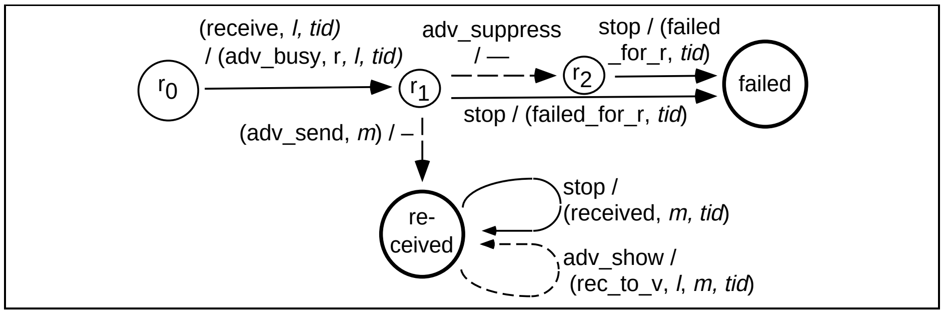
[16]This may cause a second state transition of a given sub-machine that already processed an input from the adversary.

[17]The sub-machines do not get the parameters $s, r,$ and i, which TH handles itself.



(send, *l*, *m*, *tid*) / (adv_busy, *s*, *l*, *tid*)
adv_suppress / –
stop / (failed _for_s, *tid*)
failed
stop / (failed_for_s, *tid*)
adv_receive / (adv_msg, *m*)
adv_show / (rec_to_v, *l*, *m*, *tid*)
stop /(sent,*tid*)
show / (adv_busy, *v*, *l*, *tid*) &(rec_to_v, *l*, *m*, *tid*)
– / (rec_to_v, *l*, *m*, *tid*)
show / (adv_busy, *v*, *l*, *tid*)
re ceived
show- ing
adv_show / (rec_to_v, *l*, *m*, *tid*)

Figure 9.8: Trusted Host for Certified Mail: $\mathsf{th}(s, r, i)$ for a correct s and incorrect r. (The scheduling of each round first executes an enabled dashed transition and then a solid one.)



(receive, *l*, *tid*) / (adv_busy, r, *l*, *tid*)
adv_suppress / —
stop / (failed _for_r, *tid*)
failed
stop / (failed_for_r, *tid*)
(adv_send, *m*) / –
re- ceived
stop / (received, *m*, *tid*)
adv_show / (rec_to_v, *l*, *m*, *tid*)

Figure 9.9: Trusted Host for Certified Mail: $\mathsf{th}(s, r, i)$ for a correct r and incorrect s.

input matching parameters and the machine is idling and waiting for the end of the potentially successful protocol run. If the adversary asks for the message, the sub-machine changes into state sr_5 to disable suppressing this run. If the adversary suppresses this run, the sub-machine changes into state sr_3 to prevent that the adversary obtains the message. In state sr_1 and sr_2, only one of the participants made an input and the protocol waits for termination in order to output failed. Note that the adversary is able to show the receipt in state sr_5 already while the correct sender still waits for the end of the protocol.

Output dispatching: All outputs of multiple values at one port are arranged in sets, i.e., multiple outputs of one value are ignored. In each round i, each machine $\mathsf{th}(s, r, j)$ makes at most three outputs. While processing the adversary's input, the machine may produce an output rec_to_v or $(\mathsf{adv\_msg}, m)$. Furthermore, at most two outputs separated by "&" are produced while processing the inputs of the correct users. Machine TH dispatches them to the global output ports as follows:

(sent, tid), which only occurs for $s \in H$, leads to an element (sent, tid) at

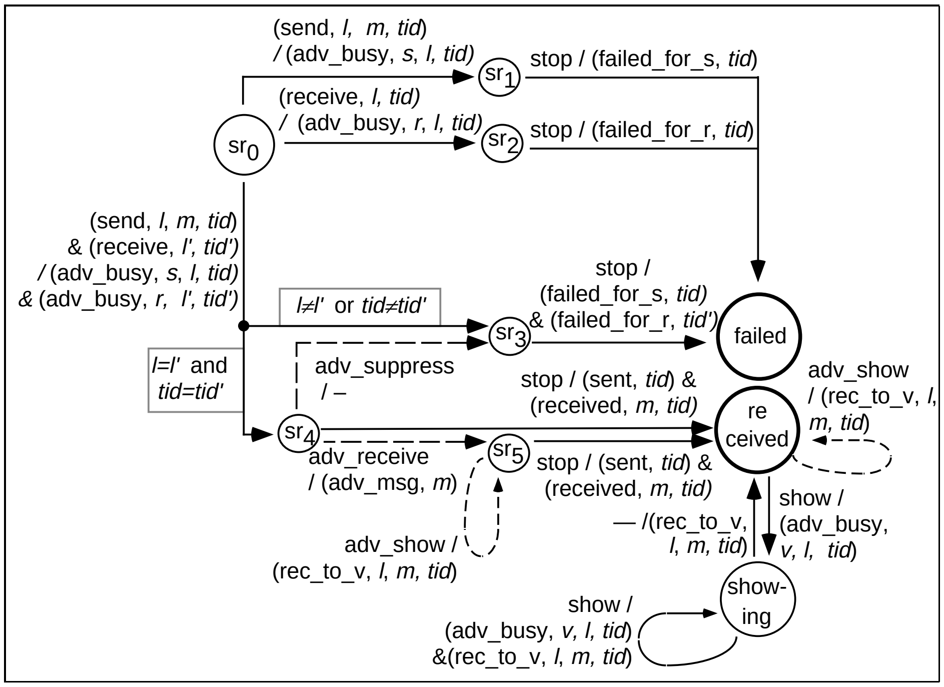

(send, l, m, tid) / (adv_busy, s, l, tid)
sr1
stop / (failed_for_s, tid)
(receive, l, tid) / (adv_busy, r, l, tid)
sr0
sr2
stop / (failed_for_r, tid)
(send, l, m, tid) & (receive, l', tid') / (adv_busy, s, l, tid) & (adv_busy, r, l', tid')
l≠l' or tid≠tid'
stop / (failed_for_s, tid) & (failed_for_r, tid')
sr3
failed
l=l' and tid=tid'
adv_suppress / –
stop / (sent, tid) & (received, m, tid)
adv_show / (rec_to_v, l, m, tid)
sr4
re ceived
sr5
adv_receive / (adv_msg, m)
stop / (sent, tid) & (received, m, tid)
show / (adv_busy, v, l, tid)
— /(rec_to_v, l, m, tid)
adv_show / (rec_to_v, l, m, tid)
show / (adv_busy, v, l, tid) &(rec_to_v, l, m, tid)
show-ing

Figure 9.10: Trusted Host for Certified Mail: $\mathsf{th}(s, r, i)$ for correct r and s.

the port $out(s)$.

$(\mathsf{received}, m, tid)$, which only occurs for $r \in H$, leads to an element $(\mathsf{received}, m, tid)$ at the port $out(r)$.

$(\mathsf{failed\_for\_s}, tid)$ leads to an element (failed, tid) at the port $out(s)$.

$(\mathsf{failed\_for\_r}, tid)$ leads to an element (failed, tid) at the port $out(r)$.

Each output of $(\mathsf{rec\_to\_v}, l, m, tid)$ leads to one element $(\mathsf{received}, s, r, l, m, tid)$ in the set of results output at port $out(v)$.

Each output of a tuple $(\mathsf{adv\_busy}, u, l, tid)$ by sub-machine $\mathsf{th}(s, r, t)$ leads to an element $(s, r, t, (\mathsf{adv\_busy}, u, l, tid))$ at the adversary port adv_out, and the output of $(\mathsf{adv\_msg}, m)$ leads to an element $(s, r, t, (\mathsf{adv\_msg}, m))$.

◇

Since Def. 4.1 required any labeled certified mail scheme to provide at least three machines, it is clear that no trusted host is a labeled certified mail scheme in that sense. However, we now show that our trusted hosts fulfill the requirements of labeled certified mail if we assume that the adversary does not suppress runs between correct players. Since it can be decided in polynomial time whether a given run fulfills the following integrity requirements (i.e., all requirements except [R. 4.2f]), the integrity of any secure implementation follows from Lemma 8.7.

For the secrecy of the message [R. 4.2f], this does not follow immediately. However, if sufficient information about the message would be revealed in an implementation (while it is not in TH), this information could be used to distinguish the trusted host and its implementation. Thus, the implementation would no longer be secure.

Lemma 9.1
The trusted host from Scheme 9.4 fulfills the security requirements in Def. 4.2 if the machine TH *is correct and ignores inputs* $adv\_in_i(s, r, t) = \mathsf{adv\_suppress}$ *for correct* s *and* r.[18] □

Proof. We consider each of the requirements:

Correct Execution (R. 4.2a): First let $s \in H_S$ and $r \in H_R$ input $(\mathsf{send}, r, l_S, m, tid)$ and $(\mathsf{receive}, s, l_R, tid)$, respectively, in Round i, and let tid be fresh for both. These become elements $inS_i(s, r)$ and $inR_i(r, s)$, and TH dispatches them to $\mathsf{th}(s, r, i)$. If $l_S = l_R$, the sub-machine enters state sr_4. As $\mathsf{adv\_suppress}$ is ignored, the input of stop, which TH makes after Δ_{send} Rounds, leads to an output (sent, tid) and $(\mathsf{received}, m, tid)$. These outputs are dispatched into outputs (sent, tid) at the port $out(s)$ and $(\mathsf{received}, m, tid)$ at the port $out(r)$. If $l_S \neq l_R$, the sub-machine enters state sr_3 and outputs $\mathsf{failed\_for\_s}$ and $\mathsf{failed\_for\_r}$, respectively. These outputs are dispatched into outputs (failed, tid) at port $out(s)$ and $out(r)$.

Secondly, consider that $(\mathsf{received}, m, tid)$ is output at the port $out(r)$ after an input $inR_i(r, s) = (\mathsf{receive}, s, l, tid)$ with fresh tid. Then $tidstate(r, tid)$ was ϵ before this input was dispatched and (s, r, i) afterwards, and $(\mathsf{receive}, l, tid)$ was input to $\mathsf{th}(s, r, i)$. No other $\mathsf{th}(s', r, j)$ ever gets an input with this tid, because earlier, it would have destroyed the condition $tidstate(r, tid) = \epsilon$ and later, it will not pass the test $tidstate(r, tid) = \epsilon$. Hence the output must come from an output $(\mathsf{received}, m, tid)$ of $\mathsf{th}(s, r, i)$. Figure 9.10 for correct s and r easily shows that this requires an input $(\mathsf{send}, l, m, tid)$ and thus a global input element $inS_{j'}(s, r) = (\mathsf{send}, r, l, m, tid)$.

Unforgeability of Receipts (R. 4.2b): An output $(\mathsf{received}, s, r, l, m, tid)$ at the port $out(v)$ only occurs after an output $(\mathsf{rec\_to\_v}, l, m, tid)$ of $\mathsf{th}(s, r, i)$ for some i. If the recipient is correct, this output is never made without an input $(\mathsf{receive}, l, tid)$. This input only occurs after an input $(\mathsf{receive}, s, l, tid)$ at Port $in_i(r)$.

Receipts are Fixed (R. 4.2c): If a verifier output $(\mathsf{received}, s, r, l, m, tid)$ then there exists a $\mathsf{th}(s, r, i)$ that stores m, l, and tid, and output $(\mathsf{rec\_to\_v}, l, m, tid)$. In both sub-machines with correct recipient, this output only occurs after the correct recipient input $(\mathsf{receive}, s, l, tid)$ and either a correct sender input $(\mathsf{send}, r, l, m, tid)$ or else the adversary input $(\mathsf{adv\_send}, m)$ to this sub-machine. As a consequence, on input stop, the sub-machine will output $(\mathsf{received}, m, tid)$ containing the same message m that is output to the verifier.

Verifiability of Valid Receipts (R. 4.2d): If the trusted host outputs (sent, tid) to a correct sender $s \in H_S$ on input $(\mathsf{send}, r, l, m, tid)$, then the corresponding sub-machine is from then on always in state $\mathsf{received}$ or showing

[18]Note that this restriction corresponds to our former assumption that the network is reliable. However, it is only needed for proving "Correct Execution".

and will output $(\mathsf{received}, s, r, l, m, tid)$ on input (show, tid) in both states. Note that subsequent inputs of (show, tid) lead to subsequent outputs of $\mathsf{received}$ at v.

No Surprises for the Recipient (R. 4.2e): If a verifier output $(\mathsf{received}, s, r, l, m, tid)$ then there exists a $\mathsf{th}(s, r, i)$ that output $(\mathsf{rec\_to\_v}, l, m, tid)$ for the stored m and l. From any state where such an output is possible, both sub-machines (for correct and incorrect sender) can no longer produce an output $(\mathsf{failed\_for\_r}, tid)$, i.e., the recipient will not output (failed, tid).

Secrecy of the Message (R. 4.2f): If a correct sender s obtains an output (failed, tid) on input $(\mathsf{send}, r, l, m, tid)$, then the corresponding sub-machine $\mathsf{th}(s, r, i)$ in slot $tidstate(s, tid)$ output $(\mathsf{failed\_for\_s}, tid)$. Thus, it does not output $(\mathsf{adv\_msg}, m)$, $(\mathsf{received}, m, tid)$, or $(\mathsf{rec\_to\_v}, l, m, tid)$ (these are the only outputs revealing information about the message). Thus, the recipient or adversary do not obtain knowledge about the message since this particular message is only known by this particular sub-machine.

Termination on Synchronous Network (R. 4.2g): After an input of $(\mathsf{send}, r, l, m, tid)$ or $(\mathsf{receive}, s, l, tid)$ with a fresh tid, the trusted host sends a stop signal at time $\Delta_{send} + i$ to each sub-machine started at time i. Since this leads to an output in all states of a running protocol, the "send" protocol terminates in time Δ_{send}. Additionally, the verification of a receipt requires at most time 1 in all machines.

■

By proving that the labeled certified mail scheme defined by Scheme 9.5 is as secure as the trusted host, we will almost show that it is a secure labeled certified mail scheme: Unlike the trusted host, it fulfills the requirements of Def. 4.1 by construction. The requirements from Def. 4.2, except for "Secrecy of the Message" are integrity requirements in the sense of Section 8.3.5. Hence Lemma 9.1 and Lemma 8.7 imply that the real system also fulfills these requirements.

Remark 9.6. This trusted host defines a fixed trust model: In any given system with some corrupt players, the behavior for the correct players is defined by the trusted host. Thus, we did not express restrictions on the trust model as in the requirement "limited trust in T" [R. 4.3b]. However, this could be done by a separate trusted host modeling the behavior if T is corrupted.

Remark 9.7. Since there exists no notion of a third party, a trusted-host-based specification cannot be used to specify the notion of "optimistic protocols." In other words, the main aspect of optimism, that no messages are exchanged with the third party under certain conditions, is a structural requirement on the real system. ○

9.2.2 A Provably Secure System

We now describe a concrete system that is as secure as the ideal system defined above. The protocol is shown in Figure 9.11 and a more precise timing diagram in Figure 9.15. The scheme is synchronous (as everything in this chapter), with two-party verification, and optimistic on disagreement, and it needs 4 messages in time 6 in the optimistic case.

We need four cryptographic primitives. The first is a signature scheme, where we use the notation as in Section 9.1.2. We tacitly assume that its message space is sufficiently large to allow for all our messages to be signed.

The second primitive is a one-way function $\mathcal{F} : \{0,1\}^* \to \{0,1\}^*$, i.e., the probability that a polynomial-time adversary is able to guess m given $\mathcal{F}(m)$ for a randomly chosen $m \overset{\mathcal{R}}{\leftarrow} \{0,1\}^k$ is negligible in k. To compute a one-time signature, a value $r \overset{\mathcal{R}}{\leftarrow} \{0,1\}^k$ (k is the given security parameter) will be chosen, $\mathcal{F}(r)$ will be signed (this is the authentication of a one-time public key), and later r serves as a signature under an a-priori known message.

The third primitive is a chameleon commitment scheme [BrCC 88]. In an ordinary commitment scheme a recipient generates the key that enables a sender to fix a message while keeping it secret. Furthermore, the sender may later open the commitment to prove that it in fact contained a given message. Chameleon commitments provide the same service, except that the party generating the key of the scheme can take an opened commitment on a message m and show how it could instead have been opened to an arbitrary message m'. For concreteness, we use the scheme from [BoCP 88, ChHP 92, Pede 92] with a “chameleon” extension. One party randomly chooses a a k-bit prime q and a $k'(k)$-bit prime p where q divides $p-1$ ($k'(k) > k$ denotes a sufficiently large security parameter of the scheme). Then it randomly selects a generator g of the unique subgroup G_q of order q in $\mathbb{Z}_p^*$ and $x \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_q^*$. It makes p, q, g, and $h := g^x$ public. A commitment on a message $m \in \mathbb{Z}_q$ is denoted by $\mathrm{Com}(m,r) := g^m h^r \bmod p$, where $r \in \mathbb{Z}_q$ should be randomly chosen. Using the secret key x and given m and r, one can open $\mathrm{Com}(m,r)$ to reveal any other $m' \in \mathbb{Z}_q$ by setting

$$r' := (m - m')/x + r$$

because $g^{m'}h^{r'} = g^{m'+xr'} = g^{m'+(m-m')+xr} = g^m h^r$. The resulting commitment scheme is perfectly hiding and computationally binding.[19]

The fourth primitive is a collision-resistant hash-function $\mathcal{H} : \{0,1\}^{\leq len} \to \{0,1\}^{k-1}$, i.e., the probability that a polynomial-time adversary is able to compute $m' \neq m$ with $\mathcal{H}(m') = \mathcal{H}(m)$ is negligible in k. It is used for computing images of the input message. Provably secure hash-functions (e.g., [ChHP 92]) usually need a key. In our scheme, this key will be chosen by the third party.

Scheme 9.5 (Reactive Certified Mail)
Let a message space $Msg \subseteq \{0,1\}^{\leq len}$, a label space $L \subseteq \{0,1\}^{\leq len}$, a set of transaction identifiers $TIDs \subseteq \{0,1\}^{\leq len}$, a round number $rnds$, and numbers n_S, n_R be given as in Definition 9.4 (in the sequel, we usually omit the security parameter k).

We mainly describe a set of machines $\{\mathsf{M_u} | u \in M \cup \{t\}\}$ with $t := n+1$ for $n := n_S + n_R + 1$, i.e., one machine for each user and additionally a machine $\mathsf{M_t}$ for a third party. The system Sys^{LCM} is the set of all structures

$$(M(H), G(H), s^*(H))$$

[19]In order to break the binding property of a commitment $co = g^m h^r$ without the secret key x, an adversary is required to find another representation $co = g^{m^*} h^{r^*}$ with $m^* \neq m$, which is computationally hard without knowing the secret key x.

for any set $H \subseteq M$ of correct users, where $M(H) = \{\mathsf{M_u} | u \in H\}$, and $G(H)$ and $s^*(H)$ are defined below.

The machines $\mathsf{M_u}$ are of four types: one for senders ($u \in M_S$), one for recipients ($u \in M_R$), one for the verifier ($u = v$) and one for the third party ($u = t$). Let $A = M - H$ be the set of the corrupted players.

Ports and graph: Each machine $\mathsf{M_u}$ has a pair $(in(u), out(u))$ of in- and output ports to its user (called user u), one broadcast output channel $out_0(u)$[20], and a list of input channels $in_0(v, u)$ for each other machine $v \in M$. Furthermore, it has a pair $(netw\_in(w, u), netw\_out(u, w))$ of in- and output ports for exchanging messages with $\mathsf{M_w}$ for each $w \in M$. We define $s^*(H) := \{in'(u), out'(u)\}_{u \in H}$. The verifier $\mathsf{M_v}$ has no user-input channel $in(u)$ while machine $\mathsf{M_t}$ has no user ports at all.

The connection graph $G(H)$ defines authentic broadcast channels for key exchange. Channels with the verifier and third party are assumed to be reliable. All other connections are left to the discretion of the adversary: Each key exchange output port $out_0(u)$ is connected to a free output port $out'_0(u)$ as well as to all corresponding input ports $in_0(u, w)$ for $w \in M(H)$, i.e., the adversary can listen to, but not modify the key exchange channels between correct players. Each reliable output port $netw\_out(u, w)$ with $u \in \{t, v\}$ or $w \in \{t, v\}$ is connected to a replicated free port $netw\_out'(u, w)$ as well as the corresponding input port $netw\_in(u, w)$. All other ports $netw\_out(s, r)$ and $netw\_in(s, r)$ are connected to ports $netw\_out'(s, r)$ and $netw\_in'(s, r)$ that are in $\mathsf{free}(G(H), M(H))$. Similarly, $G(H)$ connects $in'(u)$ to $in(u)$ and $out(u)$ with $out'(u)$. As before, we omit the primes in the sequel.

Initialization (Round 0): The third party generates the parameters q, p, g, x, h of the chameleon commitment scheme as well as a hash-function $\mathcal{H}$. It broadcasts q, p, g, h, and $\mathcal{H}$. Each machine generates signature keys ($\mathrm{sign}_{\mathsf{u}}, \mathrm{test}_{\mathsf{u}}$) and broadcasts $\mathrm{test}_{\mathsf{u}}$.

Each sender $\mathsf{M_s}$ with $s \in H_S$ initializes the following variables, where $Slots_S := M_R \times \{1, \ldots, rnds\}$:

$$\begin{aligned} tidstate_s &: TIDs \rightarrow \{\epsilon\} \cup Slots_S; \\ slotstate_s &: Slots_S \rightarrow Localstate. \end{aligned}$$

As the sender can start a "send" protocol with each recipient in each round, this corresponds to the "slots" or opportunities for such protocol runs. Initially, $tidstate_s$ contains ϵ everywhere. $Localstate$ is the set of possible states of each protocol sub-machine $\mathsf{cm\_s}_s(r, i)$. Each slot-state is initialized with state $ps_0(r, i)$.[21] All protocol sub-machines $\mathsf{cm\_s}_s(r, i)$ store the slot as a parameter.

[20]Unlike secure channels, we now also require consistency as guaranteed by the multicast channels in order to ensure that the verifier and all recipients obtain the same public keys.

[21]These matrices will usually be sparsely populated. In practice, the state of the few sub-machines in non-starting states can either be stored in a database or else by spawning sub-processes who keep the corresponding state.

Each recipient $\mathsf{M_r}$ with $r \in H_R$ initializes the following variables, where $\mathit{Slots}_R := M_S \times \{1, \dots, \mathit{rnds}\}$:

$$\begin{aligned} \mathit{tidstate}_r &: \mathit{TIDs} \rightarrow \{\epsilon\} \cup \mathit{Slots}_R; \\ \mathit{slotstate}_r &: \mathit{Slots}_R \rightarrow \mathit{Localstate}. \end{aligned}$$

Initially, $\mathit{tidstate}_r$ contains ϵ everywhere. All states store the slot as a parameter. Thus, each slot-state is initialized with state $\mathit{pr}_0(s, i)$.

Sender Input Dispatching: Each sender obtains a vector $\mathit{in}_i(s)$ with $\mathit{in}_i(s) = (\mathit{inS}_i(s, r) | r \in M_R)$ from its user. For $\mathit{inS}_i(s, r)$ = $(\mathsf{send}, r, l, m, \mathit{tid})$, the sender machine verifies that tid is fresh for $\mathsf{M_s}$, i.e., $\mathit{tidstate}_s(\mathit{tid}) = \epsilon$. If yes, it sets $\mathit{tidstate}_s(\mathit{tid}) = (r, i)$ and inputs $(\mathsf{send}, r, l, m, \mathit{tid})$ to the sub-machine $\mathsf{cm\_s}_s(r, i)$.

For $\mathit{inS}_i(s, v)$ = $(\mathsf{show}, \mathit{tid})$, it looks up $\mathit{slot} := \mathit{tidstate}_s(\mathit{tid})$. If this is ϵ, the input is ignored. Else, it inputs show to the sub-machine $\mathsf{cm\_s}_s(\mathit{slot})$.

For each message msg received at $\mathit{netw\_in}(r, s)$ that contains a tid, the tuple $(\mathsf{msg}, \mathit{msg})$ is input to $\mathsf{cm\_s}_s(\mathit{tidstate}_S(\mathit{tid}))$.

Recipient Input Dispatching: Each recipient obtains a vector $\mathit{in}_i(r)$ with $\mathit{in}_i(r) = (\mathit{inR}_i(r, s) | s \in M_S)$ from its user. For $\mathit{inR}_i(r, s)$ = $(\mathsf{receive}, s, l, \mathit{tid})$, it checks whether tid is fresh for $\mathsf{M_r}$, i.e., $\mathit{tidstate}_r(\mathit{tid}) = \epsilon$. If yes, it sets $\mathit{tidstate}_r(\mathit{tid}) := (s, i)$ and inputs $(\mathsf{receive}, s, l, \mathit{tid})$ to $\mathsf{cm\_r}_r(s, i)$.

For each message msg received at $\mathit{netw\_in}(s, r)$ that contains a tid, the tuple $(\mathsf{msg}, \mathit{msg})$ is input to $\mathsf{cm\_r}_r(\mathit{tidstate}_R(\mathit{tid}))$.

Input Dispatching of $\mathsf{M_t}$ *and* $\mathsf{M_v}$*:* The third party and the verifier do not have matrices of sub-machines. Instead, they process received messages independently and in parallel.

The “send“*-Protocol:* A protocol run is depicted in Figure 9.11. The individual sub-machines are depicted in Figures 9.12, 9.13, and 9.14. The timing is depicted in Figure 9.15. The detailed behavior is as follows:

Machine $\mathsf{cm\_s_s}(r, i)$*:* On input of $(\mathsf{send}, r, l, m, \mathit{tid})$ from its super-machine $\mathsf{M_s}$, a sender sub-machine $\mathsf{cm\_s}_s(r, i)$ chooses $r_S \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_q$, sets $d_S := (s, r, l, \mathit{tid}, i)$, computes $m_1 \leftarrow \mathrm{sign}_{\mathsf{s}}(d_S, \mathsf{Com}(\mathcal{H}(m), r_S))$[22] to be sent to r and outputs (msg, r, m_1)[23]. On input of a correct tuple (msg, m_2) in Round $i + 2$, the machine computes $m_3 := (\mathit{tid}, r_S, m)$ and outputs (msg, r, m_3). Otherwise, it waits for the protocol to end. If it output m_3 and does not receive a correct input (msg, m_4) in Round i+4, it executes the sub-protocol “resolve“.

After 6 rounds, it outputs $(\mathsf{sent}, \mathit{tid})$ to the super-machine if it received m_2 and else $(\mathsf{failed}, \mathit{tid})$. (Note that it will receive m_4 or m_6 since the connection to T is reliable.)

[22]In the sequel, we tacitly assume that signed message can unambiguously be distinguished. This can, e.g., be guaranteed by signing a message identifier +m_i+ together with the contents of m_i.

[23]This is a message envelope that signals to the super-machine that message m_1 shall be sent to machine $\mathsf{M_r}$.

Machine $\mathsf{cm\_r_r}(s,i)$*:* Upon receipt of an input $(\mathsf{receive}, s, l, tid)$ from its super-machine $\mathsf{M_r}$, a recipient sub-machine $\mathsf{cm\_r}_r(s,i)$ waits for an input (msg, m_1) and verifies that the parameters are as expected. Otherwise, it waits. If m_1 was correct, it computes $m_2 \leftarrow \mathrm{sign}_\mathsf{r}(m_1, \mathcal{F}(r_R))$ with $r_R \overset{\mathcal{R}}{\leftarrow} \{0,1\}^k$ and outputs (msg, s, m_2). Then, it waits for a correct (msg, m_3) in Round 3. If this input is not received, it waits for a correct input (msg, m_6) in Round $i+6$. If (msg, m_3) is received, it computes $m_4 := (tid, r_R)$, outputs (msg, s, m_4), and outputs $(\mathsf{received}, m, tid)$ at time $i+6$.

If m_3 was not received, it waits for an input (msg, m_6) in Round $i+6$. If a correct m_6 is input, it outputs $(\mathsf{received}, m, tid)$ and (failed, tid), else.

Sub-protocol “resolve“: Machine $\mathsf{cm\_s}_s(r,i)$ computes $m_5 := (m_1, m_2, m_3)$ and outputs (msg, t, m_5). Upon receiving m_5, $\mathsf{M_t}$ verifies it and verifies that the round number in d_S as contained in m_1 is $i-5$. If these checks succeed, it sends $m_6 \leftarrow \mathrm{sign}_\mathsf{t}(m_5)$ to $\mathsf{M_s}$ and $\mathsf{M_r}$. Else, it aborts. (Note that we assume that machine $\mathsf{M_t}$ can process multiple requests in parallel, i.e., we do not elaborate on its sub-machines.)

The “show“*-Protocol:* On input (show, tid) from the super-machine, a sender machine $\mathsf{cm\_s}_s(r,i)$ in state $\mathsf{received}$ outputs either $(\mathsf{msg}, v, (m_1, m_2, m_3, m_4))$ while it outputs (msg, v, m_6) in state $\mathsf{received}'$.

Upon receipt of one of these receipts, the verifier machine $\mathsf{M_v}$ outputs $(\mathsf{received}, s, r, l, m, tid)$ if these messages are correct while using the parameters as fixed in m_1 as well as the message from m_3. (Again, we assume that $\mathsf{M_v}$ can process multiple requests in parallel; however multiple receipts for one tid of the same participants only lead to one output of $\mathsf{received}$.)

Output Dispatching: All outputs of multiple values at one port are arranged in sets. For each tuple (msg, w, msg) with $w \in M$ that is output by a sub-machine, the super-machine $\mathsf{M_u}$ adds an element msg to the set of messages to be output at $netw\_out(u,w)$. All other outputs by sub-machines are output at $out(u)$.

All outputs of $\mathsf{M_v}$ of parallel executions of “show“ in the current round are output as a set $out(v)$.

$\diamond$

Remark 9.8. This scheme does not fulfill Requirements 4.3a and 4.3c from Definition 4.3: It is clear that in the described scheme where the third party chooses the commitment scheme, Requirement 4.2c does not hold if the third party misbehaves: Using the secret key x, a sender colluding with the third party can change the receipt such that it matches any message. Therefore, the requirement “limited trust in the third party” is not fulfilled for this scheme. $\circ$

9.2.3 The Security Proof

We now prove the security of Scheme 9.5 with respect to the trusted host defined in Def. 9.4. The overall structure of the proof is similar to the proof for

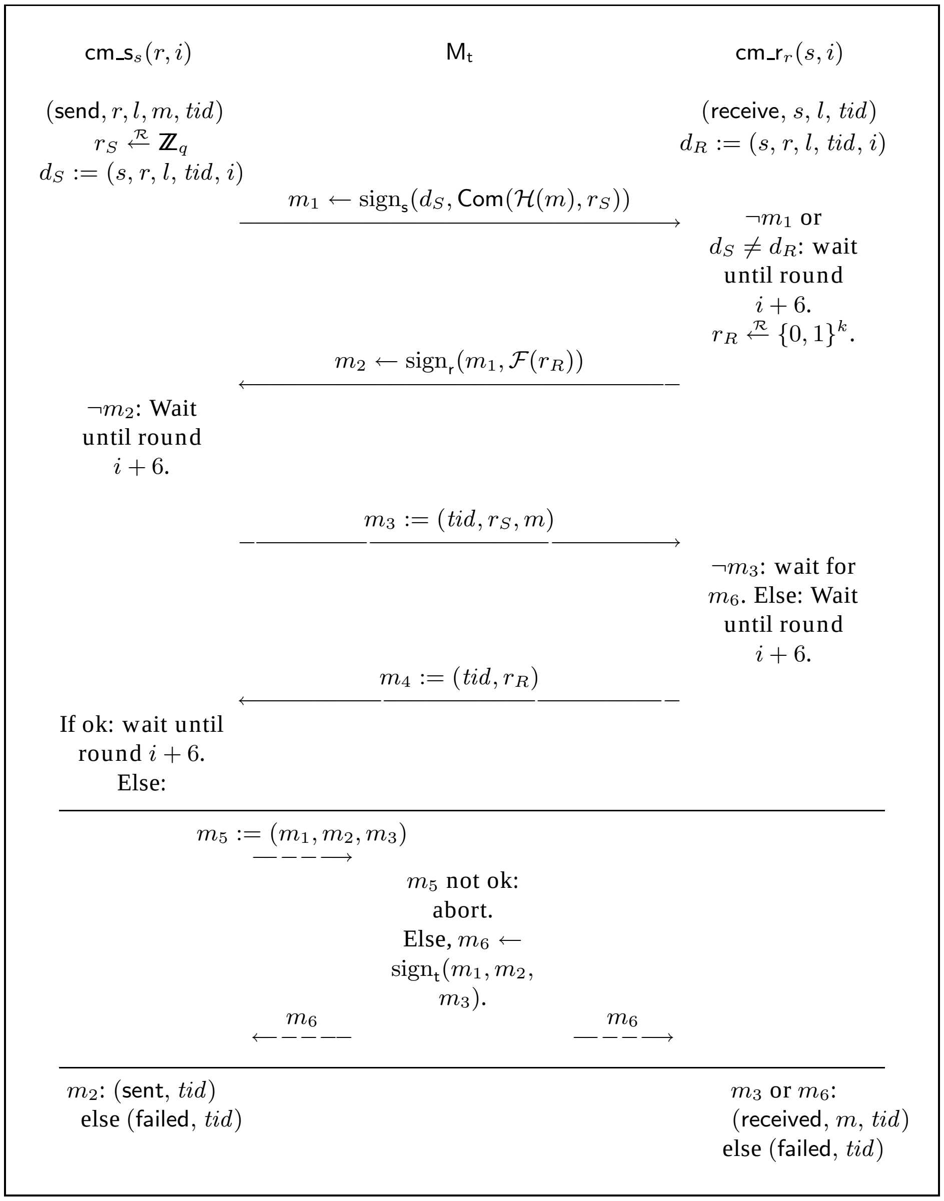

$\mathsf{cm\_s}_s(r, i)$
$\mathsf{M_t}$
$\mathsf{cm\_r}_r(s, i)$
$(\mathsf{send}, r, l, m, tid)$
$r_S \xleftarrow{\mathcal{R}} \mathbb{Z}_q$
$d_S := (s, r, l, tid, i)$
$(\mathsf{receive}, s, l, tid)$
$d_R := (s, r, l, tid, i)$
$m_1 \leftarrow \mathrm{sign}_{\mathsf{s}}(d_S, \mathsf{Com}(\mathcal{H}(m), r_S))$
$\neg m_1$ or $d_S \neq d_R$: wait until round $i + 6$.
$r_R \xleftarrow{\mathcal{R}} \{0, 1\}^k$.
$m_2 \leftarrow \mathrm{sign}_{\mathsf{r}}(m_1, \mathcal{F}(r_R))$
$\neg m_2$: Wait until round $i + 6$.
$m_3 := (tid, r_S, m)$
$\neg m_3$: wait for m_6. Else: Wait until round $i + 6$.
$m_4 := (tid, r_R)$
If ok: wait until round $i + 6$. Else:
$m_5 := (m_1, m_2, m_3)$
m_5 not ok: abort. Else, $m_6 \leftarrow \mathrm{sign}_{\mathsf{t}}(m_1, m_2, m_3)$.
m_6
m_6
m_2: (sent, tid) else (failed, tid)
m_3 or m_6: $(\mathsf{received}, m, tid)$ else (failed, tid)

Figure 9.11: Scheme 9.5 for Synchronous Labeled Certified Mail (dashed arrows are only needed in case of exceptions).

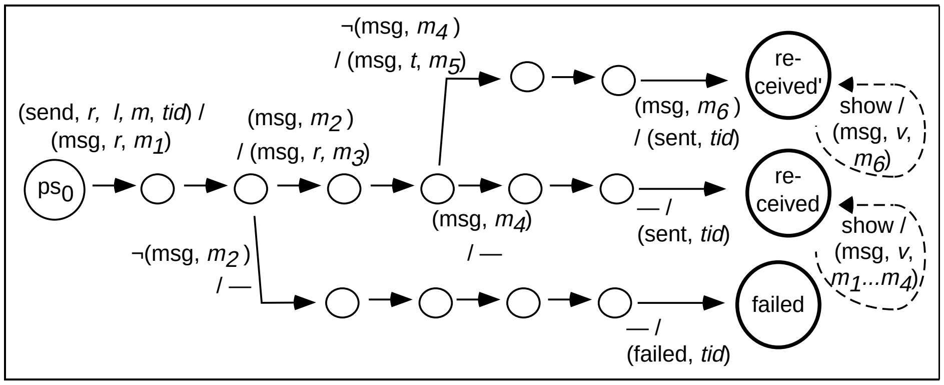


Figure 9.12: Sender for Certified Mail (Sub-Machine cm_s$_s(r, i)$; unlabeled transitions are triggered by time).

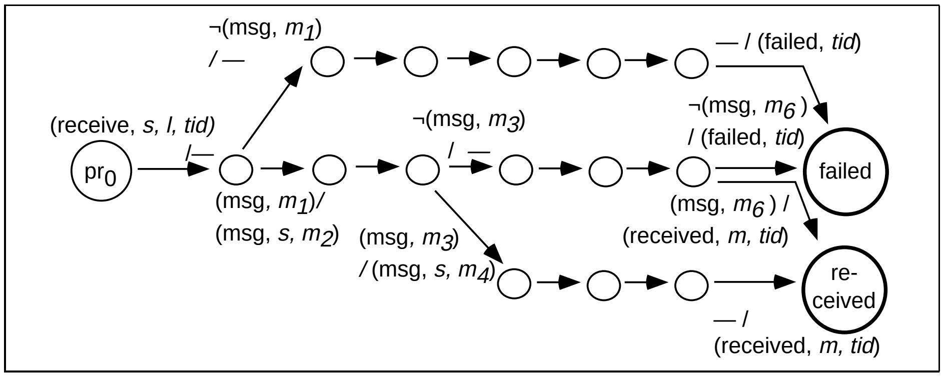


Figure 9.13: Recipient Machine for Certified Mail (Sub-Machine cm_r$_r(s, i)$).

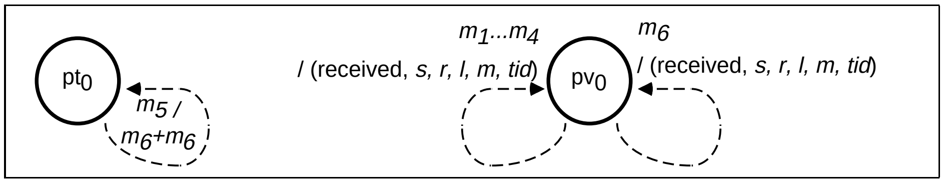

pt0
m5 / m6+m6
m1...m4 / (received, s, r, l, m, tid)
pv0
m6 / (received, s, r, l, m, tid)

Figure 9.14: Third Party and Verifier Machines for Certified Mail.

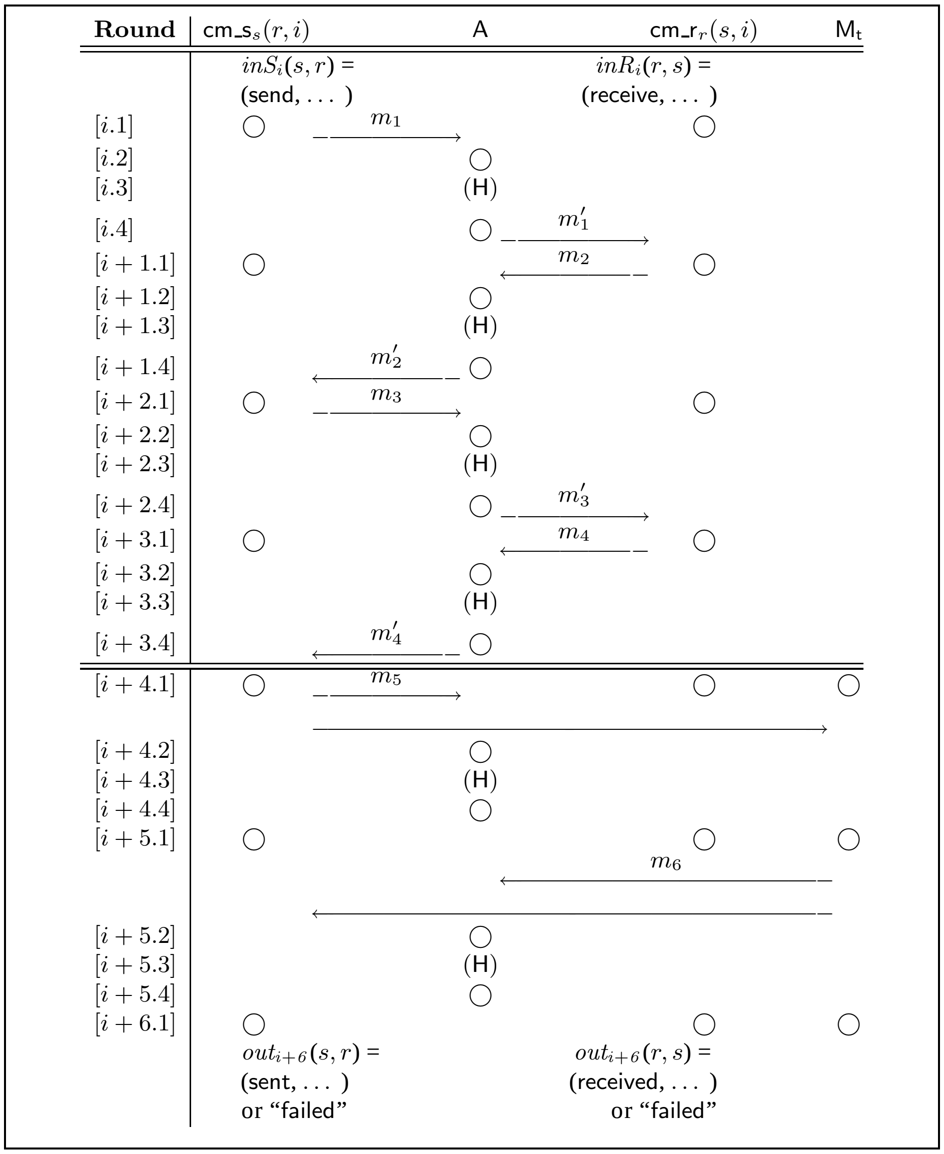


Figure 9.15: Timing of the “send“ Protocol of Scheme 9.5 for Correct Submachines cm_s$_s(r,i)$ and cm_r$_r(s,i)$. (Arrows to A correspond to messages $netw\ldots$, arrows from A to messages $netw'\ldots$. ◯ denotes switching of the machine in this column, (H) denotes switching of H.)

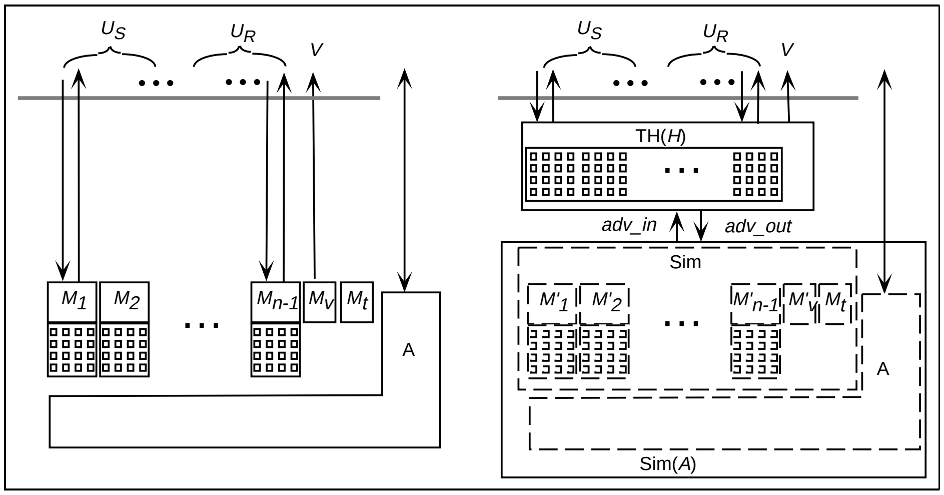


Figure 9.16: Structure of the System and its Simulation (the matrices of tiny squares depict the slots with sub-machines).

secure channels, i.e., for each set $H \subseteq M$ of correct participants and any given adversary A for this system, we make a blackbox-reduction without rewinding.

The main cryptographic aspect of the simulation of the system in the simulation of a message m_1: At the time m_1 shall be sent, the trusted host only outputs a busy-signal without revealing the actual message. However, our simulator has to give a correct-looking network message to A, which includes a commitment that is supposed to fix the message m to be sent. If the protocol run is successful, it has to open this commitment two rounds later. If it then reveals a message $m' \neq m$, i.e., not the input of the honest user to TH, the simulation is not correct: e.g., an honest recipient may not get the message an honest sender sent. Here is why we needed the chameleon property: It allows the simulator (who also only simulates the machine $\mathsf{M_t}$ and thus knows its key) to first make the commitment on an arbitrary message m_{sim}, and later open it to the correct message m.

We now define a corresponding adversary Sim(A) on the trusted-host system. It consists of a simulator Sim that simulates all machines $\mathsf{M_u}$ of the system and any given adversary A. The machine Sim as described below simulates the behavior of correct real machines by machines $\{\mathsf{M'_u} | u \in H\}$ and $\mathsf{M_t}$ and $\mathsf{M_v}$ using the information obtained at machine TH in order to give the simulated A its expected environment (see Figure 9.16). In order to enable a modular proof, the main aspects of the simulation are handled by sub-machines cm_s′ and cm_r′ that simulate the sub-machines cm_s and cm_r of the real protocol. However, instead of processing dispatched inputs from the user H, the simulated machines interact with the sub-machines th in the corresponding slot of TH.

Scheme 9.6 (Simulation of Labeled Certified Mail)
Let a set $H \subseteq M$ of correct participants be given. Let $A := M - H$.

We now define an adversary Sim(A). It consists of a simulator Sim interacting with A and TH. Sim behaves as follows:

Ports and Graph: The ports and graph between the simulated machines $\mathsf{M'_u}$ $u \in$

H are identical to the real system. In addition, the simulator Sim has two ports adv_in' and adv_out' for interacting with TH.

Initialization (Round 0*):* In Round 0, the behaviour and key exchange of all simulated machines M'_u is identical to their real counterparts from Scheme 9.5. The sub-machines of a sender and a recipient are denoted with $\mathsf{cm\_s}'(r,i)$ and $\mathsf{cm\_r}'(s,i)$, respectively.

Input Dispatching: The dispatching of network inputs is not changed. Each input at port adv_out' is dispatched as follows using a fixed message $m_{sim} \in Msg$:

Input at adv_out':	Input:	at:
$(s, r, i, (\mathsf{adv\_busy}, s, l, tid))$	$(\mathsf{send}, r, l, m_{sim}, tid)$	M'_s
$(s, r, i, (\mathsf{adv\_busy}, r, l, tid))$	$(\mathsf{receive}, s, l, tid)$	M'_r
$(s, r, i, (\mathsf{adv\_busy}, v, l, tid))$	(show, tid)	M'_s
$(s, r, i, (\mathsf{adv\_msg}, m))$	$(\mathsf{adv\_msg}, m)$	$\mathsf{cm\_s}'_s(r,i)$ in M'_s

Machines M'_s and M'_r then dispatch these inputs to their sub-machines as usual.

Sender Sub-machine $\mathsf{cm\_s}'_s(r,i)$*:* A state transition in Round $[i.1]$ of the real machines is usually simulated in two parts: In $[i-1.4]$ the simulation verifies incoming messages and sends requests to the trusted host. In $[i.2]$, the answers of the trusted host are received and the resulting messages are sent.

The sender sub-machine $\mathsf{cm\_s}'_s(r,i)$ simulates a correct sender sub-machine $\mathsf{cm\_s}(r,i)$, except for the following changes (see Figure 9.17 and 9.18):

Round [i.2]: The sender sub-machine computes $m_{1,sim}$ like m_1 with the input message m_{sim}. With r_{sim}, we denote the random number used for computing the commitment $\mathrm{Com}(\mathcal{H}(m_{sim}), r_{sim})$.

Round [i+1.4]: Upon input of a correct (msg, m_2), it outputs $\mathsf{adv\_receive}$ (that will be dispatched to $adv\_in'(s,r,i)$) to obtain the message m. Otherwise, it outputs $\mathsf{adv\_suppress}$.

Round [i+2.2]: Upon input of $(\mathsf{adv\_msg}, m)$, the sub-machine computes $m_{3,sim} := (tid, r'_S, m)$ using the message m and $r'_S := (\mathcal{H}(m_{sim}) - \mathcal{H}(m))x^{-1} + r_{sim} \mod q$, which is computed using the secret key x of the simulated machine T. This choice of r'_S guarantees that $\mathrm{Com}(\mathcal{H}(m), r'_S) = \mathrm{Com}(\mathcal{H}(m_{sim}), r_{sim})$ using the chameleon property of the commitment scheme.

If $(\mathsf{adv\_msg}, m)$ is not input after outputting $\mathsf{adv\_receive}$, the sub-machine stops prematurely.

Round [i+6.2]: The sub-machine does not output its final result.

No changes are made to the behavior of the sender in the “show“-protocol.

Recipient Sub-machine $\mathsf{cm\_r}'_r(s,i)$*:* This machine simulates a correct recipient sub-machine $\mathsf{cm\_r}_r(s,i)$, except for the following changes (see Figure 9.17 and 9.18):

Round [i.4] If no correct $(\mathsf{msg}, m_{1,sim})$ was input, the sub-machine outputs $\mathsf{adv\_suppress}$.

Round [i+2.4] After the input of a correct $(\mathsf{msg}, m_{3,sim})$, it outputs $(\mathsf{adv\_send}, m)$.[24]

Round [i+6.2]: The sub-machine does not output its final result.

Third Party M'_t: The third party M'_t is simulated with the following change:

Round [i+4.4] The third party verifies the input message m_5 and decides whether it will send m_6 at time $[i+5.2]$ as usual. If yes, it outputs $((s,r,i), (\mathsf{adv\_send}, m))$ for the tuple (s,r,i) as included in m_1.

Verifier M'_v: The simulated verifier M'_v verifies input messages like M_v. For each correct message, it retrieves s, r and the starting round i of the corresponding protocol run from m_1 and outputs $((s,r,i), \mathsf{adv\_show})$.

Output Dispatching: Each super-machine M'_u with $\mathsf{u} \notin \{\mathsf{v}, \mathsf{t}\}$ dispatches all messages (i.e., tuples starting with msg) without changes. All other outputs out_{slot} of a sub-machine in slot $slot$ are output as $(slot, out_{slot})$ to Sim to be forwarded to TH. Non-network outputs of M'_t and M'_v are output to Sim.

The simulator Sim collects all adversary outputs out_{slot} for each slot (s,r,i) and outputs them as a set $adv\_in'((s,r,i))$. Adversary outputs are $((s,i), out_{slot})$ from M'_r with $r \in H_R$, $((r,i), out_{slot})$ from M'_s with $s \in H_S$, $((s,r,i), out_{slot})$ from M'_t, and $((s,r,i), out_{slot})$ from M'_v.

◇

Theorem 9.2
Let a message space $Msg \subseteq \{0,1\}^{\le len}$*, a label space* $L \subseteq \{0,1\}^{\le len}$*, a set of transaction identifiers* $TIDs \subseteq \{0,1\}^{\le len}$*, a round number* $rnds \in \mathbb{N}$*, and numbers* n_S, n_R *be given as in Definition 9.4. Let* $\Delta_{send} := 6$*.*

Then

$$Sys^{LCM} \ge_{sec}^{f_{cm},poly} TH^{LCM}$$

holds for Sys^{LCM} *as defined in Scheme 9.5,* TH^{LCM} *as defined in Scheme 9.4, and*

$$f_{cm}((M(H), G(H), s^*(H))) := (\{\mathsf{TH}(H)\}, G_{TH}(H), s^*(H)).$$

□

Proof. We show for any given configuration ($M(H)$, $G(H)$, $s^*(H)$, H, A, G_{AH}) and the corresponding configuration ($TH(H)$, $G_{TH}(H)$, $s^*(H)$, H, Sim(A), G_{AH}) based on the simulator Sim(A) as defined in Scheme 9.6 that the families of distributions of the view of H are polynomially indistinguishable unless H has ports from $\mathsf{free}(G_{TH}(H), \{TH(H)\}) - s^*(H)$.

[24]This input enables the adversary to start verification at time $[i+3.2]$ in case of $s \in A_S$, even though correct players have to wait for the end of the protocol.

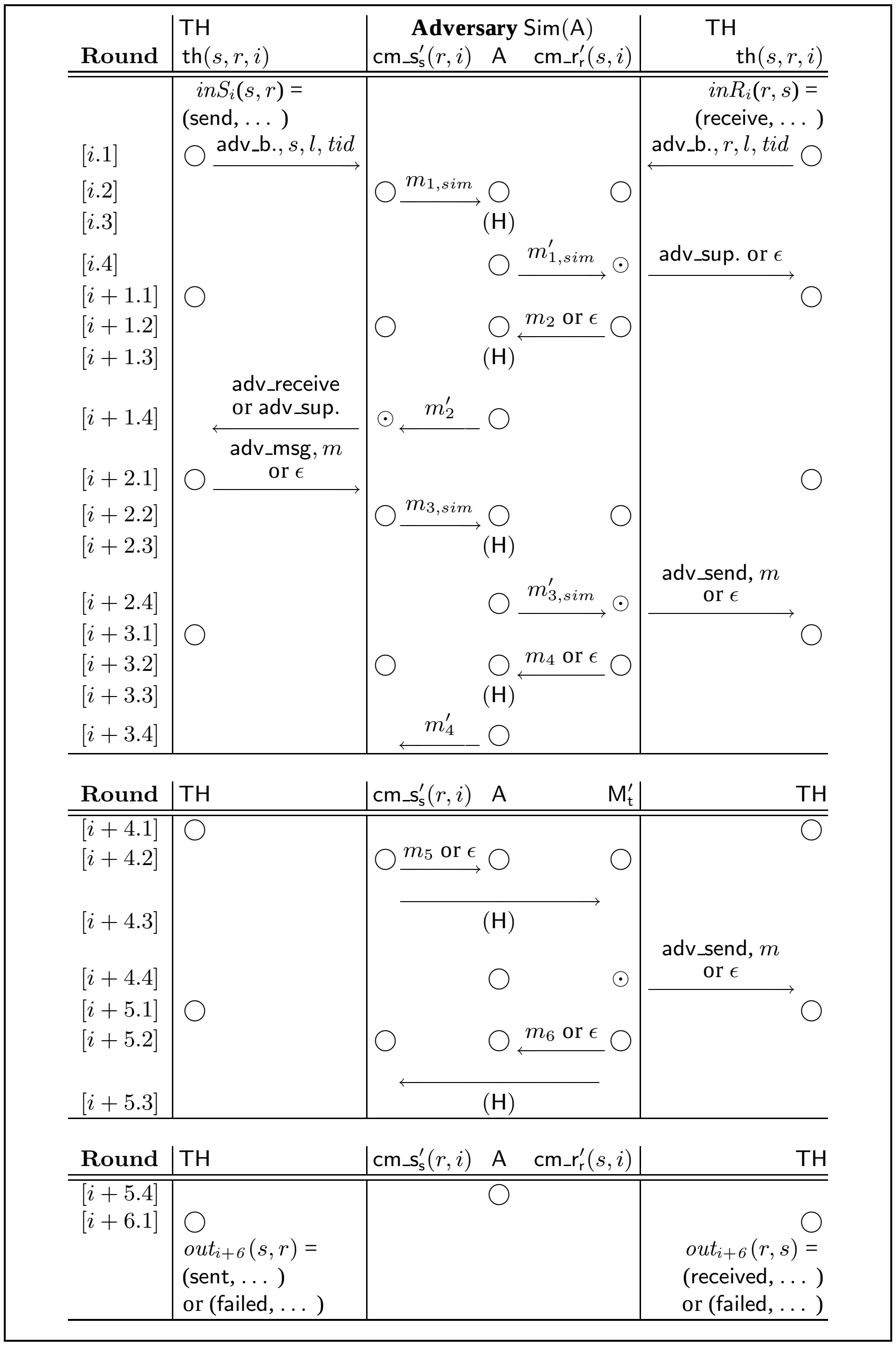


Figure 9.17: Simulation of the Labeled Certified Mail Protocol. (The symbol $\odot$ denotes additional state transitions not in the protocol; primed messages denote messages that may have been changed by the adversary).

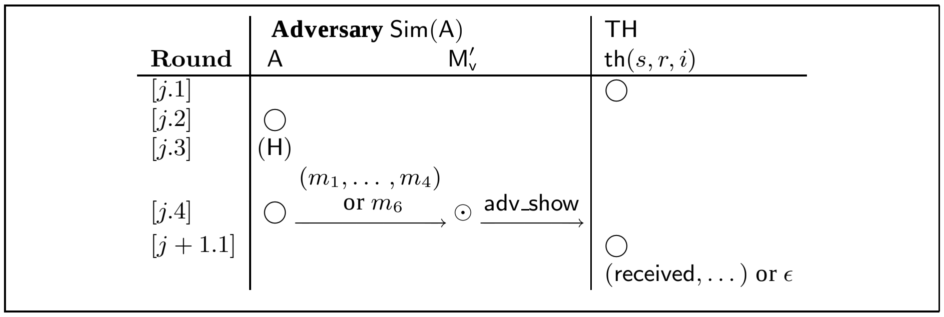


Figure 9.18: Simulation of the Verification for Labeled Certified Mail.

Defining $(\mathbf{s}, \mathbf{r}, \mathbf{i})$-Systems: We now define sub-systems containing all sub-machines that handle a given slot (s, r, i) of TH.

Third Party: $\mathsf{M_t}$ and $\mathsf{M'_t}$ process any network input only if it includes correct signed messages m_1 and m_2 with the same (s, r, i). All other inputs are ignored. We denote the behavior for a given triple (s, r, i) with $\mathsf{cm\_t}(s, r, i)$ and $\mathsf{cm\_t'}(s, r, i)$, respectively.

Verifier: The verifier $\mathsf{M_v}$ outputs $(\mathsf{received}, s, r, l, m, \mathit{tid})$ only after processing correct network inputs m_1 and m_2 including the same (s, r, i) (either included in the tuple $(m_1, \ldots, m_4)$ or in m_6). All other inputs are ignored, and similarly by $\mathsf{M'_v}$. We denote the behavior of for a given triple (s, r, i) with $\mathsf{cm\_v}(s, r, i)$ and $\mathsf{cm\_v'}(s, r, i)$, respectively.

In the sequel, the sub-machines $\mathsf{cm\_s_s}(r, i)$, $\mathsf{cm\_r_r}(s, i)$, $\mathsf{cm\_t}(s, r, i)$, and $\mathsf{cm\_v}(s, r, i)$ are called (s, r, i)-system, while $\mathsf{cm\_s'_s}(r, i)$, $\mathsf{cm\_r'_r}(s, i)$, $\mathsf{th}(s, r, i)$, $\mathsf{cm\_t'}(s, r, i)$, and $\mathsf{cm\_v'}(s, r, i)$ are called (s, r, i)-simulation.

Correct Dispatching within $(\mathbf{s}, \mathbf{r}, \mathbf{i})$-Systems: We first show that all inputs at $s^*(H)$ or at the network in the simulation and the real system are either ignored or lead to identical inputs at the (s, r, i)-system and its simulation.

Then, we show that all outputs that are not output at $s^*(H)$ or the network are internal to each (s, r, i)-system and its simulation. In particular, they are not in A's and H's view.

Inputs at the User-Interface $s^*(H)$: An input of $(\mathsf{send}, r, l, m, \mathit{tid})$ to TH at port $in(s)$ with $s \in H_S$ in Round i is input to $\mathsf{cm\_s_s}(r, i)$ or $\mathsf{th}(s, r, i)$, respectively. An input $(\mathsf{show}, \mathit{tid})$ to TH at port $in(s)$ is ignored if this sender did not input $(\mathsf{send}, r, l, m, \mathit{tid})$ with the same tid before, and machine $\mathsf{M_s}$ ignores these inputs as well. Else, it is input to $\mathsf{cm\_s_s}(r, i)$ or $\mathsf{th}(s, r, i)$, respectively. An input of $(\mathsf{receive}, s, l, \mathit{tid})$ to TH at port $in(r)$ with $r \in H_R$ is input to $\mathsf{cm\_r_r}(s, i)$ or $\mathsf{th}(s, r, i)$, respectively. All other inputs at $s^*(H)$ are ignored.

Network Inputs: Network inputs are dispatched and transported to the (s, r, i)-system and its simulation without changes.

Visible Outputs: All network outputs $(\mathsf{msg}, r, \mathit{msg})$ are dispatched and output to A without changes in the system and its simulation. The outputs at $out(v)$ are collected from the outputs of the sub-machines of $\mathsf{M_v}$ and $\mathsf{M'_v}$ without changes.

We show now that all non-network outputs of the sub-machines of an (s, r, i)-system and all outputs of $\mathsf{th}(s, r, i)$ that are neither $\mathsf{adv\_busy}$ and $\mathsf{adv\_msg}$ correspond to each other and lead to identical outputs at the user interface

$s^*(H)$.

If $\mathsf{cm\_s_s}(r, i)$ outputs $(\mathsf{sent}, \mathit{tid})$, machine $\mathsf{M_s}$ outputs $(\mathsf{sent}, \mathit{tid})$ at $\mathit{out}(s)$. This output corresponds to the output $(\mathsf{sent}, \mathit{tid})$ at $\mathsf{th}(s, r, i)$, which is output by TH at $\mathit{out}(s)$ as well.

The output $(\mathsf{failed}, \mathit{tid})$ by $\mathsf{cm\_s_s}(r, i)$ leads to the same output at $\mathit{out}(s)$ of $\mathsf{M_s}$. This output corresponds to the output of $(\mathsf{failed\_for\_s}, \mathit{tid})$ by $\mathsf{th}(s, r, i)$, which leads to an output $(\mathsf{failed}, \mathit{tid})$ at TH as well.

The output $(\mathsf{received}, m, \mathit{tid})$ by $\mathsf{cm\_r_r}(s, i)$ leads to the same output at $\mathit{out}(r)$ of $\mathsf{M_r}$. It corresponds to the output $(\mathsf{received}, m, \mathit{tid})$ at $\mathsf{th}(s, r, i)$ which leads to the same output at TH.

The output $(\mathsf{failed}, \mathit{tid})$ by $\mathsf{cm\_r_r}(s, i)$ leads to an output $(\mathsf{failed}, \mathit{tid})$ at $\mathit{out}(r)$ of $\mathsf{M_r}$. It corresponds to the output $(\mathsf{failed\_for\_r}, \mathit{tid})$ by $\mathsf{th}(s, r, i)$, which leads to an output $(\mathsf{failed}, \mathit{tid})$ at $\mathit{out}(r)$ of TH as well.

The output $(\mathsf{received}, s, r, l, m, \mathit{tid})$ at $\mathsf{cm\_v}(s, r, i)$ is output at $\mathit{out}(v)$. It corresponds to $(\mathsf{rev\_to\_v}, l, m, \mathit{tid})$ by $\mathsf{th}(s, r, i)$, which leads to an output $(\mathsf{received}, s, r, l, m, \mathit{tid})$ at $\mathit{out}(v)$ as well.

Other Outputs are Internal: The only other outputs of the (s, r, i)-simulation that have not been considered so far are $\mathsf{adv\_busy}$, $\mathsf{adv\_msg}$, $\mathsf{adv\_send}$, $\mathsf{adv\_receive}$, $\mathsf{adv\_show}$, and $\mathsf{adv\_suppress}$.

The output $\mathsf{adv\_busy}$ by $\mathsf{th}(s, r, i)$ is forwarded as $\mathit{adv\_out}(s, r, i)$ to Sim which inputs the corresponding input command (see Page 149) to $\mathsf{M'_s}$ or $\mathsf{M'_r}$, respectively. These machines then initialize $\mathit{tidstate}(\mathit{tid})$ and input the command to the corresponding sub-machine.

An output $(\mathsf{adv\_msg}, m)$ by $\mathsf{th}(s, r, i)$ is dispatched via TH and Sim to $\mathsf{cm\_s'_s}(r, i)$.

An output $(\mathsf{adv\_send}, m)$ by $\mathsf{cm\_r'}_r(s, i)$ or $\mathsf{cm\_t'}(s, r, i)$ is dispatched to $\mathsf{th}(s, r, i)$.

An output $\mathsf{adv\_receive}$ by $\mathsf{cm\_s'_s}(r, i)$ is dispatched to $\mathsf{th}(s, r, i)$.

An output $\mathsf{adv\_show}$ by $\mathsf{cm\_v'}(s, r, i)$ is dispatched to $\mathsf{th}(s, r, i)$.

An output $\mathsf{adv\_suppress}$ by $\mathsf{cm\_s'_s}(r, i)$ or $\mathsf{cm\_r'_r}(s, i)$ is dispatched to $\mathsf{th}(s, r, i)$.

In the sequel, we usually omit the dispatching and use abbreviations like "$\mathsf{cm\_s'}(r, i)$ inputs $\mathsf{adv\_suppress}$ to $\mathsf{th}(s, r, i)$", even though this message is passed through machines $\mathsf{M'_s}$, Sim, and TH.

Correct Sender; Incorrect Recipient: Let us now consider a protocol execution of a given (s, r, i)-system and its simulation on corresponding inputs. We show that the visible behavior (i.e., on the network and at $s^*(H)$) is indistinguishable.

Let $s \in H_S$ and $r \in A_R$. We therefore compare the behavior of $\mathsf{th}(s, r, i)$, $\mathsf{cm\_s'_s}(r, i)$, $\mathsf{cm\_v'}(s, r, i)$, and $\mathsf{cm\_t'}(s, r, i)$ with $\mathsf{cm\_s}_s(r, i)$, $\mathsf{cm\_v}(s, r, i)$, and $\mathsf{cm\_t}(s, r, i)$. Before Round i, all sub-machines do nothing. If send is not input in Round i, all (s, r, i)-machines remain in their starting state forever without making any outputs. (Initially, neither $\mathsf{cm\_v}(s, r, i)$ nor $\mathsf{cm\_v'}(s, r, i)$ will output $\mathsf{received}$ for this s, r, tid for any fresh tid.)

Let us now consider the case that $(\mathsf{send}, r, l, m, \mathit{tid})$ is input with a fresh tid to $\mathsf{cm\_s_s}(r, i)$ and to $\mathsf{th}(s, r, i)$ (inputs with non-fresh tid are ignored in both systems). Then, $\mathsf{th}(s, r, i)$ changes to $\mathsf{s_1}$ and outputs $(\mathsf{adv\_busy}, s, l, \mathit{tid})$, which is dispatched as $(\mathsf{send}, r, l, m_{sim}, \mathit{tid})$ to $\mathsf{cm\_s'_s}(r, i)$. Then, the sender sub-machines send m_1 and $m_{1,sim}$, respectively. These messages only differ with respect to the committed message. The perfect indistinguishability of these two commitments follows from the hiding property of the underlying commitment scheme. If the adversary does not send a correct m_2, then $\mathsf{cm\_s_s}(r,i)$ outputs (failed, tid) at time $i+6$. $\mathsf{cm\_s'_s}(r,i)$ on the other hand sends $\mathsf{adv\_suppress}$ to $\mathsf{th}(s,r,i)$, which changes from state $\mathsf{s_1}$ to $\mathsf{s_3}$. Thus, since TH inputs stop at time $i+6$, this machine outputs (failed, tid) at $s^*(H)$ as well.

If we now assume that $\mathsf{cm\_s_s}(r,i)$ and $\mathsf{cm\_s'_s}(r,i)$ receive a correct m_2, $\mathsf{cm\_s_s}(r,i)$ sends m_3 while $\mathsf{cm\_s'_s}(r,i)$ inputs $\mathsf{adv\_receive}$ to $\mathsf{th}(s,r,i)$, which changes from state $\mathsf{s_1}$ to $\mathsf{s_2}$ while outputting $(\mathsf{adv\_msg}, m)$. Then, $\mathsf{cm\_s'_s}(r,i)$ sends $m_{3,sim}$. The adversary cannot distinguish (m_1, m_3) from $(m_{1,sim}, m_{3,sim})$ since r as well as r'_S are uniformly distributed in $\mathbb{Z}_q$ (recall that r_{sim} is never revealed to the adversary):

$$\begin{aligned} r'_S &= (\mathcal{H}(m_{sim}) - \mathcal{H}(m))\, x^{-1} + r_{sim} \mod q \\ &= \mathsf{const} + r_{sim} \mod q. \end{aligned}$$

After sending m_3 and $m_{3,sim}$ the behavior of $\mathsf{cm\_s_s}(r,i)$ and $\mathsf{cm\_s'_s}(r,i)$ with respect to the network is identical (including $\mathsf{cm\_t}(s,r,i)$ and $\mathsf{cm\_t'}(s,r,i)$). (Note that an output $(\mathsf{adv\_send}, m)$ by $\mathsf{cm\_t'}(s,r,i)$ will be ignored by $\mathsf{th}(s,r,i)$ after m_3 has been sent.) This includes recovery in case m_4 was not received correctly (in particular, at the end, they will be in the same state). Let us now consider the outputs at $s^*(H)$ at time $i+6$. After receiving m_2, the real protocol outputs (sent, tid) at time $i+6$ at $s^*(H)$ in any case. (Recall that the connection to the correct $\mathsf{cm\_t}(s,r,i)$ is reliable.) The same holds for $\mathsf{th}(s,r,i)$, since it is in state $\mathsf{s_2}$ and will receive an input stop. This implies that $\mathsf{th}(s,r,i)$ changes to state $\mathsf{received}$ exactly if and when $\mathsf{cm\_s_s}(r,i)$ changes to $\mathsf{received}$ or $\mathsf{received'}$.

This covers all inputs that are accepted from A and H during the "send"-protocol. Let us now consider all other accepted inputs, namely network inputs to the verifier as well as the input of show to the correct sender.

The real verifier only produces an output at $s^*(H)$ iff a correct receipt $(m'_1, \ldots, m'_4)$ or m'_6 was received that passes all the verifications. The same holds for the output $((s,r,i), \mathsf{adv\_show})$ by $\mathsf{cm\_v'}(s,r,i)$ by construction. Let us now consider the outputs of sub-machine $\mathsf{th}(s,r,i)$ at $out(v)$: If $\mathsf{cm\_v'}(s,r,i)$ outputs $(s,r,i,\mathsf{adv\_show})$ while $\mathsf{th}(s,r,i)$ does not output $(\mathsf{received}, s, r, l, m, tid)$, then $\mathsf{th}(s,r,i)$ either was not in state $\mathsf{s_2}$, $\mathsf{received}$, or $\mathsf{showing}$[25] or else these states store different parameters (see Fig. 9.8).

In the first case, the simulated sender $\mathsf{cm\_s'_s}(r,i)$ did not output $\mathsf{adv\_receive}$ and thus did not send m_3. Since the verifier obtained a correct m'_1 and m'_3 for this tid (directly or in m_6), the security of a primitive was broken:

1. If the adversary was able to present an $m'_1 \neq m_1$ for the same (s,r,i), then the signature scheme was broken since $\mathsf{cm\_s'_s}(r,i)$ signs m_1 only once for each (s,r,i) and no other sub-machine of $\mathsf{M'_s}$ signs m_1 for this (s,r,i).[26]

2. Otherwise, the commitment is fixed by m_1. If the adversary was able to produce a pair (r', c') with $\mathsf{Com}(c', r') = \mathsf{Com}(c, r)$, then, it was able to

[25]In this state the input $\mathsf{adv\_show}$ is ignored but an output $(\mathsf{received}, \ldots)$ at $s^*(H)$ is produced anyway.

[26]Recall that we defined that signed messages carry a type identifier, so that no message not called "m_1" can equal a message m_1. Moreover, we required that the implementation of tuples guarantees unambiguous decomposition, and m_1 contains s, r and i in fixed components.

break the binding property if $c' \neq c$, or else the hiding property of the commitment scheme.

In the second case, $\mathsf{th}(s,r,i)$ stores the parameters input to $\mathsf{cm\_s}'_\mathsf{s}(r,i)$ and fixed in m_1 while $\mathsf{cm\_v}(s,r,i)$ received an $m'_1 \neq m_1$ or $m'_3 \neq m_3$. Thus, one of the primitives was broken, too. (If $m'_1 = m_1$ then the adversary knows (c,r) and is required to find a $c' \neq c$ to break the binding property in Item 2, or to find a collision of the hash function.)

Let us now consider an input of (show, tid) in Round j to $\mathsf{cm\_s_s}(r,i)$ and $\mathsf{th}(s,r,i)$, respectively. Machine $\mathsf{cm\_s_s}(r,i)$ considers this input iff it is in in state $\mathsf{received}$ or $\mathsf{received}'$, while $\mathsf{th}(s,r,i)$ considers it if it is in state $\mathsf{received}$ or $\mathsf{showing}$. Above we showed that this happens under the same conditions (in Round $i+6$), and that $\mathsf{cm\_s}'_\mathsf{s}(r,i)$ is then in the same state as $\mathsf{cm\_s_s}(r,i)$. By construction, the message $\mathsf{cm\_s_s}(r,i)$ then sends to $\mathsf{cm\_v}(s,r,i)$ is accepted, and thus $\mathsf{cm\_v}(s,r,i)$ outputs $(\mathsf{received}, s, r, l, m, tid)$ in Round $j+1$. In the simulation, $\mathsf{th}(s,r,i)$ changes to state $\mathsf{showing}$ and outputs $(\mathsf{adv\_busy}, v, l, tid)$, which leads to an input (show, tid) to $\mathsf{cm\_s}'_\mathsf{s}(r,i)$. Hence, still in Round j, machine $\mathsf{cm\_s}'_\mathsf{s}(r,i)$ sends the corresponding message to $\mathsf{cm\_v}'(s,r,i)$ as $\mathsf{cm\_s_s}(r,i)$, hence the network messages are indistinguishable. In Round $j+1$, $\mathsf{th}(s,r,i)$, being in state $\mathsf{showing}$, outputs $(\mathsf{received}, s, r, l, m, tid)$. (Additionally in Round $[j.4]$, $\mathsf{cm\_v}'(s,r,i)$ accepts and outputs $\mathsf{adv\_show}$; this is ignored by $\mathsf{th}(s,r,i)$ in Round $j+1$ in state $\mathsf{showing}$. This corresponds to the fact that the trusted host specifies that showing receipts by a correct sender should *always* work.)

Correct Recipient; Incorrect Sender: Let $s \in A_S$ and $r \in H_R$. We therefore compare the behavior of $\mathsf{th}(s,r,i)$, $\mathsf{cm\_r}'_r(s,i)$, $\mathsf{cm\_v}'(s,r,i)$, and $\mathsf{cm\_t}'(s,r,i)$ with $\mathsf{cm\_r}_r(s,i)$, $\mathsf{cm\_v}(s,r,i)$, and $\mathsf{cm\_t}(s,r,i)$. Before Round i, all sub-machines do nothing. If $(\mathsf{receive}, s, l, tid)$ is not input in Round i at $in(r)$, all machines remain in their starting state forever without making any outputs.

Let us now consider the case that $(\mathsf{receive}, s, l, tid)$ with a fresh tid is input to $\mathsf{cm\_r_r}(s,i)$ and $\mathsf{th}(s,r,i)$, respectively. Then $\mathsf{th}(s,r,i)$ changes to state $\mathsf{r_1}$ and outputs $(\mathsf{adv\_busy}, r, l, tid)$, which leads to an input $(\mathsf{receive}, s, l, tid)$ to $\mathsf{cm\_r}'_\mathsf{r}(s,i)$. Then, both sub-machines wait for m_1 (we omit the distinction between m_1 and $m_{1,sim}$ since they are indistinguishable). If the adversary does not send m_1 then $\mathsf{cm\_r_r}(s,i)$ outputs (failed, tid) at time $i+6$. $\mathsf{cm\_r}'_\mathsf{r}(s,i)$ on the other hand outputs $\mathsf{adv\_suppress}$ to $\mathsf{th}(s,r,i)$, which changes from state $\mathsf{r_1}$ to $\mathsf{r_2}$. Thus, since TH inputs stop at time $i+6$, this sub-machine outputs (failed, tid) at $s^*(H)$ as well.

If $\mathsf{cm\_r_r}(s,i)$ and $\mathsf{cm\_r}'_\mathsf{r}(s,i)$ receive a correct m_1, both sub-machines send m_2.

If a correct m_3 is not received, both machines wait for m_6. If m_6 is not received either, the real machine $\mathsf{cm\_r_r}(s,i)$ outputs (failed, tid) at time $i+6$, and $\mathsf{th}(s,r,i)$ outputs (failed, tid) on input stop since $\mathsf{cm\_r}'_\mathsf{r}(s,i)$ did not output $(\mathsf{adv\_send}, m)$.

If a correct m_3 is received, both machines send m_4 and $\mathsf{cm\_r}'_\mathsf{r}(s,i)$ inputs $(\mathsf{adv\_send}, m)$ to $\mathsf{th}(s,r,i)$, which changes from state $\mathsf{r_1}$ to $\mathsf{received}$. In this case, machine $\mathsf{cm\_r_r}(s,i)$ as well as the trusted host output $(\mathsf{received}, m, tid)$ at time $i+6$.

After this execution of the “send“-protocol $\mathsf{cm\_r_r}(s,i)$ as well as $\mathsf{th}(s,r,i)$ ignore all inputs at $s^*(H)$ while $\mathsf{cm\_r_r}(s,i)$ and $\mathsf{cm\_r}'_\mathsf{r}(s,i)$ ignore all network inputs from A as well.

Let us now again consider all other accepted inputs. The real verifier $\mathsf{cm\_v}(s,r,i)$ only produces an output (at $s^*(H)$) iff a correct receipt $(m_1',\ldots,m_4')$ or m_6' was received that passes all the verifications. The same holds for the output $((s,r,i),\mathsf{adv\_show})$ by $\mathsf{cm\_v}'(s,r,i)$ by construction. As argued before, $\mathsf{th}(s,r,i)$ does not produce the same output as $\mathsf{cm\_v}(s,r,i)$ only if it is not in state $\mathsf{received}$ or if this state stores different parameters (see Fig. 9.9).[27]

If the recipient sent m_4 at time $[i+3.1]$ in the real protocol, the adversary is first able to cause an output $(\mathsf{received},\ldots)$ using this m_4 at time $[i+4.1]$. In the simulation, the recipient inputs $(\mathsf{adv\_send},m)$ at time $[i+2.4]$ and sends m_4 at time $[i+3.2]$. The adversary obtains m_4 at time $[i+3.2]$ and may first send it to the simulated verifier, which outputs $\mathsf{adv\_show}$ to $\mathsf{th}(s,r,i)$ at time $[i+3.4]$. $\mathsf{th}(s,r,i)$ then outputs $(\mathsf{received},\ldots)$ at time $[i+4.1]$ as well. If the third party sent m_6 at time $[i+5.2]$, the same timing argument holds since the third party input $(\mathsf{adv\_send},m)$ before sending m_6, too. (Since $(\mathsf{adv\_send},m)$ is ignored by $\mathsf{th}(s,r,i)$ after round $i+5$, since $\mathsf{cm\_t}(s,r,i)$ and $\mathsf{cm\_t}'(s,r,i)$ do not accept m_5 after round $i+5$.)

If machine $\mathsf{th}(s,r,i)$ is in state $\mathsf{r_1}$, $\mathsf{r_1}$, or failed, then it either obtained an input $\mathsf{adv\_suppress}$ and thus machine $\mathsf{cm\_r}_\mathsf{r}'(s,i)$ never sent m_2, or it did not receive an input $(\mathsf{adv\_send},m)$ in time and thus neither m_4 was sent by $\mathsf{cm\_r}_\mathsf{r}'(s,i)$ nor m_6 was sent by $\mathsf{cm\_t}'(s,r,i)$. If the adversary presents (m_1',m_2',m_3',m_4') during verification, it was either able to forge the signature on a message $m_2'\neq m_2$. (Again this must really be a forgery, because a message with this type identifier and these parameters s, r, and i is not signed anywhere else.) Else, if $m_2'=m_2$, it was able to break the one-way property of $\mathcal{F}$ by computing a pre-image to $\mathcal{F}(r_R)$ since, except in m_4, no information about the randomly chosen r_R is revealed. If the adversary was able to present any valid message m_6' containing m_2 fixing the slot (s,r,i), it was able to forge a signature of the third party, or to make $\mathsf{cm\_t}'(s,r,i)$ sign it. For this, however, it would again need to forge m_2.

If machine $\mathsf{th}(s,r,i)$ is in state $\mathsf{received}$ but this state contains different parameters than output by $\mathsf{cm\_v}(s,r,i)$, the messages (m_1',m_2',m_3',m_4') or m_6' with $m_2'\neq m_2$ or $m_3'\neq m_3$ were shown to $\mathsf{cm\_v}(s,r,i)$ (m_2,m_4 are the messages sent by the correct recipient). In the first case, the signature on m_2' has been forged. In the second case, the adversary presented $m_3'=(tid,r',m')$ for $m_1=\mathrm{sign}_\mathsf{s}(d_S,\mathsf{Com}(\mathcal{H}(m),r_S))$ as signed in m_2. Thus, it was either able to compute a (r',c') with $c'\neq c$ and $\mathsf{Com}(c',r')=\mathsf{Com}(c,r)$ (i.e., it was able to break the binding property of the commitment scheme), or else compute a $c=c'=\mathcal{H}(m')$ for an $m'\neq m$ (i.e., break the collision-resistance of the hasfunction).

Correct Sender and Recipient: Let $s\in H_S$ and $r\in H_R$. We therefore compare the behavior of $\mathsf{th}(s,r,i)$, $\mathsf{cm\_s'}_s(r,i)$, $\mathsf{cm\_r'}_r(s,i)$ $\mathsf{cm\_v}'(s,r,i)$, and $\mathsf{cm\_t}'(s,r,i)$ with $\mathsf{cm\_s}_s(r,i)$, $\mathsf{cm\_r}_r(s,i)$, $\mathsf{cm\_v}(s,r,i)$, and $\mathsf{cm\_t}(s,r,i)$. Before Round i, all sub-machines do nothing. If neither send nor $\mathsf{receive}$ is input in Round i, all machines remain in their starting states forever without making any outputs.

If $(\mathsf{send},r,l,m,tid)$ is input to $\mathsf{cm\_s}_s(r,i)$ and $\mathsf{th}(s,r,i)$, but $(\mathsf{received},s,l,$ $tid)$ is not input to the corresponding recipient port $in(r)$, machine $\mathsf{th}(s,r,i)$

[27]Here, we need the fact that two adversary outputs of Sim can be processed in parallel. The adversary may cause an output $(\mathsf{adv\_send},m)$ at $\mathsf{cm\_t}'(s,r,i)$ by recovering a completed run in parallel to causing an output $\mathsf{adv\_show}$ by showing the receipt obtained in the completed run.

changes to state sr_1 and outputs $(\mathsf{adv\_busy}, s, l, tid)$, which leads to an input $(\mathsf{send}, r, l, m_{sim}, tid)$ to $\mathsf{cm\_s'}_s(r, i)$. The sender machines then send m_1 and $m_{1,sim}$, respectively. Since the recipient is correct, it will not send m_2 and thus, if the signature scheme is not broken, both recipient machines will not receive m_2 (m_2 again fixes the parameters and a correct recipient never signs it if its input from $\mathsf{th}(s, r, i)$ contained different parameters). In this case, $\mathsf{cm\_s}_s(r, i)$ will outputs (failed, tid) at time $i + 6$. In the simulation, machine $\mathsf{th}(s, r, i)$ is in state sr_1, ignores the input $\mathsf{adv\_suppress}$ from $\mathsf{cm\_s}'_\mathsf{s}(r, i)$, and outputs $(\mathsf{failed\_for\_s}, tid)$ on input of stop at time $i + 6$, which leads to an output (failed, tid), too.

If $(\mathsf{receive}, r, l, tid)$ is input to $\mathsf{cm\_r}_\mathsf{r}(s, i)$ and $\mathsf{th}(s, r, i)$, but $(\mathsf{send}, r, l, m, tid)$ for any $m \in Msg$ is not input to the corresponding sender machines, $\mathsf{th}(s, r, i)$ changes to state sr_2 and outputs $(\mathsf{adv\_busy}, r, l, tid)$, which leads to an input $(\mathsf{receive}, s, l, tid)$ at $\mathsf{cm\_r}'_\mathsf{r}(s, i)$. Then, both recipient machines wait for m_1. Since the sender machines are correct, they will not sign m_1 and thus, if the signature scheme is not broken, both will not receive m_1. In this case, $\mathsf{cm\_r}_\mathsf{r}(s, i)$ will output (failed, tid) at time $i{+}6$ while machine $\mathsf{th}(s, r, i)$ is in state sr_2, ignores the input $\mathsf{adv\_suppress}$ from $\mathsf{cm\_r}'_\mathsf{r}(s, i)$, and outputs $(\mathsf{failed\_for\_r}, tid)$ on inputstop at time $i + 6$, which leads to an output (failed, tid) at time $i + 6$ as well.

Let us now consider the case that a different label or tid was input. In both cases, machine $\mathsf{cm\_s}_\mathsf{s}(r, i)$ sends m_1 while the recipient does not receive it as expected (unless the adversary was able to forge an $m'_1 \neq m_1$). Thus, without sending any more messages, the sender and the recipient will output (failed, tid) at time $i + 6$. In the simulation only $m_{1,sim}$ is sent as well and the trusted host $\mathsf{th}(s, r, i)$ changes to state sr_3 and outputs (failed, tid) to both at the end. (Again, $\mathsf{th}(s, r, i)$ ignores the inputs $\mathsf{adv\_suppress}$ from $\mathsf{cm\_s}'_\mathsf{s}(r, i)$ and $\mathsf{cm\_r}'_\mathsf{r}(s, i)$, respectively.)

Let us now consider the case that $(\mathsf{send}, r, l, m, tid)$ and $(\mathsf{receive}, r, l, tid)$ was input to the corresponding ports with matching parameters. In this case, the sender sub-machines $\mathsf{cm\_s}_\mathsf{s}(r, i)$ and $\mathsf{cm\_s}'_\mathsf{s}(r, i)$ send m_1 and $m_{1,sim}$, respectively.

If the adversary does not send a correct m'_1, then $\mathsf{cm\_s}_\mathsf{s}(r, i)$ and $\mathsf{cm\_r}_\mathsf{r}(s, i)$ output (failed, tid) at time $i + 6$ without sending additional messages. In the simulation, $\mathsf{cm\_r}'_\mathsf{r}(s, i)$ inputs $\mathsf{adv\_suppress}$ to $\mathsf{th}(s, r, i)$ without sending any message, i.e., $\mathsf{th}(s, r, i)$ changes from state sr_4 to sr_3 and outputs (failed, tid) to both participants as well.

If $m'_1 \neq m_1$, then the adversary was able to forge a signature of the sender for the given (s, r, i).

If the adversary forwards the correct m_1 then the recipient machines $\mathsf{cm\_r}_\mathsf{r}(s, i)$ and $\mathsf{cm\_r}'_\mathsf{r}(s, i)$ send m_2.

If no correct m'_2 is sent by the adversary, machine $\mathsf{cm\_s}_\mathsf{s}(r, i)$ outputs (failed, tid) while $\mathsf{cm\_s}'_\mathsf{s}(r, i)$ inputs $\mathsf{adv\_suppress}$ to $\mathsf{th}(s, r, i)$ and $\mathsf{th}(s, r, i)$ outputs (failed, tid) to both participants at time $i + 6$. Since the sender did not receive m_2, $\mathsf{cm\_r}_\mathsf{r}(s, i)$ does not receive a correct m'_3 (unless one of the primitives was broken). Thus, it outputs (failed, tid) as well.

If the adversary was able to compute a $m'_2 \neq m_2$, then it was able to forge a signature of the recipient for the given (s, r, i).

If the sender machine $\mathsf{cm\_s}_\mathsf{s}(r, i)$ receives m_2, it sends m_3, while machine $\mathsf{cm\_s}'_\mathsf{s}(r, i)$ inputs $\mathsf{adv\_receive}$ to $\mathsf{th}(s, r, i)$. Machine $\mathsf{th}(s, r, i)$ changes from state sr_4 to sr_5 and outputs $(\mathsf{adv\_msg}, m)$ while $\mathsf{cm\_s}'_\mathsf{s}(r, i)$ sends $m_{3,sim}$ as well. In this case, indistinguishability of (m_1, m_3) and $(m_{1,sim}, m_{3,sim})$ follows from indistinguishability for incorrect recipient.

At this point, the final output in both cases will be (sent, tid) and $(\mathsf{received}, tid)$ while the behavior on the network will be identical. (The output $(\mathsf{adv\_send}, m)$ from $\mathsf{cm\_r}'_{\mathsf{r}}(s, i)$ to $\mathsf{th}(s, r, i)$ is ignored.)

The argument for network inputs to the verifier as well as an input (show, tid) is identical to the case with an incorrect recipient, except that we now use the fact that machine $\mathsf{th}(s, r, i)$ changes to sr_5 (instead of s_2) on input $(\mathsf{adv\_send}, m)$. ■

Chapter 10

Conclusion and Outlook

In this part, we have defined the new relation "as secure as" comparing the overall security of two reactive systems implementing the same service. Furthermore, we have introduced the new concept of real-world trusted hosts.

Compared to earlier approaches to trusted-host-based specification, real-world trusted hosts do not define the ideal service but rather the real-world service including vulnerabilities of a service that need not be avoided.

Together, these two concepts enable efficient and practical protocols that are provably secure.

As a consequence, each designer of a new practical system need not specify the service from scratch, but may rather just show that the new design is as secure as the well-established trusted host for this service.

Finally, we demonstrated our approach by specifying and evaluating reactive systems for certified mail and secure message transmission. Unfortunately, we were unable to present a certified mail scheme that is secure according to our definition as well as optimistic.

While this new notion of security for reactive systems is a first step towards a general definition, there are still many open problems to solve. At first, the new formalism and its implications may be studied in more detail.

One area of extensions are further refinements of our model: The extension to asynchronous systems, the inclusion of more powerful adversary models, such as mobile or dynamic adversaries instead of static ones, and the possibility of limiting the scope of the adversary, are desirable extensions that should be pursued. A starting point for such extensions would be existing solutions for these problems in other models.

Another much more complex issue is to reduce the over-specification of our trusted hosts while still retaining the guarantee that the resulting systems remain secure in practice. Such practical specification should leave more freedom to the implementors. However, an ambiguous specification may easily introduce new security leaks such as covert channels. Thus, identifying an acceptable tradeoff between provable security and coverage of most secure real-world implementations seems to be one of the harder problems to solve.

Part III

Transfer-based Optimistic Fair Exchange

Chapter 11

Introduction and Overview

This part describes a design and protocols for *generic* and *optimistic* fair exchange of any two business items.

Part I discussed the efficiency of two instances of fair exchange, namely *certified mail* and *fair purchase*, even though there are many more types of two-party fair exchanges (see Section 2.1). On an abstract level, any two items, such as signatures, data, or payments, can be exchanged for each other. For, e.g., these items, one would require at most nine different fair exchange protocols and solutions to these nine fair exchange problems have already been developed. So why develop *generic* fair exchange?

The reason is that this abstract view does not hold in practice: In practice, differentiating between payments, signatures, and data is too coarse. A fair exchange protocol for payment for receipt may work with one payment scheme but not with another. So instead of having nine different protocols for exchanging signatures, payments, and data, each new implementation of a business item may require new fair exchange protocols for each item which has already been installed. Therefore, for a given number of n different kinds of business items, this leads to about $n^2/2$ different fair exchange protocols if one only wants to exchange one single item for another. Adding a new type of item means adding $n+1$ additional fair exchange protocols. Furthermore, exchanging multiple business items (e.g., a payment in exchange for the delivery of a program and a signed delivery note) requires specific fair exchange protocols for any fixed combination of items to be exchanged.

The solution to this problem is to provide exchanges which are independent of the items. This is done by defining so-called *exchange-enabling properties* of transfers[1] of business items, which can be exploited by fair exchange protocols to guarantee fairness.

A *fair exchange* is implemented as two virtually parallel transfers of two business items satisfying the fixed expectations of both exchanging parties. Compared to just two transfers, the focus lies on the "atomic parallelism" which is not provided by two subsequent transfers where one party (the first one to receive a complete transfer) usually has an advantage. Therefore, in order to guarantee this "atomic parallelism", generic fair exchange interleaves

[1]A *transfer* sends an item from a sender to a recipient. Examples of transfers are payments (transfers value), signature protocols (transfers a signature on a document), or messages (transfers data).

both transfers while the exchange-enabling properties guarantee the atomicity of the interleaving.

Unfortunately, fairness requires a third party in case of faults[2]. Again, we try to avoid invoking the third party in the normal case. This is done by defining so-called *optimistic* exchange-enabling properties of transfers that enable optimistic fair exchanges (cf. [BüPf 89]) where the third party is only needed in case of faults, i.e., if both players are correct, the third party is not actively involved.

As we will show, the fairness of our generic exchange protocols is based on four exchange-enabling properties of transfers:

Sender Verifiability: A sender can convince the third party that a correct recipient was able to obtain an item.

A simple example of providing sender verifiability of, e.g., signatures is to re-send the signature to the third party who verifies and forwards it.

Recipient Verifiability: A recipient can convince the third party that it is unable to obtain an item.

Recipient verifiability can, e.g., be provided by asking a bank whether a particular coin has been deposited or not. Furthermore, the coin needs to be blacklisted in order to prevent later deposit.

Generatability: The third party can be authorized by the sender to complete or redo a transfer without the cooperation of the sender.

A simple example of providing generatability, e.g., for signatures is to authorize the third party to sign on one's behalf.

Revocability: The third party can be authorized by the recipient to "undo" a transfer without cooperation of the recipient.

A simple example of providing revocability for credit-card payments is to authorize the third party to revoke a given credit-card payment.

Based on these properties, we are able to exchange many different pairs of items with only few generic fair exchange protocols. Exchanging multiple business items is also simplified: The transfer-based exchanges no longer differentiate between single items and multiple items as long as the transfer of multiple items provides an exchange-enabling property. Furthermore, adding new implementations of business items is considerably easier: Now, the implementor is only required to provide an exchange-enabling property in order to enable fair exchange of the new items.

In principle, our protocols for fair exchange all follow the same pattern: Both parties participating in the exchange sign an agreement what items will be exchanged and what item will be transferred first. Then, each item is transferred using the underlying transfer protocol. After successfully receiving the transfer from the business partner, the fair exchange protocol ends. Else, if the expected item is not received, the participants may ask the third party to restore fairness, e.g., as follows:

[2]A generic protocol cannot be better than instances of it. For contract signing, fairness cannot be guaranteed without a third party in a limited time (see Theorem 3.2 in Section 3.3.2).

- If one of the items is generatable and was not sent, this item is replaced by the third party.
- If one of the items is revocable and the other item was not sent, this item is revoked.

Thus, the third party is able to restore fairness if the items support the required combination of exchange-enabling properties. A protocol for, e.g., contract signing, following this pattern works as follows: Both prospective signatories digitally sign that they want to sign a certain contract. If these letters of understanding have been exchanged, both send their signatures under the contract. However, if one of the signatories receives no signature from the peer, it sends its own signature as well as the letters of understanding to the third party and asks it to sign an affidavit on behalf of the incorrect party not sending its signature.

11.1 Overview

In Chapter 12, we define fair exchanges and transfers with exchange-enabling properties.

Then, in Chapter 13, we describe two protocols for transfer-based fair exchange of two items. Together with the given simulations between exchange-enabling properties, these two protocols are sufficient to exchange any two exchange-enabled items if at least one of them is generatable or revocable.

Finally, in Chapter 14, we describe the *SEMPER* Fair Exchange Framework embedding these definitions and protocols into an object-oriented framework for fair exchange.

11.2 A Short Overview of *SEMPER*

The problem of transfer-based fair exchange as described in this part was identified while developing the *SEMPER* Framework for Secure Electronic Commerce.

The goal of the *SEMPER* project (see `http://www.semper.org`) was to develop an open framework for secure electronic commerce on the Internet. The project was funded from 1995 until 1998 by the ACTS programme of the European Commission.

As a basis for understanding the fair exchange framework described in Chapter 14, we sketch the *SEMPER* model for electronic commerce as well as the framework based on it. For a more detailed description, we have to refer to [ABPP 00, ScWW 98, ScWW1 99].

11.2.1 The *SEMPER* Model of Secure Electronic Commerce

The framework is based on a model of secure two-party electronic commerce. The main idea of this model is to describe business interactions in terms of sequences of transfers and fair exchanges of business items. After each such action, each side decides which action to enable next based on the success of the previous actions (see Figure 11.1).

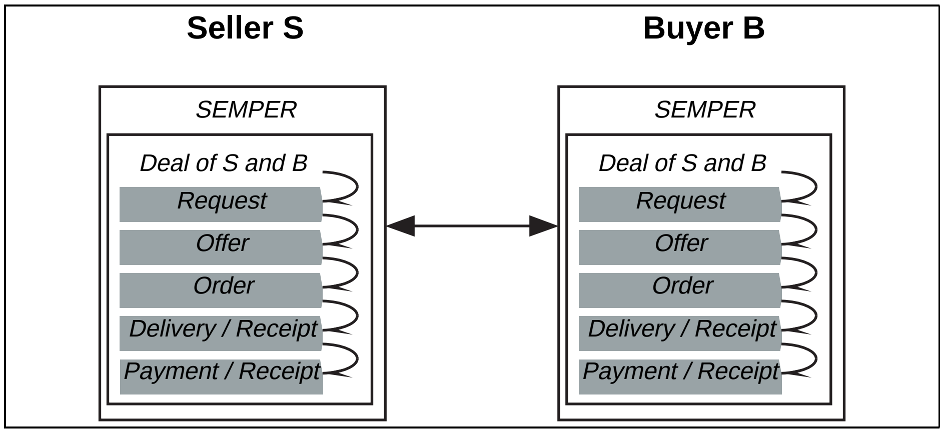


Figure 11.1: Example of a *SEMPER* Deal. (Note that the deal might enable other sequences as well, e.g., after "Contract" the transfer "Payment without Receipt" might also be enabled.)

Interactions: Transfers and Fair Exchanges

The interactive actions between two players are transfers and fair exchanges.

In a transfer, one party sends a package of business items to the business partner. The sending party can define certain security requirements, such as confidentiality, anonymity, or non-repudiation of origin.

In an exchange, each party describes the business items it offers and the business items it expects in exchange. Again, the parties can define security requirements. At the end, if all expectations were met, the business items are exchanged, i.e., each correct participant obtains all expected business items. Else, no exchange takes place and each participants does not obtain additional information on the business items offered by other correct participants.

Sequence of Interactions

In *SEMPER*, a *deal* contains the sequences of enabled interactions. Any deal enables certain sequences of transfers and fair exchanges based on user-interaction, local computations, and local decisions in between two interactions (see Figure 11.1).

In the course of an ongoing deal, after each transfer or exchange, the parties are either

- *satisfied*, and thus willing to proceed with a certain number of other transfers or exchanges, or
- *dissatisfied*, in which case an exception or dispute is raised which might end up at a real court if all else fails.

Based on these facts, it is decided if and how to proceed, i.e., which interaction shall be executed next.

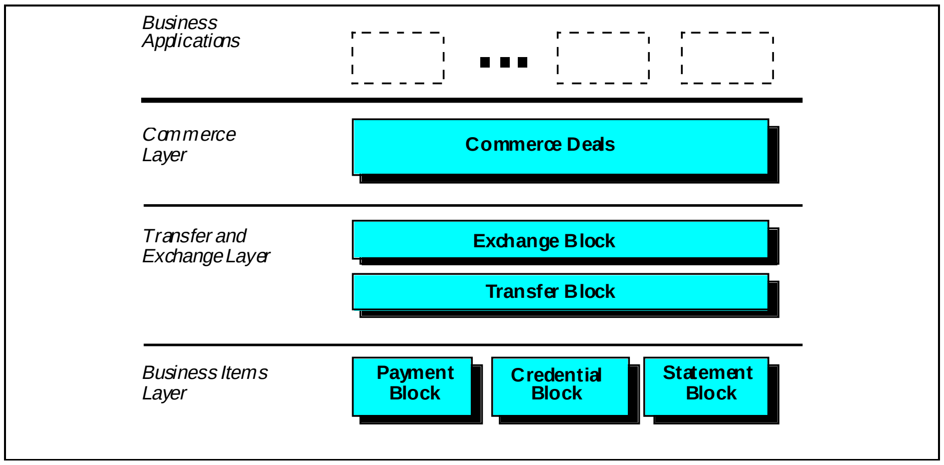


Figure 11.2: Central Blocks of the *SEMPER* Framework for Electronic Commerce.

11.2.2 The *SEMPER* Framework for Secure Electronic Commerce

The *SEMPER* framework (see Figure 11.2) is structured in layers. The lowest of the following layers deals with existing business items whereas the highest layer intuitively deals with commerce issues while abstracting from all details:

- The *Commerce Layer* offers high-level services for executing deals like "mail-order retailing", "on-line purchase of information", or "registration with service provider". It is configurable by downloading new services or extending existing ones.

- The *Transfer and Exchange Layer* provides a framework for transfer and exchange of business items.

  The *Exchange Block* provides services implementing our generic fair exchange protocols.

  The *Transfer Block* defines services to transfer multiple business items in one transaction while guaranteeing common security attributes and exchange-enabling properties.

- The *Business Items Layer* provides business items, such as payments, signatures, or data. Each business item may implement the services defined by the transfer block.

In addition to these layers, any block may use the so-called *Supporting Services*, which are the usual cryptographic services, communication, archiving of data (keys, non-repudiation tokens, audit trail), transaction support, preferences management, access control, and the trusted user interface. Furthermore, it provides secure communication services implementing security services such as anonymity which must be guaranteed for subsequent interactions in one deal.

11.3 Notations and Assumptions

The external behavior of our protocols is depicted as UML sequence diagrams [FoSc 97], i.e., we denote the involved parties by vertical lines sending in- and outputs as messages to each other. Participants in lined boxes are mandatory, whereas participants in dashed ones are optional (e.g., a third party which does not participate if the protocol is optimistic). Grayed participants denote external applications using the service. Dashed arrows denote protocol executions.

As parameters of these in- and outputs, we use the identifiers introduced in the corresponding definition. Commonly used identifiers as well as some assumptions are listed in Figure 11.3.

Like in Part I, we denote a machine Z composed from machines X and Y as $\mathsf{Z} = \langle \mathsf{X}, \mathsf{Y} \rangle$. The sub-machines are defined to be local to each other, i.e., communicate using in- and outputs to each other instead of messages. We use the identifier of the first machine as the identifier of the composite machine, i.e., protocols and signatures of machine Y inside $\langle \mathsf{X}, \mathsf{Y} \rangle$ will be verified under the identifier X. We assume that messages are automatically dispatched to the appropriate sub-machine and that the identifier of X is sufficient for sending messages to the sub-machine Y as well.

If a machine S of a composite machine $\langle \mathsf{S}, \mathsf{S}_{\mathsf{sub}} \rangle$ uses another local machine $\mathsf{S}_{\mathsf{sub}}$ for executing sub-protocols, we distinguish external in- and outputs of the protocol from in- and outputs to the machines running sub-protocols by putting them in quotes, i.e., "$[\mathsf{S}^{in}_{tid} | \mathsf{input}(arg)]$" denotes an external input to the machine S, whereas $[\mathsf{S}_{\mathsf{sub}}{}^{in}_{tid} | \mathsf{command}(args)]$ denotes an output from S input to its sub-machine $\mathsf{S}_{\mathsf{sub}}$. The sub-protocols are separated from the actual protocol using dotted horizontal lines.

For simplicity, we use a synchronous model, i.e., all sub-protocols run for a fixed time and the super-protocol only continues after the sub-protocol has been completed. This holds even if a player only makes an input without receiving an output, i.e., the super-protocol waits until the sub-protocol is supposed to be completed. Furthermore, we assume that messages between correct players are not corrupted and delivered within one round.

In our definitions, each in- and output of a particular protocol execution is labeled with a so-called *Transaction Identifier* "tid". For each protocol run, this identifier is assumed to be fresh and unique for the machines executing the protocol, i.e., the tuples (R, tid) or (S, tid) unambiguously identify a given protocol run as executed by machines R and S. A correct machine only executes a protocol if the tid is fresh and locally unique, i.e., subsequent executions of the same or different protocols are required to use different tids. For subsequent protocol executions of one scheme, we sometimes use unique extensions of one tid. They may be defined as $tid' := (tid, e_1)$, $tid'' := (tid, e_2)$, $\ldots$ with extensions e_1, e_2 that are unique and specific to this scheme in order to guarantee that all resulting tids are fresh if the underlying tid is fresh.

In addition to these transaction identifiers, which are local to one protocol execution and machine, we use so-called *external references* $xref \in XREFs$ (usually strings) for linking[3] two or more protocol executions. We assume that a $xref$ input to a correct protocol machine is fresh and unique for this machine.

[3]This is similar to nesting of sub-transactions [LMWF 94]. Later, in our design, protocols will be encapsulated in so-called transaction objects. Differences are that we, e.g., assume byzantine failures of untrusted machines.

X	An interactive probabilistic machine X named X. (A machine keeps its own state and can, e.g., be implemented as a process.)
σ_X	A particular state σ_X of machine X.
$\mathsf{Z}=\langle\mathsf{X},\mathsf{Y}\rangle$	A composite machine Z, e.g., composed from sub-machines X and Y. The sub-machines are defined to be local to each other. Local sub-machines communicate via in- and outputs instead of messages.
$\mathrm{sign}_{\mathsf{X}}(msg)$	The signature of X under msg. We assume that X as well as msg can efficiently be computed given the signature.
d_X	The description of the item to be sent by machine X.
d'_X	The description of the item expected from the peer machine X.
tid	A transaction identifier linking the in- and outputs belonging to the same protocol execution.
$tid', tid'', \ldots$	Unique extensions of tid usually input to subsequent protocols of the same scheme. It can, e.g., be computed as $tid' := (tid, 1)$, $tid'' := (tid, 2), \ldots$.
$TIDs$	The domain of the tids.
$xref$	The external reference linking two different protocol executions. In order to identify one protocol run unambiguously, we require that $xref$ is fresh and unique among the players executing the subsequent protocols of a given scheme.
$XREFs$	The domain of the $xref$s.
m_i	A message numbered i. This message is usually sent in round i of a synchronous protocol.
t	An absolute time usually measured in terms of the clock of machine T.
t_0	The starting time of the first protocol of a scheme.
Δ	A difference between two times such as the run-time of a protocol or a delay.
ϵ	The empty word denoting “no output”.
$[\mathsf{P}^{in}_{tid}\|\mathsf{cmd}(args)]$	The input of a command cmd with arguments $args$ to a protocol running under tid executed by machine P. For composite machines, inputs are made directly to each sub-machine, i.e., there is no notion of in- and outputs for composite machines.
$[\mathsf{P}^{out}_{tid}\|\mathsf{tag}\colon par]$	The output of a result tagged tag with output parameters par by a protocol running under tid executed by machine P.

Figure 11.3: Identifiers, Notations, and Assumptions as used in our Protocols.

However, in order to link multiple protocol runs, we sometimes allow that one $xref$ is input to multiple protocols that shall be linked. However, even in this case, we assume that this $xref$ is not input to any other protocol execution. Examples of nested protocols where this link is required for security are nesting transfers into an exchange, or nesting exchanges into sequences (see Section 11.2.1). For simplicity, we assume that the domain $XREFs$ is sufficiently large to allow for the $xref$s as input by our super-protocols to used sub-protocols without explicitly defining a larger $XREFs$ for each sub-protocol.

Like in Part I, we assume that signed messages are typed and labeled with the protocol parameters [AnNe1 95], e.g., that sending $m_2 = \text{sign}_{\mathsf{O}}(text)$ in protocol "prot" using a third party T executed by machine S running with a tid started at time t_0 to machine R actually sends the signed message $\text{sign}_{\mathsf{O}}(\text{“}\mathsf{prot}\text{“}, \mathsf{S}, \mathsf{R}, \mathsf{T}, tid, t_0, \text{“}\mathsf{m\_2}\text{”}, text)$ in order to prevent attacks from interchanging messages between different protocols and runs (the identifier "m_2" denotes the unique name identifying message m_2; for clarity of our protocols, we may nevertheless mention some of the parameters that are included and signed automatically). Messages without this form or with unexpected parameters are simply ignored. Messages that do not arrive in their designated round are ignored.

Remark 11.1. Note that in Part I, the tid was used for linking different in- and outputs of one protocol as well as different protocols. For nesting two transfer protocols into exchanges, however, we felt that these two functionalities should be clearly separated. ◦

Chapter 12

Foundations of Transfer-based Fair Exchange

In this chapter, we first define transfer-based fair exchange of business items in Section 12.1. Then we define the exchange-enabling properties of transfers in Section 12.2. Finally, in Section 12.3, we illustrate how to provide exchange-enabling properties for some common business items.

12.1 Business Items and Fair Exchanges

Figure 12.1 depicts the roles in transfers and fair exchanges: An exchange-enabling transfer is executed between a sender and a recipient whereas an exchange is initiated by an originator and answered by a responder. One of our goals is that generic exchanges use exchange-enabling transfers.

12.1.1 Business Items

A business item is something which can be owned and has some use. Examples include digital signatures, payments, or data.

In order to phrase requirements for transfers and fair exchanges, we have to define what "gaining" or "losing" (parts of) a business item means. For each existing item, such a notion clearly exists: One knows a message, received a payment, or can show a signature. However, no generic notion of having an item which includes all those cases exist.

Therefore, we model the "ownership" of an item by defining a family of

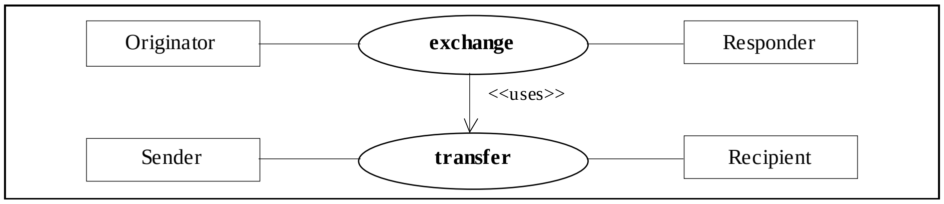


Figure 12.1: Roles for Transfers and Fair Exchanges.

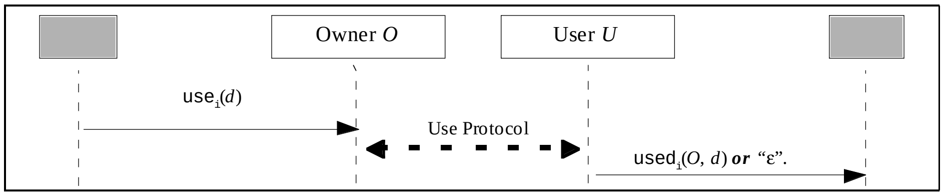


Figure 12.2: Interactions for Using Basic Commerce Items.

protocols named "use" between the alleged owner and a correct user machine U (the interactions are depicted in Figure 12.2). During the execution of this protocol, the user decides whether the alleged owner has the item or not. Note that these "use"-protocols are mainly a tool for defining our requirements on the underlying items. They are not used by the protocols but rather for proving their security, i.e., we assume that a family of "use"-protocols is defined for each item and then express the requirements on exchange-enabling transfers and fair exchanges with respect to these "use"-protocols.

The reason why we define a family of "use"-protocols instead of just one protocol is that we have to tackle the problem of partial information: A message is only secret if no partial information can be guessed, i.e., we have to rule out that an adversary guesses any part. For some schemes, such as some digital signature schemes, no families are needed: There may be just one "use" protocol, namely the verification of the signature.

Definition 12.1 (Business Items)
A *business item* (or short "item") is a tuple $B = (\mathsf{U}, D, \mathit{TIDs})$ containing a correct machine U, a set D of descriptions[1], and a set TIDs of transaction identifiers where the machine U can execute a family $\{$"use$_\mathsf{i}$"$|i \in \mathbb{N}\}$ of protocols:

Using the Item: The "use$_\mathsf{i}$"-protocol is started by the input $[\mathsf{O}^{in}_{tid}|\mathsf{use_i}(d)]$ at a machine O with $d \in D$ and any $tid \in \mathit{TIDs}$ which is fresh for O and U[2]. It may eventually output $[\mathsf{U}^{out}_{tid}|\mathsf{used_i}\colon \mathsf{O}, d]$ with $d \in D$ to the user U.

Owning an Item: A correct machine O *owns an item* with a description d, if an input $[\mathsf{O}^{in}_{tid}|\mathsf{use_i}(d)]$ for any $i \in \mathbb{N}$ with any fresh tid eventually leads to an output $[\mathsf{U}^{out}_{tid}|\mathsf{used_i}\colon \mathsf{O}, d]$.

Illegally Obtaining an Item: An incorrect machine O' *obtained knowledge on an item with description* d during a protocol run if, intuitively speaking, the gained information non-negligibly increases the probability that O' is able to execute any "use$_\mathsf{i}$"-protocol ($i \in \mathbb{N}$) such that the user outputs $[\mathsf{U}^{out}_{tid}|\mathsf{used_i}\colon \mathsf{O}', d]$.

◇

Remark 12.1. Note that by assuming only the owner and the user as being correct, we subsume additional machines which are required to be correct for

[1]In practical implementations, one may use a half-order (e.g., \$5 is more than \$4 but incomparable with EUR3). For our definitions, this would change the equality comparison of two descriptions to the application of this half-order. For simplicity, however, we did not use them in the sequel.

[2]This means that this is the first protocol run on this machine with this particular tid.

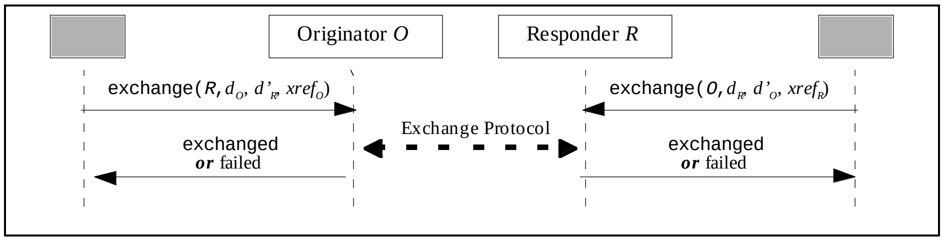


Figure 12.3: Interactions for Exchanging Two Items.

usability of the item as sub-machines of the user. Examples are certification authorities, or the clearing center of a bank. If one would identify all these participating machines, we would need real trust models, i.e., we would need to identify for each player and requirement[3], which machines are assumed to be correct. This, however, would limit the readability of the resulting definitions without substantial gain. ∘

Examples of business items and their descriptions are (see Section 12.3 for detailed examples):

Payments can be described by the amount, the payer and the currency.

Signatures can be described by the signer and the message to be signed.

Data can be described by an input description predicate, a one-way function image[4] fixing the data, or may be evaluated interactively by asking the user.

12.1.2 Fair Exchange

In a fair exchange, each party participating in the exchange inputs one description of the item it offers to send as well as one description of the item it expects to receive in exchange. Furthermore, each inputs an external reference $xref$ for linking the exchange into other protocols such as a commerce sequence (see Section 11.2.1). The interface events of a fair exchange are depicted in Figure 12.3.

Note that we distinguish two roles, even though the service is symmetric. The reason for this distinction is the fact that in most fair exchange protocols, an originator starts the protocol by sending a message to a responder.

The security requirements are that only if both offered items fulfill the expectations of the peers, both parties will end up being owners of the expected items. If not, nothing happens, i.e., no knowledge about the items can be obtained.

Definition 12.2 (Fair Exchange)
An *exchange scheme* is a tuple $(\mathsf{O}, \mathsf{R}, B_O, B_R, XREFs, TIDs)$ where machine O is called originator, R is called responder, and $B_O = (\mathsf{U}_\mathsf{O}, D_O, TIDs_O)$ and $B_R =$

[3]The trusted players may depend on the requirement: For contract signing, e.g., a player is usually required to trust the third party for fairness but not for unforgeability of a contract.

[4]For well-known data, such as software, these images may, e.g., be certified and published by the manufacturer.

$(\mathsf{U}_\mathsf{R}, D_R, \mathit{TIDs}_R)$ are two business items. In addition, the scheme may define a set AM with auxiliary machines without in- or outputs. Machine O can execute the following protocol with R:

Exchange (Protocol "exchange"*):* The players obtain the local inputs $[\mathsf{O}^{in}_{tid}|$ $\mathsf{exchange}(\mathsf{R}, d_O, d'_R, \mathit{xref}_O)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{exchange}(\mathsf{O}, d_R, d'_O, \mathit{xref}_R)]$ with d_R, $d'_R \in D_R$, $d_O, d'_O \in D_O$, $\mathit{xref}_O, \mathit{xref}_R \in \mathit{XREFs}$, and $\mathit{tid} \in \mathit{TIDs}$. In the input to, e.g., O, exchange indicates that the "exchange"-protocol shall be executed, R identifies the user one wants to exchange the item with, d_O is the description of the item to be sent from O to R, d'_R is the description of the item O expects to receive from R, xref_O is the external reference, and tid is the common unique transaction identifier. We assume that xref and tid are fresh and unique for O and R.

At the end of the protocol run, each participant, say O, returns a local output, which can take the following values: An output "$[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$" signals that a successful exchange took place and an item with the expected description has been received. An output "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$" signals that the exchange failed and no item was sent or received.

Two correct players O and R *can exchange* two items $(d_O,\ d_R)$, if the inputs $[\mathsf{O}^{in}_{tid}|\mathsf{exchange}(\mathsf{R}, d_O, d_R, \mathit{xref})]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{exchange}(\mathsf{O}, d_R, d_O, \mathit{xref})]$ lead to the outputs $[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$ and $[\mathsf{R}^{out}_{tid}|\mathsf{exchanged}]$.

An exchange scheme is fair iff it fulfills the following requirements for all O and R:

Requirement 12.2a (Correct Execution): If O and R are correct and the inputs are $[\mathsf{R}^{in}_{tid}|\mathsf{exchange}(\mathsf{O}, d_R, d'_O, \mathit{xref}_R)]$ and $[\mathsf{O}^{in}_{tid}|\mathsf{exchange}(\mathsf{R}, d_O, d'_R, \mathit{xref}_O)]$, the protocol either outputs $[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$ and $[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$ or else $[\mathsf{O}^{out}_{tid}|$ $\mathsf{exchanged}]$ and $[\mathsf{R}^{out}_{tid}|\mathsf{exchanged}]$.

Inputs with $\mathit{xref}_R \neq \mathit{xref}_O$, $d_R \neq d'_R$, or $d_O \neq d'_O$ must lead to the outputs $[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$ and $[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$.

Requirement 12.2b (Transfer): If the protocol outputs $[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$ on input $[\mathsf{O}^{in}_{tid}|\mathsf{exchange}(\mathsf{R}, d_O, d'_R, \mathit{xref}_O)]$ to a correct participant, say O, then O owns the item B_R with description d'_R.[5]

Requirement 12.2c (No Surprises): If a participant, say O, obtains an output $[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$ on input $[\mathsf{O}^{in}_{tid}|\mathsf{exchange}(\mathsf{R}, d_O, d_R, \mathit{xref}_O)]$, even an incorrect peer R has not obtained knowledge about the item B_O described by d_O[6].

Requirement 12.2d (Termination): A correct player will process a correct input within a fixed number of rounds.

$\diamond$

Remark 12.2. This definition does not require availability of an exchange, i.e., an exchange scheme that does nothing and always outputs false is correct. The reason for this remedy is that we are unable to define the pre-conditions for being able to exchange on this abstract level. For transfer-based exchanges, however, this will be fixed.

[5]Note that this does not require that the sender of the item looses it. Some item types (such as signatures) can be sent any number of times.

[6]Note that this intuitive notion is difficult to formalize for arbitrary items (see Section 15).

Remark 12.3. It is clear that a participant is able to prevent successful exchange of the items in any case: It just inputs an expectation that is not fulfilled by the item input by the peer. The only requirement is that in this case, this participant is unable to obtain the item sent by the peer.

Remark 12.4. This definition does not make any assumptions how to achieve fairness, e.g., it does not include a machine acting as a third party. Its goal is rather to identify the fundamental goals of fair exchange. ∘

12.1.3 Transferable Business Items

Business items with exchange-enabling transfers (see Section 12.2) define protocols for transferring an item d from a sender to a recipient. Transfer in this context means that the sender assigns ownership of an item to the intended recipient, in particular if this recipient was no owner of this item before.

Since the protocols and interfaces of transferable items vary considerably, we postpone the definition of the complete interfaces of particular kinds of transferable items to Section 12.2 and concentrate on the property needed to define availability of fair exchange, namely that a scheme defines whether a particular sender is able to transfer a particular item.

Definition 12.3 (Transferable Item)
A *transferable business item* is a tuple $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, \mathit{XREFs}, \mathit{TIDs}, \delta, \Delta)$ containing a sender machine S, a recipient machine R, a third-party machine T, an item $B = (\mathsf{U}, D, \mathit{TIDs}_B)$, a set XREFs of external references, a set TIDs of transaction identifiers, a predicate

$$\delta : \Sigma_S \times \Sigma_R \times D \to \{\mathsf{true}, \mathsf{false}\},$$

where Σ_S is the set of states of machine S, Σ_R is the set of states of R, and a list Δ containing the running-times of the provided protocols.

If $\delta(\sigma_S, \sigma_R, d_S)$ holds, we say that the sender S in state σ_S *can transfer an item* with description d_S to a recipient in state σ_R. ◇

Remark 12.5. The list Δ is used for time-outs of the synchronous super-protocols. Since we did not assume that all participants in a protocol produce an output, a super-protocol sometimes needs these numbers to know whether a sub-protocol should actually be completed.

Remark 12.6. Note that this definition does not require availability of transfers. This requires more detailed knowledge of the items[7]. An implementation, which is never able to transfer its items, easily fulfills the given requirements. Thus, we rather "moved" the availability problem from the definition of exchanges to the definition of transfers instead of defining it once and for all. ∘

12.1.4 Transfer-based Fair Exchange

In Definition 12.2, a fair exchange did not require availability, i.e., an exchange scheme outputting failed in all executions is fair. However, intuitively speaking,

[7]For, e.g., off-line payment schemes, we are able to require that a withdrawal should guarantee availability of the payment. However, the notion of "withdrawal" cannot be generalized.

an exchange should take place if correct players agree and are able to transfer the offered items.

Definition 12.4 (Transfer-based Fair Exchange)
A transfer-based fair exchange scheme for two sets Σ_O and Σ_R of transferable items[8] is a tuple $(\mathsf{O}, \mathsf{R}, \mathsf{T}, \mathit{XREFs}, \mathit{TIDs})$ containing an originator machine O, a responder machine R, a third-party machine T, a set XREFs of external references, and a set TIDs of transaction identifiers.

It is called secure if the following requirements are fulfilled for all

$$I_O = (\mathsf{S_O}, \mathsf{R_O}, \mathsf{T_O}, B_O, \mathit{XREFs}_O, \mathit{TIDs}_O, \delta_O, \Delta_O)$$

with $I_O \in \Sigma_O$ and

$$I_R = (\mathsf{S_R}, \mathsf{R_R}, \mathsf{T_R}, B_R, \mathit{XREFs}_R, \mathit{TIDs}_R, \delta_R, \Delta_R)$$

with $I_R \in \Sigma_R$:

Requirement 12.4a (Fairness): $(\langle \mathsf{O}, \mathsf{S_O}, \mathsf{R_R}\rangle, \langle \mathsf{R}, \mathsf{S_R}, \mathsf{R_O}\rangle,\ B_O,\ B_R,\ \mathit{XREFs},\ \mathit{TIDs})$ together with $AM := \{\langle \mathsf{T}, \mathsf{T_O}, \mathsf{T_R}\rangle\}$ is a fair exchange scheme, i.e., the machines of the transfer-based exchange scheme and the two transferable items can be composed to a fair exchange scheme[9].

Requirement 12.4b (Availability): If machine $\mathsf{S_O}$ is in state σ_{SO} and $\mathsf{R_O}$ is in state σ_{RO} with $\delta(\sigma_{SO}, \sigma_{RO}, d_O) = \mathsf{true}$ and machine $\mathsf{S_R}$ is in state σ_{SR} and machine $\mathsf{R_R}$ is in state σ_{RR} with $\delta(\sigma_{SR}, \sigma_{RR}, d_R) = \mathsf{true}$ then the composite machines $\langle \mathsf{O}, \mathsf{S_O}, \mathsf{R_R}\rangle$ and $\langle \mathsf{R}, \mathsf{S_R}, \mathsf{R_O}\rangle$ can exchange (d_O, d_R).

Requirement 12.4c (No Loss): If a participant, say $\langle \mathsf{O}, \mathsf{S_O}, \mathsf{R_R}\rangle$, with $\delta_O(\sigma_{SO}, \sigma_{RO}, d_O)$ for the state σ_{SO} of $\mathsf{S_O}$ and any state σ_{RO} of $\mathsf{R_O}$ produces an output $[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$ on input $[\mathsf{O}^{in}_{tid}|\mathsf{exchange}(\mathsf{R}, d_O, d'_R, \mathit{xref}_O)]$, then $\delta_O(\sigma_{SO}, \sigma_{RO}, d_O)$ still holds.

A transfer-based exchange scheme is called *optimistic* if the composite third party $\langle \mathsf{T}, \mathsf{T_O}, \mathsf{T_R}\rangle$ does not participate in the "exchange"-protocol if the participants are correct and agree (i.e., they input matching parameters $\mathit{xref}_R = \mathit{xref}_O$, $d_R = d'_R$, $\mathit{tid}_O = \mathit{tid}_R$ and $d_O = d'_O$). ◇

12.2 Transferable Items Enabling Fair Exchange

We now describe different kinds of transferable items enabling fair exchange. In order to enable optimistic fair exchanges where the third party is only required for restoring fairness in case of exceptions, the concept of optimism is defined for transfers as well. Each transferable item provides at least two protocols corresponding to the phases of the exchange:

Transfer: The "transfer"-protocol tries to transfer the items without contacting the third party.

[8]Each of these sets will later define the class of items that can be exchanged using this generic protocol. Intuitively, items in Σ_O are sent by O, whereas items in Σ_R are sent by R.

[9]Note that transfer-based exchange is no fair exchange by itself. However, it is an extended service, which can be used to provide fair exchange.

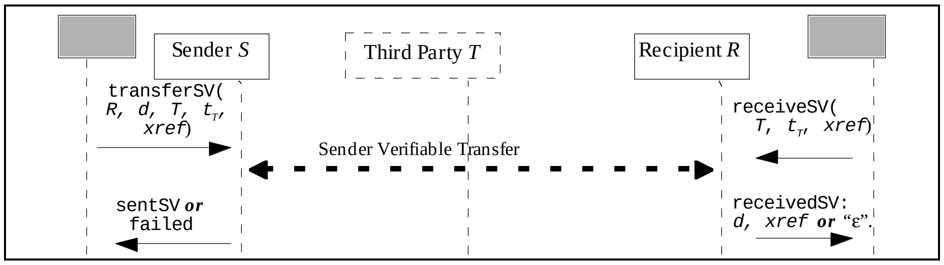

Sender S
Third Party T
Recipient R
transferSV(R, d, T, t_T, xref)
Sender Verifiable Transfer
receiveSV(T, t_T, xref)
sentSV or failed
receivedSV: d, xref or "ε".

Figure 12.4: Interactions for a Transfer with Sender Verifiability.

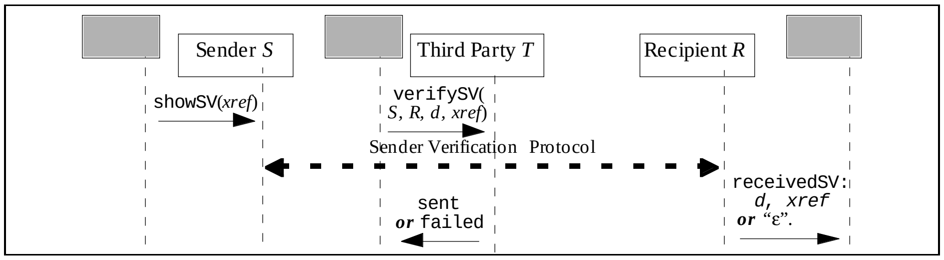


Figure 12.5: Interactions for Verifying a Transfer with Sender Verifiability.

Recovery with Third Party: After a transfer, which may have failed, the *error recovery phase* is then started if an exception, such as a wrong or missing message or a timeout, occurs. In this phase, the third party is involved and guarantees the desired exchange-enabling property of the transfer.

12.2.1 Sender Verifiability

Sender verifiability guarantees that a correct sender and the third party can make sure that a correct recipient is able to obtain an item. If sender verifiability cannot be guaranteed, the "transferSV"-protocol outputs failed to the sender and does not transfer the item.

In the optimistic case, the third party does not participate in the transfer. Therefore, the third party decides based on evidence presented during the verification protocol, i.e., the sender may present evidence and witnesses that the receiver received the item or may send the item again, whereas the receiver may present evidence and witnesses to prove that it was unable to obtain the item.

Definition 12.5 (Transferable Items with Sender Verifiability)
A *transferable item with sender verifiability* is a tuple $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_{SV}, (\Delta_t, \Delta_v))$ where machine S is the sender, R is the recipient, T is the third party, B is a business item, $XREFs$ is a set of external references, $TIDs$ is the set of transaction identifiers, and δ_{SV} is the transferability predicate, and $\Delta_t, \Delta_v \in \mathbb{N}$ are the fixed running times[10]of the protocols that need to be provided by the item:

"transferSV": This protocol is started on input of $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T,$

[10]Recall that we assumed that a correct machine produces its output at the end of the protocol and that a super-protocol waits for a sub-protocol to be completed, even if no output is made.

$xref)]$[11] and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, xref)]$ with a locally fresh and unique $tid \in TIDs$, a description $d \in D$, a future time t_T measured in terms of T's clock fixing the time until which sender verifiability shall be guaranteed, and a fresh external reference $xref \in XREFs$, which has not been used in an execution of "transferSV" between S and R before. The sender may output $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$ signaling that sender verifiability can be provided or $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ signaling failure without a transfer.

The recipient R may produce an output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon d, xref]$.

"verifySV": This protocol is started on input of $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, d, xref)]$ and $[\mathsf{S}^{in}_{tid}|\mathsf{showSV}(xref)]$ with two idenfifiers S and R, an $xref \in XREFs$, and a fresh tid. It may output $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$ or $[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$.

The recipient R may produce an output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon d, xref]$.

We define $\delta_{SV}(\sigma_S, \sigma_R, d) := \mathsf{true}$ (i.e., S in state σ_S "can transfer an item d" to R in state σ_R) iff the inputs $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, xref)]$ at any time t_0 with any fresh tid, the given T, any fresh $xref \in XREFs$, and any $t_T \in \mathbb{N}$ with $t_T \geq t_0 + \Delta_t$ to correct machines lead to an output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon d, xref]$ to a correct recipient R starting in state σ_R. The transferable item is called "optimistic" if this output is produced without contacting the third party.

If the third party is correct, the item is required to fulfill the following requirements:

Requirement 12.5a (Correct Execution): If S and R are correct and $\delta_{SV}(\sigma_S, \sigma_R, d)$ for state σ_S of S and state σ_R of R holds then the inputs $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, xref)]$ with $t_T \geq t_0 + \Delta_t$ leads to an output $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$.

If $\delta_{SV}(\sigma_S, \sigma_R, d) = \mathsf{false}$ then correct machines output $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ and $[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$.

Requirement 12.5b (Sender Verification): If a correct sender output $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$ on input $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$, then subsequent inputs of $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, d, xref)]$ and $[\mathsf{S}^{in}_{tid}|\mathsf{showSV}(xref)]$ to correct parties before time t_T lead to an output $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$.

Requirement 12.5c (Correct Verification): If the third party outputs $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$ on input $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, d, xref)]$, a correct recipient R outputs $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon d, xref]$ before time t_T.

Requirement 12.5d (Transfer): If a correct recipient R outputs $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon d, xref]$ then R owns item B with the output description d.

Requirement 12.5e (No Surprises): If the sender is correct and either no input $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ for any tid, $xref$, t_T and T was made or else an output $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ was obtained, then the recipient R cannot obtain knowledge about the item B described by d.

[11] We labeled this input transfer instead of send in order to emphasize that, usually, it does not only send data but rather starts an interactive protocol.

Requirement 12.5f (No Loss): If $\delta_{SV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ for a state σ_S of a correct S and any state ${\sigma_R}$[12], the sender input $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ for any tid, $xref$, t_T and T, and an output $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ was produced, then $\delta_{SV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ holds after time $t_T + \Delta_v$.

Requirement 12.5g (Termination): A correct player will process a correct input within the fixed number of rounds.

◇

Items may be adapted as follows to provide sender verifiability:

- In on-line public-key payment systems, such as SET, the sender may re-send the payment information to the third party, which forwards it to the bank. The bank then decides whether the payment is successful or not.

  For off-line payment schemes, the third party may be required to ask the bank on-line whether a particular coin is valid or not.

- Idempotent items, such as messages, where two transfers are equivalent to one transfer, can be sent again via the third party.

Remark 12.7. The “verifySV“-protocol may include recovery of the transfer in order to reach a consistent state: A message, e.g., may be sent again in order to ensure that a correct recipient obtained the message and that a decision sent by the third party is in fact justified.

Remark 12.8. The time t_T input by the sender defines the absolute time at T until which an input verifySV is guaranteed to produce a correct result. It is included as an argument to enable practical implementations to determine the time when data collected for guaranteeing the exchange-enabling property is no longer needed. For sender-verifiability, for example, pending evidence may be deleted after this time. As an alternative, one may define an additional input of the sender that deletes this evidence (cf. Sec. 12.2.5).

○

12.2.2 Revocability

Revocability means that the third party is able to revoke a given transfer within a certain time, i.e., render the received item unusable. This is done by executing an additional protocol “revoke“ with a third party after the actual transfer. This protocol is then required to make the item unusable no matter whether the transfer was successful or not. The interactions during the transfer are depicted in Figure 12.6 and 12.7.

Definition 12.6 (Transferable Items with Revocability)
A *transferable item with revocability* is a tuple $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_R, (\Delta_t, \Delta_r))$ where machine S is the sender, R is the recipient, T is the third party, B is a business item, $XREFs$ is a set of external references, $TIDs$ is the set of transaction identifiers, δ_R is the predicate signaling the availability of a transfer, and $\Delta_t, \Delta_r \in \mathbb{N}$ are the fixed run-times of the protocols of the item:

[12]Note that this includes the states of all machines $\mathsf{R}' \neq \mathsf{R}$.

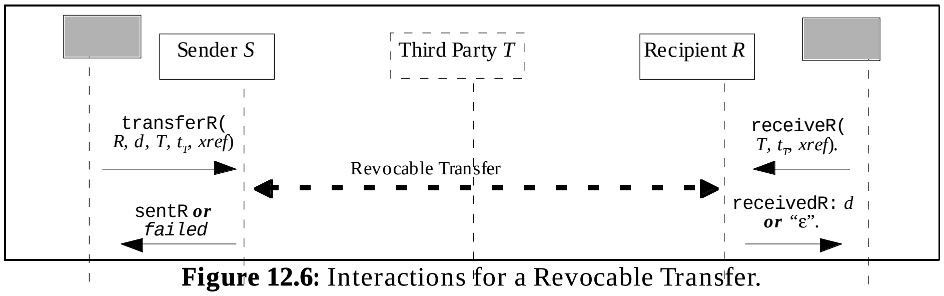


Figure 12.6: Interactions for a Revocable Transfer.

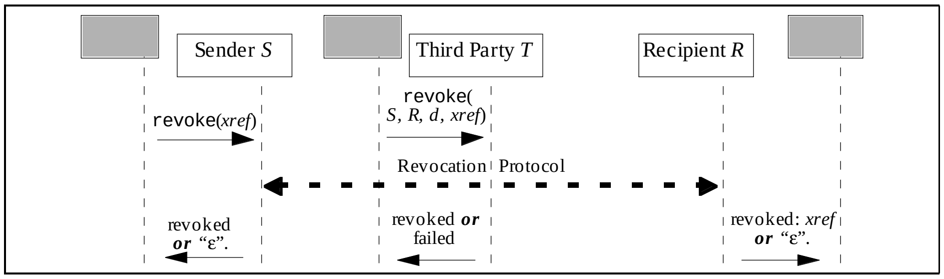


Figure 12.7: Interactions for Revoking a Transfer.

"transferR": This protocol is started on input of $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveR}(\mathsf{T}, t_T, xref)]$ at any time t_0 with a locally fresh and unique $tid \in TIDs$, a description $d \in D$, a time $t_T \in \mathbb{N}$ at machine T, and a fresh external reference $xref \in XREFs$, which has not been used in an execution of "transferR" between S and R before. The recipient R may produce an output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedR}\text{: } d]$. The sender may output $[\mathsf{S}^{out}_{tid}|\mathsf{sentR}]$ or $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$.

"revoke": This protocol is started on input of $[\mathsf{S}^{in}_{tid}|\mathsf{revoke}(xref)]$ and $[\mathsf{T}^{in}_{tid}|\mathsf{revoke}(\mathsf{S}, \mathsf{R}, d, xref)]$ with a fresh $tid \in TIDs$, two names S and R, $d \in D$, and $xref \in XREFs$. The protocol may output $[\mathsf{S}^{out}_{tid}|\mathsf{revoked}]$, $[\mathsf{T}^{out}_{tid}|\mathsf{revoked}]$ and $[\mathsf{R}^{out}_{tid}|\mathsf{revoked}\text{: } xref]$ or $[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$.

We define $\delta_R(\sigma_S, \sigma_R, d) :=$ true iff the input of $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveR}(\mathsf{T}, t_T, xref)]$ at any time t_0 with any fresh tid, the given T, any fresh $xref \in XREFs$, and any $t \in \mathbb{N}$ with $t_T \geq t_0 + \Delta_t$ to a correct S in state σ_S and a correct machine R in state σ_R leads to an output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedR}\text{: } d]$. It is called "optimistic" if this output is produced without contacting the third party.

If the third party is correct, the item is required to fulfill the following requirements:

Requirement 12.6a (Correct Execution): If S and R are correct and $\delta_R(\sigma_S, \sigma_R, d)$ for state σ_S of S and state σ_R of R holds, then the inputs $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveR}(\mathsf{T}, t_T, xref)]$ with $t_T \geq t_0 + \Delta_t$ leads to an output $[\mathsf{S}^{out}_{tid}|\mathsf{sentR}]$.

If $\delta_R(\sigma_S, \sigma_R, d) =$ false then correct machines output $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ and $[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$.

Requirement 12.6b (Correct Revocation): If a correct sender output $[\mathsf{S}^{out}_{tid}|\mathsf{sentR}]$ on input $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$, the subsequent inputs $[\mathsf{S}^{in}_{tid}|\mathsf{revoke}(xref)]$ and $[\mathsf{T}^{in}_{tid}|\mathsf{revoke}(\mathsf{S}, \mathsf{R}, d, xref)]$ before time t_T at T lead to the outputs $[\mathsf{S}^{out}_{tid}|\mathsf{revoked}]$ and $[\mathsf{T}^{out}_{tid}|\mathsf{revoked}]$.

If R is correct, it obtains an output $[\mathsf{R}^{out}_{tid}|\mathsf{revoked}\text{: } xref]$ before time $t_T + \Delta_r$.

Requirement 12.6c (Transfer): If a correct recipient R outputs $[\mathsf{R}^{out}_{tid}|\mathsf{receivedR}\text{: } d]$ on input $[\mathsf{R}^{in}_{tid}|\mathsf{receiveR}(\mathsf{T}, t_T, xref)]$ and does not output $[\mathsf{R}^{out}_{tid}|\mathsf{revoked}\text{: } xref]$ before time $t_T + \Delta_r$ then R owns the item B for the output description d after time $t_T + \Delta_r$.

Requirement 12.6d (No Surprises): If the sender and the third party are correct and either no input $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ for any tid, $xref$, T, and $t_T \in \mathbb{N}$ was made, or else an output $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ or $[\mathsf{T}^{out}_{tid}|\mathsf{revoked}]$ was produced, the recipient is not able to obtain knowledge about the item B described by d.

Requirement 12.6e (No Loss): If $\delta_R(\sigma_S, \sigma_R, d) = \mathsf{true}$ for a state σ_S of a correct S and any state σ_R of R, the sender input $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ for any tid and $xref$, T and $t_T \in \mathbb{N}$, and $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ or $[\mathsf{T}^{out}_{tid}|\mathsf{revoked}]$ was output, then $\delta_R(\sigma_S, \sigma_R, d) = \mathsf{true}$ holds after time $t_T + \Delta_r$.

Requirement 12.6f (Termination): A correct player will process a correct input within the fixed number of rounds.

$\diamond$

Some examples of revocability are:

- Credit-card payments can usually be revoked within a certain time.
- Revocability may be added to data by encrypting it with the key of the third party. Then, a correct sender would automatically send the data at time $t_T + \Delta_r$. If not, the third party would decrypt it, if it was not revoked. (Note that in our terms, this is similar to a simulation of revocability using generatability.)
- A signature can be made revocable by means of revocation lists: In addition to the signed message, the signer signs the condition that this signature is only binding after time $t_T + \Delta_r$ if no revocation entry signed by T was posted at a given revocation list.

12.2.3 Recipient Verifiability

Recipient verifiability guarantees that a recipient can convince the third party that it was unable to obtain a particular item fixed in a "$\mathsf{prepareRV}$"-protocol, even if it would have misbehaved.

The interactions for a transfer with recipient verifiability are depicted in Figure 12.8. The interactions for its verification are depicted in Figure 12.9.

Definition 12.7 (Transferable Items with Recipient Verifiability)
A *transferable item with recipient verifiability* is a tuple $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_{RV}, (\Delta_p, \Delta_t, \Delta_v))$ where machine S is the sender, R is the recipient, T

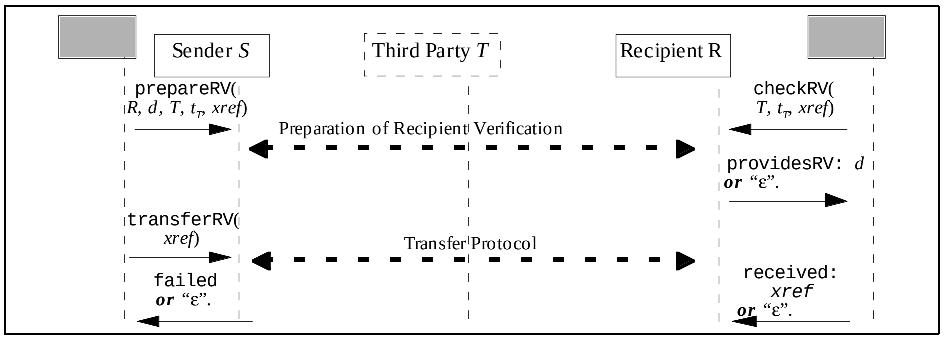


Figure 12.8: Interactions for a Transfer with Recipient Verifiability.

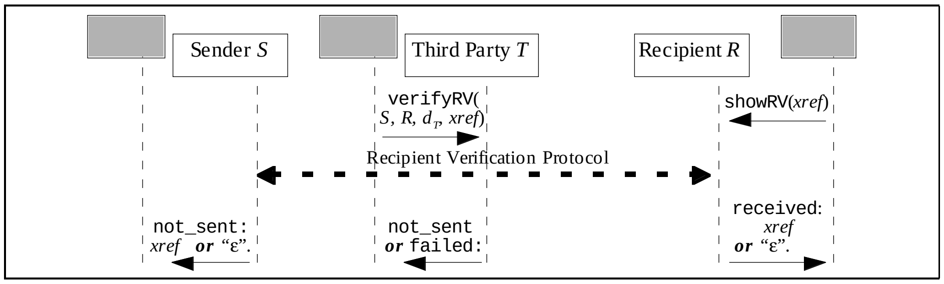


Figure 12.9: Interactions for Verifying a Transfer with Recipient Verifiability.

is the third party, B is a business item, $XREFs$ is a set of external references, $TIDs$ is the set of transaction identifiers, δ_{RV} is the transferability predicate, and $\Delta_p, \Delta_t, \Delta_v \in \mathbb{N}$ are the fixed run-times of the protocols of the item:

"prepareRV": This protocol is started on input of $[\mathsf{S}_{tid}^{in}|\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}_{tid}^{in}|\mathsf{checkRV}(\mathsf{T}, t_T, xref)]$ at any time t_0 where T is the machine to be used as the third party, R is the intended recipient of the transfer, d is the description under which the item will be transferred, t_T is the time at T until which recipient verifiability shall be guaranteed, and $xref \in XREFs$ is a fresh external reference, which has not been used in an execution of "prepareRV" between S and R before.

The recipient may obtain an output $[\mathsf{R}_{tid}^{out}|\mathsf{providesRV}\text{: } d]$.

"transferRV": This protocol is started on input of $[\mathsf{S}_{tid}^{in}|\mathsf{transferRV}(xref)]$ with a fresh tid an $xref \in XREFs$. The recipient may produce an output $[\mathsf{R}_{tid}^{out}|\mathsf{received}\text{: } xref]$. The sender may produce an output $[\mathsf{S}_{tid}^{out}|\mathsf{failed}]$.

"verifyRV": This protocol is started on input of $[\mathsf{T}_{tid}^{in}|\mathsf{verifyRV}(\mathsf{S}, \mathsf{R}, d, xref)]$ and $[\mathsf{R}_{tid}^{in}|\mathsf{showRV}(xref)]$ with a fresh tid, two names S and R, and an $xref \in XREFs$. The protocol may either output $[\mathsf{T}_{tid}^{out}|\mathsf{not\_sent}]$ and $[\mathsf{S}_{tid}^{out}|\mathsf{not\_sent}\text{: } xref]$ or else $[\mathsf{T}_{tid}^{out}|\mathsf{failed}]$. Furthermore, the protocol may output $[\mathsf{R}_{tid}^{out}|\mathsf{received}\text{: } xref]$ to the recipient.

We define $\delta_{RV}(\sigma_S, \sigma_R, d) :=$ true iff the input of $[\mathsf{S}_{tid}^{in}|\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}_{tid}^{in}|\mathsf{checkRV}(\mathsf{T}, t_T, xref)]$ at any time t_0 and $[\mathsf{S}_{tid}^{in}|\mathsf{transferRV}(xref)]$ after time $t_0 + \Delta_p$ to correct players with a given T, any fresh $tid \in TIDs$ and $xref \in XREFs$, and $t_T \in \mathbb{N}$ with $t_T \geq t_0 + \Delta_p + \Delta_t$ leads to the output $[\mathsf{R}_{tid}^{out}|$

received: $xref$] to a correct recipient R starting in state σ_R. The transferable item is called "optimistic" if this output is produced without contacting the third party.

If the third party is correct, the item is required to fulfill the following requirements:

Requirement 12.7a (Correct Execution): If S and R are correct and $\delta_{RV}(\sigma_S, \sigma_R, d)$ for state σ_S of S and state σ_R of R holds then the input [S^{in}_{tid}|prepareRV(R, d, T, t_T, $xref$)] and [R^{in}_{tid}|checkRV(T, t_T, $xref$)] at any time t_0 with $t_T \geq t_0 + \Delta_p$ leads to an output [R^{out}_{tid}|providesRV: d] to a correct recipient.

If $\delta_{RV}(\sigma_S, \sigma_R, d) =$ false then [S^{out}_{tid}|failed] is output.

Requirement 12.7b (Correct Verification): If a correct recipient output [R^{out}_{tid}| providesRV: d] on input [R^{in}_{tid}|checkRV(T, t_T, $xref$)] and does not output [R^{out}_{tid}|received: $xref$], the inputs [T^{in}_{tid}|verifyRV(S, R, d, $xref$)] and [R^{in}_{tid}| showRV($xref$)] to correct parties before time t_T imply that [T^{out}_{tid}|not_sent] or [R^{out}_{tid}|received: $xref$] is output[13].

If the sender is correct and the correct third party outputs [T^{out}_{tid}|not_sent], the sender outputs [S^{out}_{tid}|not_sent: $xref$].

Requirement 12.7c (Transfer): If a correct recipient R outputs [R^{out}_{tid}|received: $xref$] and does not input [R^{in}_{tid}|showRV($xref$)], then R owns the item B with the output description d.

Requirement 12.7d (No Surprises): If the sender is correct and either no input [S^{in}_{tid}|prepareRV(R, d, T, t_T, $xref$)] for any tid, $xref$, t_T and T was made or else, no input [S^{in}_{tid}|transferRV($xref$)] without an output [S^{out}_{tid}|failed] was made and no input [T^{in}_{tid}|verifyRV(S, R, d, $xref$)] without an output [T^{out}_{tid}| not_sent] was made, then the recipient R is not able to obtain knowledge about the item B described by d.

Requirement 12.7e (No Loss): If $\delta_{RV}(\sigma_S, \sigma_R, d) =$ true for a state σ_S of a correct machine S and any state σ_R and either no input [S^{in}_{tid}|prepareRV(R, d, T, t_T, $xref$)] for any tid, $xref$, t_T and T was made, or else no input [S^{in}_{tid}| transferRV($xref$)] without an output [S^{out}_{tid}|failed] was made and no input [T^{in}_{tid}|verifyRV(S, R, d, $xref$)] without an output [T^{out}_{tid}|not_sent] was made then $\delta_{RV}(\sigma_S, \sigma_R, d) =$ true holds after time $t_T + \Delta_v$.

Requirement 12.7f (Termination): A correct player will process a correct input within the fixed number of rounds.

◇

Some examples of recipient verifiability are:

- Together with the bank, a payment can be made recipient verifiable by fixing it during "prepareRV". Then, if this particular payment was not deposited before "showRV" is executed, the third party will decide on not_sent and a later deposit of this particular payment will not be accepted anymore.

[13]Recall that received may be output in "transferRV" or "verifyRV".

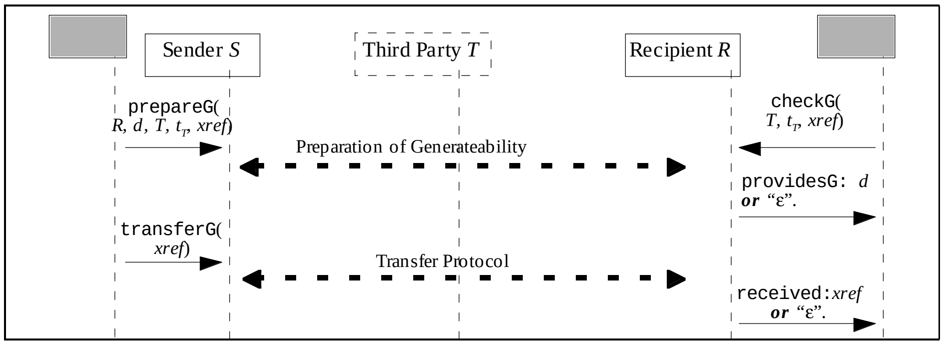


Figure 12.10: Interactions for a Transfer with Generatability.

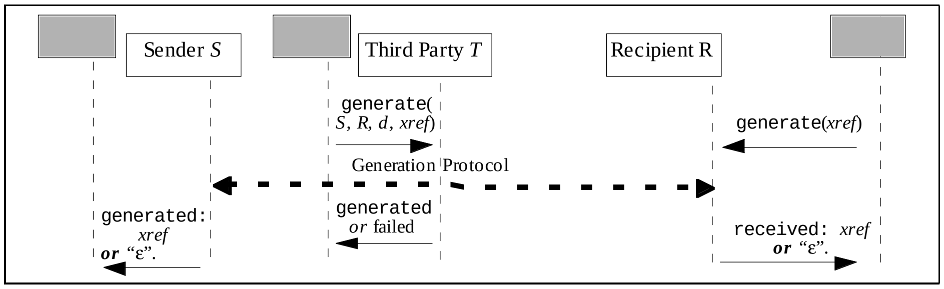


Figure 12.11: Interactions for Generating a Transfer.

- A credit-card payment can usually be revoked by the bank. After revocation, the third party can then decide on not_sent.
- For messages, the message may be fixed during "prepareRV" using a commitment. If the sender refuses to re-send the message and open the commitment during "showSV", the third party may decide on not_sent.

Remark 12.9. During "prepareRV", items may need to be locked or evidence stored to enable later exploitation of the property. Therefore, for cleaning up, we again included a time t_T until which the property can be exploited. ○

12.2.4 Generatability

Generatability means that the sender is able to authorize a third party to complete or redo a transfer in a given context "$xref$" on its behalf.

In order to authorize the third party to generate an item, the sender and the recipient execute the "prepareG"-protocol. Then, after successful completion, the third party can replace the transfer by executing the "generate"-protocol, i.e., do or redo the transfer on behalf of an incorrect or absent sender.

The activities are depicted in Figure 12.10 and 12.11. Again, the third party may not participate in the "prepareG" and "transferG" protocols if the implementation is optimistic.

Definition 12.8 (Transferable Items with Generatability)
A *transferable item with generatability* is a tuple $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_G, (\Delta_p, \Delta_t, \Delta_g))$ where machine S is the sender, R is the recipient, T

is the third party, B is a business item, $XREFs$ is a set of external references, $TIDs$ is the set of transaction identifiers, δ_G is the transferability predicate, and $\Delta_p, \Delta_t, \Delta_g \in \mathbb{N}$ are the fixed run-times of the protocols of the item:

"prepareG": This protocol is started on input of $[\mathsf{S}^{in}_{tid}|\mathsf{prepareG}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$ at any time t_0 where T is the machine to be used as third-party, R is the intended recipient of the transfer, d is the description of the item, $t_T \in \mathbb{N}$ is the time at T until which generatability shall be guaranteed, and $xref \in XREFs$ is a fresh external reference, which has not been used in an execution of "prepareG" between S and R before.

The recipient may output $[\mathsf{R}^{out}_{tid}|\mathsf{providesG}\colon d]$.

"transferG": This protocol is started on input of $[\mathsf{S}^{in}_{tid}|\mathsf{transferG}(xref)]$ with a fresh tid and an $xref \in XREFs$. The recipient may output $[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$.

"generate": This protocol is started on input of $[\mathsf{R}^{in}_{tid}|\mathsf{generate}(xref)]$ and $[\mathsf{T}^{in}_{tid}|\mathsf{generate}(\mathsf{S}, \mathsf{R}, d, xref)]$ with two names S and R, a description $d \in D$, and an $xref \in XREFs$ by the third party after time $t_0 + \Delta_p$ and before time t_T. The protocol may either output $[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$, $[\mathsf{T}^{out}_{tid}|\mathsf{generated}]$, and $[\mathsf{S}^{out}_{tid}|\mathsf{generated}\colon xref]$ or else $[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$.

We define $\delta_G(\sigma_S, \sigma_R, d) :=$ true iff the input of $[\mathsf{S}^{in}_{tid}|\mathsf{prepareG}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$ at any time t_0 to a correct player with a given T, any fresh $tid \in TIDs$ and $xref \in XREFs$, and $t_T \in \mathbb{N}$ with $t_T \geq t_0 + \Delta_p + \Delta_t$ leads to the output $[\mathsf{R}^{out}_{tid}|\mathsf{providesG}\colon d]$ to a correct recipient R starting in state σ_R.[14] The transferable item is called "optimistic" if this output is produced without contacting the third party and a subsequent execution of the "transferG"-protocol does not involve the third party.
If the third party is correct, the protocols are required to fulfill the following requirements:

Requirement 12.8a (Correct Execution): If correct parties input $[\mathsf{S}^{in}_{tid}|\mathsf{prepareG}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$ at the same time and the recipient outputs $[\mathsf{R}^{out}_{tid}|\mathsf{providesG}\colon d]$, then the input $[\mathsf{S}^{in}_{tid}|\mathsf{transferG}(xref)]$ after time $t_0 + \Delta_p$ leads to an output $[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$ at a correct recipient.

Requirement 12.8b (Correct Generation): If a correct recipient obtained an output $[\mathsf{R}^{out}_{tid}|\mathsf{providesG}\colon d]$ on input $[\mathsf{R}^{in}_{tid}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$, then the subsequent input of $[\mathsf{R}^{in}_{tid}|\mathsf{generate}(xref)]$ and $[\mathsf{T}^{in}_{tid}|\mathsf{generate}(\mathsf{S}, \mathsf{R}, d, xref)]$ by correct players before time t_T lead to the outputs $[\mathsf{T}^{in}_{tid}|\mathsf{generated}()]$ and $[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$.

If S is correct, it outputs $[\mathsf{S}^{out}_{tid}|\mathsf{generated}\colon xref]$ before time $t_T + \Delta_g$.

Requirement 12.8c (Transfer): If a correct recipient R outputs $[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$ after an output $[\mathsf{R}^{out}_{tid}|\mathsf{providesG}\colon d]$ on input $[\mathsf{R}^{in}_{tid}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$, then R owns B with the output description d.

[14] Note that the recipient knows whether the sender is able to transfer the item after "prepareG". This is different from recipient verifiability, where preparation may succeed even if the sender is unable to transfer the item. In this case, however, successful preparation is still required to guarantee correct verification.

Requirement 12.8d (No Surprises): If the sender is correct and either no input $[\mathsf{S}^{in}_{tid}|\mathsf{prepareG}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ for any tid, $xref$, t_T and T was made, or else neither $[\mathsf{S}^{in}_{tid}|\mathsf{transferG}(xref)]$ nor $[\mathsf{T}^{in}_{tid}|\mathsf{generate}(\mathsf{S}, \mathsf{R}, d, xref)]$ was input, then the recipient R is not able to obtain knowledge about the item B described by d.

Requirement 12.8e (No Loss): If $\delta_G(\sigma_S, \sigma_R, d) = \mathsf{true}$ for a state σ_S of a correct S and any state σ_R and S inputs $[\mathsf{S}^{in}_{tid}|\mathsf{prepareG}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$ and neither $[\mathsf{S}^{in}_{tid}|\mathsf{transferG}(xref)]$ nor $[\mathsf{T}^{in}_{tid}|\mathsf{generate}(\mathsf{S}, \mathsf{R}, d, xref)]$ was input, then $\delta_G(\sigma_S, \sigma_R, d) = \mathsf{true}$ holds after time $t_T + \Delta_g$.

Requirement 12.8f (Termination): A correct player will process a correct input within the fixed number of rounds.

A generatable item is called "optimistic" if the "prepareG" and "transferG" protocols executed by a correct sender being able to transfer an item and a correct recipient output providesG and received without involving the third party. $\diamond$

Some examples of recipient verifiability are:

- For signatures, generatability can be provided by authorizing the third party to sign an affidavit, i.e., the sender signs "I herewith authorize the third party to sign contract C in the transaction $xref$ on my behalf". Then, this authorization together with the signature of T will later be accepted as a valid contract[15].

- Data can be made generatable by encrypting it with the third party's public key, i.e., during "prepareG", a message $\mathrm{sign}_{\mathsf{S}}(xref, \mathsf{R}, E_T(r; xref, \mathsf{R}, data))$ is sent to the recipient (a random number r is used for randomizing the encryption). Later, the third party decrypts this message. If the decryption fails, the data is assumed to be empty. Else, the third party sends the data to the recipient.

Remark 12.10. For generatability, the non-repudiation of the external reference is of particular importance: Without this reference, a generatable transfer could be moved from one exchange (e.g., buyer with seller) to another (e.g., buyer and buyer), thus enabling one participant (e.g., the buyer) to ask the third party to generate the item in exchange for the item that is transferred to itself instead of the intended recipient. $\circ$

12.2.5 Towards Asynchronous Transfers

For simplicity, all exchange-enabling transfers were defined for synchronous networks only. However, we are convinced that they can be adapted for asynchronous networks, too. In this case, the time-outs would be replaced by additional "unprepare" protocols to release pending evidence and allow re-use of locked items.

On asynchronous networks, the protocols including incorrect parties may not terminate since pending messages may not be received and therefore, no result can be produced. Therefore, for adapting the transfers to asynchronous

[15]Less intrusive generatability for signatures can be provided by using verifiable encryption [AsSW1 98].

networks, we would apply the approach used in Part I: The "prepare" and "transfer"-protocols can at any time be terminated by an input wakeup. After this input, the protocol is required to output a result within a fixed time (i.e., the protocol must not wait for messages from parties that may be incorrect but rather terminate while only interacting with parties known to be correct). In order to guarantee this, the recovery protocols (i.e., "verifySV", "revoke", "verifyRV", and "generate") must not contact parties that may be incorrect (cf. Lemma 3.2).

As we will later show, sender and recipient verifiability are equivalent on synchronous networks. This, however, will not be the case for asynchronous networks: For sender verifiability, the evidence is usually stored at the sender (e.g., the message to be sent again). Therefore, to provide recipient verifiability based on sender verifiability, one has to contact the sender that may be incorrect. This is not possible on asynchronous networks.

12.3 Examples of Transferable Items

To illustrate our generic approach for transferable items, we now describe the examples of transferable items necessary to instantiate the most common instances of fair exchange, namely contract signing, certified mail, and fair purchase.

Note that we only describe some selected exchange-enabling properties for each transferable item. Further properties can either be provided by additional protocols or else by using the simulations described in Section 13.3.

12.3.1 Signatures

We now describe sender-verifiable and generatable transfers of digital signatures.

For sender-verifiability, a sender transfers a signature by signing the input message. Sender verifiability is provided by re-sending the signature. An alleged owner executes the "use$_\mathsf{i}$"-protocol by sending the signature to the user who verifies them.

Scheme 12.1 (Signatures with Sender Verifiability)
Let $\mathrm{sign}_\mathsf{N}(msg)$ be the secure digital signature of a machine $\mathsf{N} \in id\_space$ under $msg \in Msg$. We assume that each machine can only sign under its own name and that all machines can verify all signatures.

We define a scheme $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_{SV}, (\Delta_t, \Delta_v))$ for sender-verifiable signatures with $B = (\mathsf{U}, D, TIDs)$, $D := id\_space \times Msg$, and $TIDs := \{0,1\}^*$ as follows:

Using a Signed Message (Protocol "use$_\mathsf{i}$"*):* On input $[\mathsf{R}_{tid}^{in}|\mathsf{use_i}((\mathsf{X}, msg))]$, the recipient checks whether it knows an m_2 containing $\mathrm{sign}_\mathsf{X}(msg, \mathsf{R})$. If this is the case, it sends $(\mathrm{sign}_\mathsf{X}(msg, \mathsf{R}), i, tid)$ to U.

On receipt of a message, the user checks whether this message has the form $(\mathrm{sign}_\mathsf{X}(msg, \mathsf{R}), i, tid)$ for an $\mathsf{X} \in id\_space$, $xref \in XREFs$, $msg \in Msg$, $i \in \mathbb{N}$, and $tid \in TIDs$. If this is the case, it outputs $[\mathsf{U}_{tid}^{out}|\mathsf{used_i}\colon \mathsf{R}, (\mathsf{X}, msg)]$.

Sending a Signature (Protocol “transferSV“*):* On input $[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, \mathit{xref})]$, machine R sends $m_1 := \mathrm{sign}_{\mathsf{R}}(\mathsf{T}, t_T, \mathit{xref}, \mathit{tid})$ to S. On input $[\mathsf{S}^{in}_{tid}| \mathsf{transferSV}(\mathsf{R}, (\mathsf{X}, \mathit{msg}), \mathsf{T}, t_T, \mathit{xref})]$, machine S checks whether $\mathsf{S} = \mathsf{X}$ and waits for m_1. If $\mathsf{S} \neq \mathsf{X}$ or m_1 is not received, it outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$. Else, it sends a message $m_2 := (\mathrm{sign}_{\mathsf{S}}(\mathit{msg}, \mathsf{R}), m_1)$ to the recipient R and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$. Furthermore, it stores m_2 under xref.

Upon receipt and successful verification, the recipient outputs $[\mathsf{R}^{out}_{tid}| \mathsf{receivedSV}\colon (\mathsf{S}, \mathit{msg}), \mathit{xref}]$.

Sender Verification (Protocol “verifySV“*:* On input of $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, (\mathsf{S}_T, \mathit{msg}_T), \mathit{xref}_T)]$ and $[\mathsf{S}^{in}_{tid}|\mathsf{showSV}(\mathit{xref})]$, the sender retrieves m_2 and sends it to T who verifies it. If it does not match with the input parameters (i.e., $\mathit{msg}_T = \mathit{msg}$, $\mathsf{S}_T = \mathsf{S}$, and $\mathit{xref}_T = \mathit{xref}$), it outputs $[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$. Else, it sends $m'_2 := m_2$ to R and outputs $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$.

If R receives a correct m'_2, it outputs $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon (\mathsf{S}, \mathit{msg}), \mathit{xref}]$[16].

We define $\Delta_t := 2$, $\Delta_v := 2$. ◇

Lemma 12.1
Scheme 12.1 is a transferable item with sender verifiability (Def. 12.5). □

Proof. It is clear that the scheme is optimistic and that $\delta_{SV}(\sigma_S, \sigma_R, (\mathsf{X}, \mathit{msg})) =$ true for any state σ_S of machine S and any state σ_R iff $S = X$.

We now show that Scheme 12.1 fulfills the requirements of Definition 12.5:

Correct Execution (R. 12.5a): The input of $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, (\mathsf{X}, \mathit{msg}), \mathsf{T}, t_T, \mathit{xref})]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, \mathit{xref})]$ leads to an output $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$, if $\mathsf{S} = \mathsf{X}$.

Sender Verification (R. 12.5b): If a correct sender output $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$, it is able to send m'_2 (it stored the necessary information). Therefore, the third party will output sent for the same parameters.

Correct Verification (R. 12.5c): If the third party outputs $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$ on input $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, (\mathsf{S}, \mathit{msg}), \mathit{xref})]$, it forwarded a message m'_1 matching the input parameters. Therefore, a correct recipient outputs $[\mathsf{R}^{out}_{tid}| \mathsf{receivedSV}\colon (\mathsf{S}, \mathit{msg}), \mathit{xref}]$

Transfer (R. 12.5d): If a correct recipient output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon (\mathsf{S}, \mathit{msg}), \mathit{xref}]$, it knows $\mathrm{sign}_{\mathsf{S}}(\mathit{msg})$. Therefore, the user will output used_i for any $i \in \mathbb{N}$.

No Surprises (R. 12.5e): If the sender does not input $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, (\mathsf{S}, \mathit{msg}), \mathsf{T}, t_T, \mathit{xref})]$ or if $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ is output, m_1 is not sent. From the security of the signature scheme follows that the recipient is unable to show $\mathrm{sign}_{\mathsf{S}}(\mathit{msg})$ to the user U.

No Loss (R. 12.5f): δ_{SV} never changes.

Termination (R. 12.5g): All protocols terminate within the given time.

[16]Note that for a incorrect sender, this output based on m'_2 may contain a different xref than m_2. This, however, does not contradict our requirements, since the protocol makes sure that receivedSV was output for the xref input by T.

■

We now describe a generatable transfer for signatures. In order to provide generatability, the sender authorizes the third party to sign on its behalf by executing the "prepareG"-protocol. Then, if it does not send its signature in "transferG", the third party can issue an affidavit using the authorization produced during "prepareG".

Scheme 12.2 (Signatures with Generatability)
Let $\mathrm{sign}_{\mathsf{N}}(msg)$ be the secure digital signature of a machine $\mathsf{N} \in id\_space$ under $msg \in Msg$. We assume that each machine can only sign under its own name and all machines can verify all signatures. Let T be the third party as used in Scheme 12.2.

We define a scheme $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_G, (\Delta_p, \Delta_t, \Delta_g))$ for generatable signatures with $B = (\mathsf{U}, D, TIDs)$ with $D := id\_space \times Msg$ and $TIDs := \{0,1\}^*$ as follows:

Preparing Generatability (Protocol "prepareG"*):* On input $[\mathsf{R}^{in}_{tid}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$, the recipient sends $m_1 := \mathrm{sign}_{\mathsf{R}}(\mathsf{T}, t_T, xref, tid)$ to S. On input $[\mathsf{S}^{in}_{tid}|\mathsf{prepareG}(\mathsf{R}, (\mathsf{X}, msg), \mathsf{T}, t_T, xref)]$, machine S checks whether $\mathsf{S} = \mathsf{X}$ and waits for m_1. If this is the case and m_1 is received, it sends a message $m_2 := (\mathrm{sign}_{\mathsf{S}}(msg, \mathsf{gen}, m_1), tid)$ to the recipient R. Furthermore, it stores m_2 under $xref$.

Upon receipt and successful verification, the recipient outputs $[\mathsf{R}^{out}_{tid}|$ $\mathsf{providesG}\text{: } (\mathsf{S}, msg)]$ and stores m_2.

Transferring a Signature (Protocol "transferG"*):* On input of $[\mathsf{S}^{in}_{tid}|\mathsf{transferG}(xref)]$, machine S looks up m_2 and sends $m_3 := (\mathrm{sign}_{\mathsf{S}}(msg, \mathsf{R}), xref, tid)$ to the recipient R. Upon receipt and successful verification, the recipient outputs $[\mathsf{R}^{out}_{tid}|\mathsf{received}\text{: } xref]$.

Generation (Protocol "generate"*):* On input $[\mathsf{R}^{in}_{tid}|\mathsf{generate}(xref)]$ and $[\mathsf{T}^{in}_{tid}|$ $\mathsf{generate}(\mathsf{S}, \mathsf{R}, (\mathsf{S}_T, msg_T), xref_T)]$, the recipient retrieves m_2 and sends it to the third party. The third party checks the contained parameters and checks whether its current time is no larger than t_T. If this is not the case, it outputs $[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$. Else, it outputs $[\mathsf{T}^{out}_{tid}|\mathsf{generated}]$ and sends $m_4 := \mathrm{sign}_{\mathsf{T}}(m_2)$ to R and S who output $[\mathsf{R}^{out}_{tid}|\mathsf{received}\text{: } xref]$ and $[\mathsf{S}^{out}_{tid}|$ $\mathsf{generated}\text{: } xref]$, respectively.

Using a signed message (Protocol "$\mathsf{use_i}$"*):* On input $[\mathsf{R}^{in}_{tid}|\mathsf{use_i}((\mathsf{X}, msg))]$, the recipient retrieves m_3 or m_4 and sends it with R and i to U with i, tid appended.

On receipt of a message, the user checks whether this message has the form $(\mathrm{sign}_{\mathsf{S}}(msg), i, tid, \mathsf{R})$ or $(\mathrm{sign}_{\mathsf{T}}(\mathrm{sign}_{\mathsf{S}}(msg, \mathsf{gen}, \mathrm{sign}_{\mathsf{R}}(\mathsf{T}, t_T, xref, tid)), tid))$ for a $\mathsf{S} \in id\_space$, $xref \in XREFs$, $msg \in Msg$, $i \in \mathbb{N}$, and $tid \in TIDs$. If this is the case, it outputs $[\mathsf{U}^{out}_{tid}|\mathsf{used_i}\text{: } \mathsf{R}, (\mathsf{X}, msg)]$.

We define $\Delta_p := 2$, $\Delta_t := 1$, $\Delta_g := 2$. ◇

Lemma 12.2
Scheme 12.2 is a transferable item with generatability (Def. 12.8). □

Proof. It is clear that the scheme is optimistic and that $\delta_G(\sigma_S, \sigma_R, (\mathsf{X}, msg)) =$ true for any state σ_S of machine S and any state σ_R iff $S = X$.

We now show that Scheme 12.2 fulfills the requirements of Definition 12.8:

Correct Execution (R. 12.8a): If the recipient output $[\mathsf{R}_{tid}^{out}|\mathsf{providesG\colon}\ (\mathsf{S}, msg)]$ on input of $[\mathsf{S}_{tid}^{in}|\mathsf{prepareG}(\mathsf{R}, (\mathsf{S}, msg), \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}_{tid}^{in}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$, the sender has stored m_2 and sends m_3 on input input $[\mathsf{S}_{tid}^{in}|\mathsf{transferG}(xref)]$. Therefore, the recipient will output $[\mathsf{R}_{tid}^{out}|\mathsf{received\colon}\ xref]$.

Correct Generation (R. 12.8b): If a correct recipient output $[\mathsf{R}_{tid}^{out}|\mathsf{providesG\colon}\ (\mathsf{S}, msg)]$ on input of $[\mathsf{S}_{tid}^{in}|\mathsf{prepareG}(\mathsf{R}, (\mathsf{S}, msg), \mathsf{T}, t_T, xref)]$ and $[\mathsf{R}_{tid}^{in}|\mathsf{checkG}(\mathsf{T}, t_T, xref)]$ it knows a corresponding m_2. After sending this message to T before time t_T, on input $[\mathsf{T}_{tid}^{in}|\mathsf{generate}(\mathsf{S}, \mathsf{R}, (\mathsf{S}, msg), xref)]$, T will sign m_4 and R will output $[\mathsf{R}_{tid}^{out}|\mathsf{received\colon}\ xref]$. Since m_4 is sent to S as well, a correct sender will output $[\mathsf{S}_{tid}^{out}|\mathsf{generated\colon}\ xref]$.

Transfer (R. 12.8c): If a recipient output $[\mathsf{R}_{tid}^{out}|\mathsf{received\colon}\ xref]$ after an output $[\mathsf{R}_{tid}^{out}|\mathsf{providesG\colon}\ (\mathsf{S}, msg)]$, it knows m_3 or m_4, which are sufficient for using the item.

No Surprises (R. 12.8d): If $[\mathsf{S}_{tid}^{in}|\mathsf{prepareG}(\mathsf{R}, (\mathsf{S}, msg), \mathsf{T}, t_T, xref)]$ is not input, the recipient does not receive m_2 or m_3 containing R. If $[\mathsf{S}_{tid}^{in}|\mathsf{prepareG}(\mathsf{R}, (\mathsf{S}, msg), \mathsf{T}, t_T, xref)]$ is input and neither $[\mathsf{S}_{tid}^{in}|\mathsf{transferG}(xref)]$ nor $[\mathsf{T}_{tid}^{in}|\mathsf{generate}(\mathsf{S}, \mathsf{R}, (\mathsf{S}, msg), xref)]$, then the recipient obtains neither m_3 nor m_4. From the security of the signature scheme follows that message m_2 obtained by R does not contain knowledge useful for using (S, msg), since it does not help to guess m_3 or m_4.

No Loss (R. 12.8e): δ_G never changes.

Termination (R. 12.8f): All protocols have a fixed run-time.

■

Remark 12.11. This simple mechanism for generatability defines a replacement signature m_4 to be equivalent to the original signature m_3 by changing the user U. If one does not want to change the "$\mathsf{use_i}$" protocol, one can add generatability by means of verifiable encryption of the signature [AsSW1 98]: Here, during "prepareG", the sender encrypts the signature with the third party's public key and proves the correctness of the ciphertext to the recipient. Later, during "generate", the third party decrypts the ciphertext provided by the recipient and thus obtains the real signature $\mathrm{sign}_{\mathsf{S}}(msg)$. ○

12.3.2 Untraceable On-line Payments

A digital coin [Chau 89] is essentially a so-called blind signature under a coin number. It is described by its currency and its denomination.

We now describe sender-verifiable payments based on the untraceable on-line payment scheme described in [Chau 89]. In a sender-verifiable transfer, the recipient first fixes the coin numbers to be transferred. If this is done, the payment is executed. In order to link the individual coins to the parameters of the transfer (in particular R, t_T and $xref$), we use public keys as coin numbers and then authenticate the parameters with the corresponding secret keys.

Note that we again included the definition of the business item B into the definition of the payment with sender-verifiability. We assume that the value of one coin is, e.g., \$1.

Scheme 12.3 (On-line Payments with Sender Verifiability)
Let $\text{sign}_{\mathsf{N}}(msg)$ be the secure digital signature of a machine $\mathsf{N} \in id\_space$ under $msg \in Msg$. Let (pk_B, sk_B) be the public and private key of an RSA scheme, i.e., $pk_B(sk_B(msg)) = sk_B(pk_B(msg)) = msg$ and $sk_B(a \cdot b) = sk_B(a) \cdot sk_B(b)$ for all messages $m \in Msg$ for a given set Msg.

We define a scheme $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_{SV}, (\Delta_t, \Delta_v))$ with $B = (\langle \mathsf{U}, \mathsf{B} \rangle, D, TIDs)$ for sender-verifiable on-line coins as follows:

Withdrawing Coins (Protocol “withdraw“*):* On input of $[\mathsf{S}^{in}_{tid}|\mathsf{withdraw}(\$n)]$, the sender sends a list $(x_1, \ldots, x_n)$ to the bank B with $x_i := n_i pk_B(b_i)$ where all b_i’s are randomly chosen and the n_i are an encoding of public keys pk_i corresponding to randomly chosen secret keys sk_i of the given signature scheme.

The bank then debits $\$n$ from the sender’s account and sends $(y_i, \ldots, y_n)$ to S with $y_i := sk_B(x_i)$.

The sender computes the signed coins and stores them:

$$z_i := y_i / b_i = sk_B(n_i pk_B(b_i))/b_i = sk_B(n_i).$$

Sending a Payment (Protocol “transferSV“*):* On input $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, \$k, \mathsf{T}, t_T, xref)]$, the payer looks up the first k of its stored coins $z_1, \ldots, z_k$. If not enough coins are in the sender’s storage, it outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ and sends $m'_1 := (\mathsf{SV}, tid)$. Else, it sends a message $m_1 := (\mathsf{SV}, n_1, \ldots, n_k, tid, xref, \mathsf{T}, t_T)$ to the recipient. The recipient then sends $m_2 := \text{sign}_{\mathsf{R}}(m_1)$. If S does not obtain a correct m_2, it executes the sub-protocol “refresh“ and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$.

Else, the sender sends $m_3 := (\text{sign}_{sk_1}(z_1, m_2), \ldots, \text{sign}_{sk_n}(z_n, m_2))$ and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$.

Upon receipt of the coin numbers m_1 and a corresponding input $[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, xref)]$, the recipient sends m_2. If it receives m'_1, it outputs $[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$. Upon receipt of m_3, the recipient forwards m_2, m_3 to the bank who then verifies that each element i has the form $\text{sign}_{sk_i}(z_i, m_2)$, and that $pk_B(z_i) = n_i$. Furthermore, the bank marks n_i as being used and credits the amount to the account of the payee R. Then, it sends a receipt $m_4 := \text{sign}_{\mathsf{B}}(\mathsf{R}, \$k, xref)$ to the recipient who outputs $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}: \$k, xref]$ after storing the receipt.

If some coins have been deposited before, the deposit is rejected and the recipient makes no output.

Payment Recovery (Protocol “refresh“*):* This protocol is used to refresh coins where the coin-numbers have already been revealed. This guarantees unlinkability of the failed payment with future payments, i.e., without unlinkability, no such protocol would be needed. If m_2 was not received, the payer S refreshes its coins by sending fresh coin numbers $(x'_1, \ldots, x'_k)$ that are computed as above together with m_3 to the bank who issues k fresh coins while verifying the deposited coins (in case of double-deposit, it does not issue new coins). The payer then computes fresh coins z'_i and stores them.

Sender Verification (Protocol "verifySV"*):* On input of $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, \$k, \mathit{xref}_T)]$ and $[\mathsf{S}^{in}_{tid}|\mathsf{showSV}(\mathit{xref})]$, the sender sends m_2, m_3 via the third party to the bank. The bank verifies and deposits the coins, if they have not been deposited under another xref before. Then, it sends a receipt $m_5 := \mathrm{sign}_{\mathsf{B}}(\mathsf{R}, \$k, \mathit{xref})$ to the third party and to the recipient who output $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$ and $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon \$k, \mathit{xref}]$, respectively. If any of the coins have been deposited before under another xref, the bank sends $\mathrm{sign}_{\mathsf{B}}(\mathsf{double\_d}, \mathit{xref})$ to the third party who outputs $[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$[17].

Using a Payment (Protocol "$\mathsf{use_i}$"*):* On input $[\mathsf{R}^{in}_{tid}|\mathsf{use_i}(\$k)]$, the payee retrieves the receipt m_4 or m_5 and sends it as a message $(\mathrm{sign}_{\mathsf{B}}(\mathsf{R}, \$k, \mathit{xref}), \mathit{tid}, i, \mathsf{R})$ to the user U.

Upon receipt of a correct message $(\mathrm{sign}_{\mathsf{B}}(\mathsf{R}, \$k, \mathit{xref}), \mathit{tid}, i)$, the user outputs $[\mathsf{U}^{out}_{tid}|\mathsf{used_i}\colon \mathsf{R}, \$k]$.

We define $\Delta_t := 5$, $\Delta_v := 3$. ◇

Lemma 12.3
Scheme 12.3 is a transferable item with sender verifiability (Def. 12.5). □

Proof. It is clear that the scheme is optimistic and that $\delta(\sigma_S, \sigma_R, \$k) = \mathsf{true}$ holds for any state σ_R and any state σ_S of machine S where S has at least k unspent coins.

We now show that Scheme 12.3 fulfills the requirements of Definition 12.5:

Correct Execution (R. 12.5a): If $[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, \$k, \mathsf{T}, t_T, \mathit{xref})]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, \mathit{xref})]$ is input, a correct machine S outputs $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$ if it has k unspent coins left, since a correct R answers m_1 with m_2. If not enough coins are available, then S sends m'_1 and both output failed.

Sender Verification (R. 12.5b): If a correct sender output $[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$, it is able to show m_2, m_3. If we now assume that an output $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$ is not made on input $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, \$k, \mathit{xref})]$, then some of the coin numbers have been deposited before, which contradicts the assumption that S is correct and that the coin numbers were chosen randomly. Note that the coins contained in m_3 fix the xref unambiguously, i.e., an in correct R is unable to deposit the coins under another xref.

Correct Verification (R. 12.5c): If the third party outputs $[\mathsf{T}^{out}_{tid}|\mathsf{sent}]$ on input $[\mathsf{T}^{in}_{tid}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, \$k, \mathit{xref})]$, then it successfully deposited m_3 and therefore, a receipt $\mathrm{sign}_{\mathsf{B}}(\$k, \mathit{xref})$ was sent to the recipient who output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon \$k, \mathit{xref}]$

Transfer (R. 12.5d): If a correct recipient output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\colon \$k, \mathit{xref}]$, it knows a receipt $\mathrm{sign}_{\mathsf{B}}(\mathsf{R}, \$k, \mathit{xref})$. Therefore, the correct user will output $\mathsf{used_i}$ for any $i \in \mathbb{N}$.

[17] Note that in this simple example, we did not consider the security against fraud by the bank.

No Surprises (R. 12.5e): If the sender is correct and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$, it did not obtain m_2. Therefore, it does not send m_3. From the security of the payment scheme follows that the recipient does not obtain information useful for using this payment since it only obtained coin numbers but no corresponding signatures by the bank.

No Loss (R. 12.5f): f the sender is correct and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$, it did not obtain m_2. Therefore, it executes the "refresh"-protocol. This protocol will succeed since as explained for R. 12.5e, the recipient cannot deposit the payment at this point. Thus, the sender ends up with the initial number of fresh coins.

Termination (R. 12.5g): All protocols terminate within the given time.

■

Revocable payments are identical to sender-verifiable payments, except that the credit to the recipient is delayed until time $t_T + \Delta_r$.

Scheme 12.4 (On-line Payments with Revocability)
Let $\text{sign}_{\mathsf{N}}(msg)$ be the secure digital signature of a machine $\mathsf{N} \in id\_space$ under $msg \in Msg$. Let (pk_B, sk_B) be the public and private key of an RSA scheme, i.e., $pk_B(sk_B(msg)) = sk_B(pk_B(msg)) = m$ and $sk_B(a \cdot b) = sk_B(a) \cdot pk_B(b)$ for all messages $m \in Msg$ for a given set Msg.

We define a scheme $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, XREFs, TIDs, \delta_R, (\Delta_t, \Delta_r))$ with $B = (\langle \mathsf{U}, \mathsf{B} \rangle, D, TIDs)$ for on-line coins with revocability by introducing the following modified protocols:

Withdrawing Coins (Protocol "withdraw"*):* As in Scheme 12.3.

Sending a Payment (Protocol "transferR"*):* On input $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, \$k, \mathsf{T}, t_T, xref)]$, the payer looks up the first k of its stored coins $z_1, \ldots, z_k$. If not enough coins are in the sender's storage, it outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$ and sends $m_1' := (\mathsf{R}, tid)$. Else, it sends a message $m_1 := (\mathsf{R}, n_1, \ldots, n_k, tid, xref, \mathsf{T}, t_T)$ to the recipient. The recipient then sends $m_2 := \text{sign}_{\mathsf{R}}(m_1)$. If S does not obtain a correct m_2, it executes the sub-protocol "refresh" and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$.

Else, the sender sends $m_3 := (\text{sign}_{sk_1}(z_1, m_2), \ldots, \text{sign}_{sk_n}(z_n, m_2))$ and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{sentR}]$.

Upon receipt of the coin numbers m_1 and a matching input $[\mathsf{R}^{in}_{tid}|\mathsf{receiveR}(\mathsf{T}, t_T, xref)]$, the recipient sends m_2. If it receives m_1', it outputs $[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$. Upon receipt of m_3, the recipient forwards m_3 to the bank who then verifies that each element i has the form $\text{sign}_{sk_i}(z_i, m_2)$, that $pk_B(z_i) = n_i$. Furthermore, the bank marks n_i as being used and schedules the amount of $\$k$ to be credited to the account of payee R at time $t_T + \Delta_r$[18]. Then, it sends a receipt $m_4 := \text{sign}_{\mathsf{B}}(\mathsf{R}, \$k, t_T + \Delta_r, xref)$ to the recipient who outputs $[\mathsf{R}^{out}_{tid}|\mathsf{receivedR}\text{: } \$k]$ after storing the receipt.

If some coins have been deposited before, the deposit is rejected and the recipient makes no output.

[18] If one assumes that debiting the account is always possible, one may also credit them immediately and debit the amount on revocation.

Revocation (Protocol “revoke“*):* On input $[\mathsf{S}^{in}_{tid}|\mathsf{revoke}(xref)]$ by S and $[\mathsf{T}^{in}_{tid}|\mathsf{revoke}(\mathsf{S},\mathsf{R},\$k,xref)]$ by T, the sender sends m_2, m_3 to the third party. The third party checks the messages (e.g., whether they have been sent before time t_T) and forwards them to the bank if they are correct. The bank verifies the coins and makes sure that all coins have been deposited for this $xref$. If some coins were deposited under another $xref$’, then the bank sends $\mathrm{sign}_{\mathsf{B}}(\mathsf{double\_d}, xref)$ to the third party who outputs $[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$.

Then, the bank cancels the credit of $\$k$ to the recipient’s account and issues $\$k$ fresh coins to the payer using the “refresh“-protocol. Furthermore, it sends a message $\mathrm{sign}_{\mathsf{B}}(\mathsf{revoked},\mathsf{S},\mathsf{R},xref)$ to the third party, who forwards this message to S and R and stores this $xref$ in its revocation list for R.

Finally, the parties output $[\mathsf{S}^{out}_{tid}|\mathsf{revoked}]$, $[\mathsf{R}^{out}_{tid}|\mathsf{revoked}\text{: } xref]$ and $[\mathsf{T}^{out}_{tid}|\mathsf{revoked}]$.

Using a Payment (Protocol “$\mathsf{use_i}$“*):* On input $[\mathsf{R}^{in}_{tid}|\mathsf{use_i}(\$k)]$, the recipient retrieves the receipt for the payment and sends it together with tid, i, R to U.

Upon receipt of a correct message $(\mathrm{sign}_{\mathsf{B}}(\mathsf{R},\$k,t,xref), tid, i, \mathsf{R})$, after time t, the user asks the third party whether the transaction with this $xref$ has been revoked for R. If not, it outputs $[\mathsf{U}^{out}_{tid}|\mathsf{used_i}\text{: } \mathsf{R},\$k]$.

We define $\Delta_t := 5$, $\Delta_r := 4$. ◇

Lemma 12.4
Scheme 12.4 is a transferable item with revocability (Def. 12.6). □

Proof. It is clear that the scheme is optimistic and that $\delta(\sigma_S, \sigma_R, \$k) = \mathsf{true}$ for any state σ_R and any state σ_S of machine S where S has at least k unspent coins.

We now show that Scheme 12.3 fulfills the requirements of Definition 12.5:

Correct Execution (R. 12.6a): The inputs $[\mathsf{S}^{in}_{tid}|\mathsf{transferR}(\mathsf{R},\$k,\mathsf{T},t_T,xref)]$ and $[\mathsf{R}^{in}_{tid}|\mathsf{receiveR}(\mathsf{T},t_T,xref)]$ lead to an output $[\mathsf{S}^{out}_{tid}|\mathsf{sentR}]$ by a correct S if S has k unspent coins left, since a correct R answers m_1 with m_2. If not enough coins are available, then S sends m_1' and both output failed.

Correct Revocation (R. 12.6b): If a correct sender output $[\mathsf{S}^{out}_{tid}|\mathsf{sentR}]$, it is able to show m_2, m_3. If we now assume that an output $[\mathsf{T}^{out}_{tid}|\mathsf{revoked}]$ is not made on input $[\mathsf{T}^{in}_{tid}|\mathsf{revoke}(\mathsf{S},\mathsf{R},\$k,xref)]$ before time t_T, then some of the coin numbers have been deposited before for this $xref$ and R, which contradicts the assumption that S is correct and that the coin numbers were chosen randomly and then fixed in m_3 using the signatures of the keys included into the coins.

Transfer (R. 12.6c): If a correct recipient output $[\mathsf{R}^{out}_{tid}|\mathsf{receivedR}\text{: } \$k]$ on input $[\mathsf{R}^{in}_{tid}|\mathsf{receiveR}(\mathsf{T},t_T,xref)]$, then it received a receipt $m_4 = \mathrm{sign}_{\mathsf{B}}(\mathsf{R},\$k,t_T+\Delta_r,xref)$. If the run for this $xref$ and R has not been revoked (and thus $[\mathsf{R}^{out}_{tid}|\mathsf{revoked}\text{: } xref]$ was output), this receipt is sufficient for using the payment.

No Surprises (R. 12.6d): If the sender is correct and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$, it did not obtain m_2. Therefore, it does not send m_3. From the security of the payment scheme follows that the recipient does not obtain information useful for using this payment.

If the sender outputs $[\mathsf{S}^{out}_{tid}|\mathsf{sentR}]$ and the third party outputs $[\mathsf{T}^{out}_{tid}|\mathsf{revoked}]$, then the payment for this $(\mathit{xref}, \mathsf{R})$ is revoked[19] and cannot be used anymore.

No Loss (R. 12.6e): If the sender is correct and outputs $[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$, it executes "refresh" and ends up with the same number of coins that can be spent (see R. 12.5f of Scheme 12.3).

If the third party input revoke, the bank issues fresh coins for this particular sender.

Termination (R. 12.6f): All protocols terminate within the given time.

■

12.3.3 Labeled Data

For some data to be exchanged, such as the message in a labeled certified mail scheme (see Section 4), the recipient knows the intended sender and a label of the message but has no expectation on the contents of the message at all. Since this special case enables efficient generatability (the generated item is defined to be the expected one), we now describe this kind of generatable transfer for labeled data.

Scheme 12.5 (Secret Generatable Data)
Let $\mathrm{sign}_{\mathsf{N}}(msg)$ be the secure digital signature of a machine $\mathsf{N} \in \mathit{id\_space}$ under $msg \in Msg$. Let $E_T(r; msg)$ be a probabilistic encryption of msg randomized with r under T's public key. Let L be a set of message labels.

We define a scheme $I = (\mathsf{S}, \mathsf{R}, \mathsf{T}, B, \mathit{XREFs}, \mathit{TIDs}, \delta_G, (\Delta_p, \Delta_t, \Delta_g))$ for generatable data with $B = (\mathsf{U}, D, \mathit{TIDs})$ with $D := \mathit{id\_space} \times \mathit{id\_space} \times L$ and $\mathit{TIDs} := \{0,1\}^*$ as follows:

Creating Secret Data (Protocol "create"*):* This protocol stores the data to be sent. On input of $[\mathsf{S}^{in}_{tid}|\mathsf{create}(\mathsf{R}, l, m)]$, the data $m \in \{0,1\}^n$ is stored under the intended recipient R and a label l.

Preparing Generation (Protocol "prepareG"*)* Upon input of $[\mathsf{S}^{in}_{tid}|\mathsf{prepareG}(\mathsf{R}, (\mathsf{S}, \mathsf{R}, l), \mathsf{T}, t_T, \mathit{xref})]$ with $l \in L$, machine S retrieves m for the given (R, l) and selects a random number r and sends $m_1 := \mathrm{sign}_{\mathsf{S}}(E_T(r; m, \mathsf{S}, \mathsf{R}, \mathit{xref}), (\mathsf{S}, \mathsf{R}, l), \mathit{tid}, \mathsf{T}, t_T, \mathit{xref})$ to R. Furthermore, S stores (m, r, R) under xref.

On input of $[\mathsf{R}^{in}_{tid}|\mathsf{checkG}(\mathsf{T}, t_T, \mathit{xref})]$, the recipient waits for m_1 and verifies the signature and the clear-text portion of the message. If it matches the input parameters, it outputs $[\mathsf{R}^{out}_{tid}|\mathsf{providesG}\text{: } (\mathsf{S}, \mathsf{R}, l)]$ and stores m_1 under xref. Else, it ignores m_1.

[19]Since we modeled the "use" using receipts, this is independent of the actual revocation of the payment. If the payment would not be revoked together with the receipt, the scheme may still fulfill the requirements even though it is intuitively insecure.

Sending Data (Protocol “transfer“*):* On input of $[\mathsf{S}_{tid}^{in}|\mathsf{transferG}(xref)]$ to a sender S, it looks up (m, r, R) for the input $xref$. If no such entry exists, it does nothing. Else, it sends a message $(m, r, tid, xref)$ to the intended recipient R and deletes the entry. Upon receipt, the recipient verifies that $E_T(r; m, \mathsf{S}, \mathsf{R}, xref)$ in fact results in the ciphertext contained in m_1 and outputs $[\mathsf{R}_{tid}^{out}|\mathsf{received}\colon xref]$ if this succeeds.

Generating the Data (Protocol “generate“*):* On input $[\mathsf{R}_{tid}^{in}|\mathsf{generate}(xref)]$ and $[\mathsf{T}_{tid}^{in}|\mathsf{generate}(\mathsf{S}, \mathsf{R}, (\mathsf{S}, \mathsf{R}, l), xref)]$, R sends $m_1, tid, xref$ to T who decrypts it and verifies that it has the correct form and that it was sent before time t_T. If this verification fails, it outputs $[\mathsf{T}_{tid}^{out}|\mathsf{failed}]$. Else, it sends $(m, r, tid, xref)$ to R and S. Then, the machines output $[\mathsf{R}_{tid}^{out}|\mathsf{received}\colon xref]$, $[\mathsf{T}_{tid}^{out}|\mathsf{generated}]$, and $[\mathsf{S}_{tid}^{out}|\mathsf{generated}\colon xref]$.

If the verification of the encrypted message fails, the third party defines $m := \epsilon$ as the received message.

Outputting the Received Data (Protocol “retrieve“*):* On input of $[\mathsf{R}_{tid}^{in}|\mathsf{retrieve}(xref)]$ after receiving the data in a message $(m, r, tid, xref)$ from S or T under the given $xref$, the recipient outputs $[\mathsf{R}_{tid}^{out}|\mathsf{data}\colon m]$.

Using Data (Protocol “use$_\mathsf{i}$“*):* On input $[\mathsf{R}_{tid}^{in}|\mathsf{use_i}((\mathsf{S}, \mathsf{R}, l))]$, the recipient sends (tid, m, r, i, m_1) to the user.

The user U verifies that m_1 is correct and that $E_T(r; m, \mathsf{S}, \mathsf{R}, xref)$ results in the ciphertext contained in m_1. If this is the case, it outputs $[\mathsf{U}_{tid}^{out}|\mathsf{used_i}\colon \mathsf{R}, (\mathsf{S}, \mathsf{R}, l)]$. Else, it asks T to decrypt m_1 and checks whether m and m' have at least i digits in common. If this is the case, it outputs $[\mathsf{U}_{tid}^{out}|\mathsf{used_i}\colon \mathsf{R}, (\mathsf{S}, \mathsf{R}, l)]$ as well. In all other cases, it does not make any output.

We define $\Delta_p := 1$, $\Delta_t := 1$, and $\Delta_g := 2$. ◇

Remark 12.12. This transferable item together with transferable signatures providing sender verifiability (Scheme 12.1) can be used to instantiate a labeled certified mail scheme using the generic fair exchange protocol described in Section 13.1. ○

Lemma 12.5
Scheme 12.5 is a transferable item with generatability (Def. 12.8). □

Proof. It is clear that the scheme is optimistic and that $\delta(\sigma_S, \sigma_R, (\mathsf{S}, \mathsf{R}, l)) = \mathsf{true}$ for any state σ_S of machine S and any state σ_R iff S stored a message m for this R and l.

We now show that Scheme 12.5 fulfills the requirements of Definition 12.8:

Correct Execution (R. 12.8a): If the recipient output $[\mathsf{R}_{tid}^{out}|\mathsf{providesG}\colon (\mathsf{S}, \mathsf{R}, l), \mathsf{T}, t_T, xref]$, the recipient stored an m_1, i.e., a correct sender stored (m, r, R) under $xref$. Therefore, an input $[\mathsf{S}_{tid}^{in}|\mathsf{transferG}(xref)]$ will lead to an output $[\mathsf{R}_{tid}^{out}|\mathsf{received}\colon xref]$.

Correct Generation (R. 12.8b): If a correct recipient output $[\mathsf{R}_{tid}^{out}|\mathsf{providesG}\colon (\mathsf{S}, \mathsf{R}, l), \mathsf{T}, t_T, xref]$ it knows a corresponding m_1. After sending this message to T before time t_T, T will decrypt it and send m, r to S and R. Therefore, they will output $[\mathsf{R}_{tid}^{out}|\mathsf{received}\colon xref]$, $[\mathsf{S}_{tid}^{out}|\mathsf{generated}\colon xref]$ and T outputs $[\mathsf{T}_{tid}^{out}|\mathsf{generated}]$.

Transfer (R. 12.8c): If a recipient output $[\mathsf{R}_{tid}^{out}|\mathsf{received}\colon xref]$ after an output $[\mathsf{R}_{tid}^{out}|\mathsf{providesG}\colon (\mathsf{S},\mathsf{R},l),\mathsf{T},t_T,xref]$, it knows m_1, m, r, which are sufficient for ownership.

No Surprises (R. 12.8d): If $[\mathsf{S}_{tid}^{in}|\mathsf{prepareG}(\mathsf{R},(\mathsf{S},\mathsf{R},msg),\mathsf{T},t_T,xref)]$ is input by the sender and neither $[\mathsf{S}_{tid}^{in}|\mathsf{transferG}(xref)]$ nor $[\mathsf{T}_{tid}^{in}|\mathsf{generate}(\mathsf{S},\mathsf{R},(\mathsf{S},\mathsf{R},l),xref)]$, then the recipient obtains neither m nor r. From the security of the encryption scheme follows that message m_1 obtained by R does not contain knowledge useful for using $(\mathsf{S},\mathsf{R},l)$, since it does not help to guess a correct m fixed by m_1. Furthermore, if a different S', R', or $xref'$ is input, the encrypted ciphertext does not match with the parameters input by T.

No Loss (R. 12.8e): This requirement holds since S only deletes the stored data on input $[\mathsf{S}_{tid}^{in}|\mathsf{transferG}(xref)]$.

Termination (R. 12.8f): All protocols have a fixed run-time.

■

Chapter 13

Protocols for Transfer-based Fair Exchange

In Sections 13.1 and 13.2, we describe two examples of synchronous fair exchange protocols based on exchange-enabling transfers. These fair exchange protocols follow the same pattern:

1. The parties agree on the parameters of the exchange.
2. The parties prepare the second transfer.
3. The parties execute both transfers.
4. If something goes wrong, the parties involve the third party in order to restore fairness using the exchange-enabling properties of the transfers.

If the players are correct and agree, the protocol usually ends after Step 3. Therefore, the protocols are optimistic if the underlying transfers are optimistic.

The protocol described in Section 13.1 requires transferable items with sender verifiability and generatability, respectively. The protocol described in Section 13.2 requires recipient verifiability and revocability, respectively.

In order to use these two protocols to exchange any two items, we describe in Section 13.3 how to use certain exchange-enabling properties to simulate others. Together with the two given protocols for fair exchange, this enables us to exchange any two items providing exchange-enabling properties if at least one item provides generatability or revocability.

Note that these simulations are mainly for completeness. In practice, one would provide additional fair exchange protocols, which are optimized for any two particular exchange-enabling transfers that can be exchanged at all.

13.1 Fair Exchange of Sender-verifiable and Generatable Items

The following synchronous optimistic fair exchange protocol assumes that one of the items offers sender verifiability whereas the other item offers generatability. The basic idea is that the participants prepare the generatable item sent

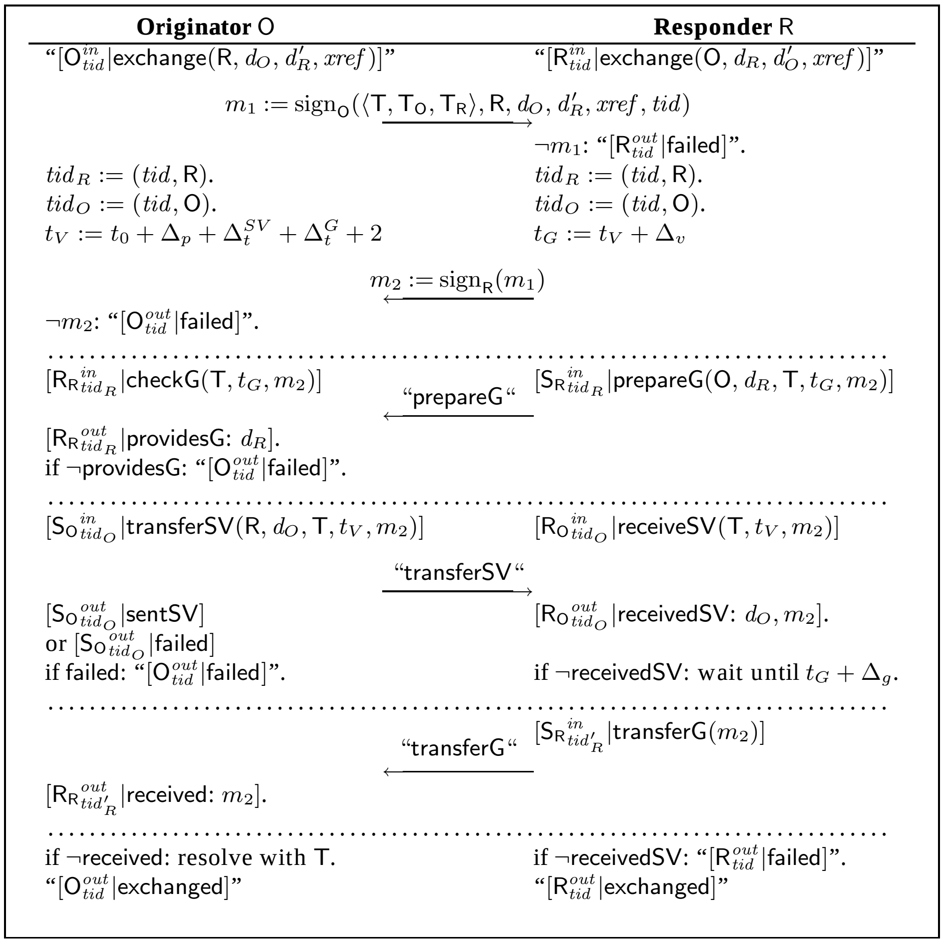

Originator O
Responder R
"$[\mathsf{O}^{in}_{tid}|\mathsf{exchange}(\mathsf{R}, d_O, d'_R, xref)]$"
"$[\mathsf{R}^{in}_{tid}|\mathsf{exchange}(\mathsf{O}, d_R, d'_O, xref)]$"
$m_1 := \mathrm{sign}_\mathsf{O}(\langle \mathsf{T}, \mathsf{T_O}, \mathsf{T_R}\rangle, \mathsf{R}, d_O, d'_R, xref, tid)$
$\neg m_1$: "$[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$".
$tid_R := (tid, \mathsf{R})$.
$tid_O := (tid, \mathsf{O})$.
$t_V := t_0 + \Delta_p + \Delta^{SV}_t + \Delta^G_t + 2$
$tid_R := (tid, \mathsf{R})$.
$tid_O := (tid, \mathsf{O})$.
$t_G := t_V + \Delta_v$
$m_2 := \mathrm{sign}_\mathsf{R}(m_1)$
$\neg m_2$: "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$".
$[\mathsf{R_R}{}^{in}_{tid_R}|\mathsf{checkG}(\mathsf{T}, t_G, m_2)]$
"prepareG"
$[\mathsf{S_R}{}^{in}_{tid_R}|\mathsf{prepareG}(\mathsf{O}, d_R, \mathsf{T}, t_G, m_2)]$
$[\mathsf{R_R}{}^{out}_{tid_R}|\mathsf{providesG}\text{: } d_R]$.
if $\neg$providesG: "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$".
$[\mathsf{S_O}{}^{in}_{tid_O}|\mathsf{transferSV}(\mathsf{R}, d_O, \mathsf{T}, t_V, m_2)]$
$[\mathsf{R_O}{}^{in}_{tid_O}|\mathsf{receiveSV}(\mathsf{T}, t_V, m_2)]$
"transferSV"
$[\mathsf{S_O}{}^{out}_{tid_O}|\mathsf{sentSV}]$
or $[\mathsf{S_O}{}^{out}_{tid_O}|\mathsf{failed}]$
if failed: "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$".
$[\mathsf{R_O}{}^{out}_{tid_O}|\mathsf{receivedSV}\text{: } d_O, m_2]$.
if $\neg$receivedSV: wait until $t_G + \Delta_g$.
$[\mathsf{S_R}{}^{in}_{tid'_R}|\mathsf{transferG}(m_2)]$
"transferG"
$[\mathsf{R_R}{}^{out}_{tid'_R}|\mathsf{received}\text{: } m_2]$.
if $\neg$received: resolve with T.
"$[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$"
if $\neg$receivedSV: "$[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$".
"$[\mathsf{R}^{out}_{tid}|\mathsf{exchanged}]$"

Figure 13.1: Optimistic Fair Exchange Protocol of Scheme 13.1 for Exchanging the Generatable Item I_R and the Sender-verifiable Item I_O sent by O.

by the responder, transfer the sender-verifiable item from the originator to the responder, and finally transfer the generatable item from the responder to the originator. If this transfer fails, the third party is asked to generate it after verifying that the sender-verifiable item was in fact transferred successfully. The behavior of the protocol in the fault-less case is sketched in Figure 13.1, the recovery is depicted in Figure 13.2.

Scheme 13.1 (Exchange "SV & G")
Let Σ_O be the set of all transferable items with sender verifiability and let Σ_R be the set of all transferable items with generatability.

A transfer-based fair exchange scheme $(\mathsf{O}, \mathsf{R}, \mathsf{T}, XREFs, TIDs)$ for Σ_O and Σ_R is defined as follows:

Let $I_O = (\mathsf{S_O}, \mathsf{R_O}, \mathsf{T_O}, B_O, \{0,1\}^*, TIDs, \delta_{SV}, (\Delta^{SV}_t, \Delta_v))$ with $I_O \in \Sigma_O$ be a sender-verifiable item. Let $I_R = (\mathsf{S_R}, \mathsf{R_R}, \mathsf{T_R}, B_R, \{0,1\}^*, TIDs, \delta_G, (\Delta_p, \Delta^G_t, \Delta_g))$ with $I_R \in \Sigma_R$ be a generatable item.[1] Then, the behavior of O, R, and T is defined as follows:

[1]For simplicity, in the sequel we assume that all schemes use the same set $TIDs$ and $XREFs$

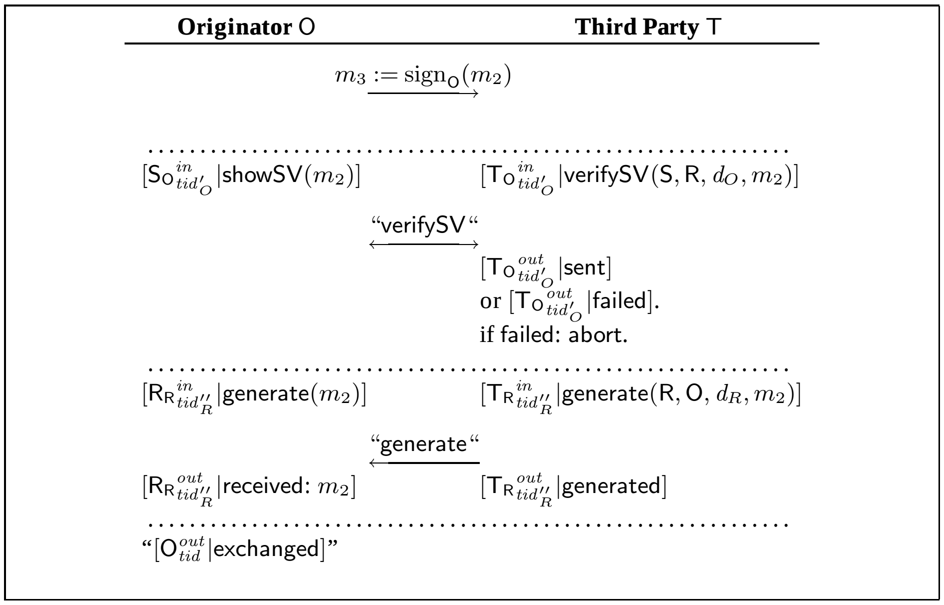

Originator O
Third Party T
$m_3 := \text{sign}_\mathsf{O}(m_2)$
$[\mathsf{S_O}{}^{in}_{tid'_O}|\mathsf{showSV}(m_2)]$
$[\mathsf{T_O}{}^{in}_{tid'_O}|\mathsf{verifySV}(\mathsf{S},\mathsf{R},d_O,m_2)]$
"verifySV"
$[\mathsf{T_O}{}^{out}_{tid'_O}|\mathsf{sent}]$
or $[\mathsf{T_O}{}^{out}_{tid'_O}|\mathsf{failed}]$.
if failed: abort.
$[\mathsf{R_R}{}^{in}_{tid''_R}|\mathsf{generate}(m_2)]$
$[\mathsf{T_R}{}^{in}_{tid''_R}|\mathsf{generate}(\mathsf{R},\mathsf{O},d_R,m_2)]$
"generate"
$[\mathsf{R_R}{}^{out}_{tid''_R}|\mathsf{received}\text{: } m_2]$
$[\mathsf{T_R}{}^{out}_{tid''_R}|\mathsf{generated}]$
"$[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$"

Figure 13.2: Recovery Protocol of Scheme 13.1.

Exchange by O*:* On input "$[\mathsf{O}^{in}_{tid}|\mathsf{exchange}(\mathsf{R}, d_O, d'_R, xref)]$" in round t_0, the machine sends m_1:=$\text{sign}_\mathsf{O}(\langle \mathsf{T}, \mathsf{T_O}, \mathsf{T_R}\rangle, \mathsf{R}, d_O, d'_R, xref, tid)$[2] and waits for m_2. If m_2 is not received, it outputs $[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$. Else, it executes the following sub-protocols:

Sub-protocol "prepareG": It inputs $[\mathsf{R_R}{}^{in}_{tid_R}|\mathsf{checkG}(\mathsf{T}, t_G, m_2)]$ with $t_G := t_V + \Delta_v$ and $t_V := t_0 + \Delta_p + \Delta_t^{SV} + \Delta_t^G + 2$. Then, it waits for an output $[\mathsf{R_R}{}^{out}_{tid_R}|\mathsf{providesG}\text{: } d_R]$ at time $t_0 + \Delta_p + 2$. If this is not output after time Δ_p rounds, it outputs "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$".

Sub-protocol "transferSV": Else, it inputs $[\mathsf{S_O}{}^{in}_{tid_O}|\mathsf{transferSV}(\mathsf{R}, d_O, \mathsf{T}, t_V, m_2)]$ and waits for an output $[\mathsf{S_O}{}^{out}_{tid_O}|\mathsf{sentSV}]$ at time $t_0 + 2 + \Delta_p + \Delta_t^{SV}$. If $[\mathsf{S_O}{}^{out}_{tid_O}|\mathsf{failed}]$ is output instead, it outputs "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$".

Sub-protocol "transferG": If it receives an output $[\mathsf{R_R}{}^{out}_{tid'_R}|\mathsf{received}\text{: } m_2]$[3] at time $t_V - 2$ it outputs "$[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$". Else, it starts recovery with T.

Exchange by R*:* On input "$[\mathsf{R}^{in}_{tid}|\mathsf{exchange}(\mathsf{O}, d_R, d'_O, xref)]$" in round t_0, machine R waits for m_1 and verifies it using the input parameters, i.e., $d_R = d'_R$, $d_O = d'_O$. If it did not receive a correct m_1[4], it outputs "$[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$". Else, it sends $m_2 := \text{sign}_\mathsf{R}(m_1)$ and then executes the following sub-protocols:

Sub-protocol "prepareG": It inputs $[\mathsf{S_R}{}^{in}_{tid_R}|\mathsf{prepareG}(\mathsf{O}, d_R, \mathsf{T}, t_G, m_2)]$ with $tid_R = (tid, \mathsf{R})$ for $t_G := t_V + \Delta_v$, $t_V := t_0 + \Delta_p + \Delta_t^{SV} + \Delta_t^G + 3$,

and that these sets are large enough to allow for our constructions.

[2]Remember that m_1 contains the starting time t_0.

[3]As described in Section 11.3, we assume that primed tids are unique extensions of the underlying tid.

[4]If different $xref$s or tids are input, no correct m_1 will be received.

and $m_2 := \mathrm{sign}_{\mathsf{R}}(m_1)$ to the sender $\mathsf{S_R}$ of the item to be sent with generatability.

Sub-protocol “transferSV“: It inputs $[\mathsf{R_O}{}^{out}_{tid_O}|\text{receiveSV: }\mathsf{T}, t_V, m_2]$ with $t_V := t_0 + \Delta_p + \Delta_t^{SV} + \Delta_t^G + 2$ and waits for an output $[\mathsf{R_O}{}^{out}_{tid_O}|$ receivedSV: $d_O, m_2]$ until time $t_G + \Delta_g$[5]. If this output is not received, it outputs “$[\mathsf{R}^{out}_{tid}|$failed$]$”.

Sub-protocol “transferG“: If the output receivedSV was obtained at time $t_0 + \Delta_p + \Delta_t^{SV} + 2$, it inputs $[\mathsf{S_R}{}^{in}_{tid'_R}|\text{transferG}(m_2)]$. If the output transferSV was obtained before time t_G, it outputs “$[\mathsf{R}^{out}_{tid}|$ exchanged$]$”.

Recovery of O: This recovery protocol is executed to generate an item if the originator claims that it sent its item d_O but did not receive the expected transfer d_R.

Machine O sends a signed request $m_3 := \mathrm{sign}_{\mathsf{O}}(m_2)$ to start recovery.

Then, the “verifySV“-protocol is started with the inputs $[\mathsf{S_O}{}^{in}_{tid'_O}|\text{showSV}(m_2)]$ and $[\mathsf{T_O}{}^{in}_{tid'_O}|\text{verifySV}(\mathsf{S}, \mathsf{R}, d_O, m_2)]$ with the R and d_O signed in m_2. If this sub-protocol outputs $[\mathsf{T_O}{}^{out}_{tid'_O}|\text{failed}]$, the third party aborts[6]. Else, if the verification outputs $[\mathsf{T_O}{}^{out}_{tid'_O}|\text{sent}]$, the item is generated:

The players T and O input $[\mathsf{T_R}{}^{in}_{tid''_R}|\text{generate}(\mathsf{R}, \mathsf{O}, d_R, m_2)]$ and $[\mathsf{R_R}{}^{in}_{tid''_R}|$ generate$(m_2)]$. If the generation outputs $[\mathsf{R_R}{}^{out}_{tid''_R}|\text{received: } m_2]$ before time $t_G + \Delta_g$, machine O outputs “$[\mathsf{O}^{out}_{tid}|$exchanged$]$” and else “$[\mathsf{O}^{out}_{tid}|$failed$]$”.

◇

Theorem 13.1 (Security of Scheme 13.1)
Scheme 13.1 is a secure transfer-based fair exchange protocol for sender-verifiable and generatable items. □

Proof. We show that all requirements of Definitions 12.2 and 12.4 are fulfilled by Scheme 13.1 if I_O is sender verifiable and I_R is generatable:

Correct Execution (R. 12.2a) and Availability (R. 12.4b): If both machines are correct and receive non-matching inputs, R outputs failed after receiving a message m_1 with unexpected parameters, whereas O outputs failed after not receiving m_2. Else, they start the “prepareG“-protocol with matching inputs.

If both machines O and R are able to transfer the items to each other, the “prepareG“-protocol outputs $[\mathsf{R_R}{}^{out}_{tid_R}|\text{providesG: } d_O]$ on input $[\mathsf{R_R}{}^{out}_{tid_R}|$ checkG: $\mathsf{T}, t_G, m_2]$ (by definition of δ_G and $t_G \geq t_0 + 2 + \Delta_p$). Then, the correct player input $[\mathsf{S_O}{}^{in}_{tid_O}|\text{transferSV}(\mathsf{R}, d_O, \mathsf{T}, t_V, m_2)]$ and $[\mathsf{R_O}{}^{in}_{tid_O}|$ receiveSV$(\mathsf{T}, t_V, m_2)]$ with $t_V \geq t_0 + 2 + \Delta_p + \Delta_t^{SV}$. If O is able to transfer d_O, this leads to an output $[\mathsf{R_O}{}^{out}_{tid_O}|\text{receivedSV: } d_O, m_2]$ and an output $[\mathsf{S_O}{}^{out}_{tid_O}|\text{sentSV}]$ [R. 12.5a]. Therefore, R will input $[\mathsf{S_R}{}^{in}_{tid'_R}|\text{transferG}($

[5]Note that the outputs of this sub-protocol may in fact be caused by the recovery during “verifySV“, e.g., if a message is sent again.

[6]In this case, the originator was incorrect since it was unable to convince T even though it obtained an output sentSV.

$m_2)$] and output "[R^{out}_{tid}|exchanged]". As a consequence, a correct O will obtain an output [$\mathsf{R_R}{}^{out}_{tid'_R}$|received: m_2] [R. 12.8a] and will output "[O^{out}_{tid}| exchanged]" as well.

Transfer (R. 12.2b): A participant, say O, only outputs "[O^{out}_{tid}|exchanged]" after receiving the expected transfer. From [R. 12.5d] or [R. 12.8c] follows that it owns the item, since no later state changes are made.

No Surprises (R. 12.2c) If participant O outputs "[O^{out}_{tid}|failed]", it either failed before inputting [$\mathsf{S_O}{}^{in}_{tid_O}$|transferSV(R, d_O, T, t_V, m_2)], the sub-protocol "transferSV" output [$\mathsf{S_O}{}^{out}_{tid_O}$|failed], or the recovery output [$\mathsf{T_O}{}^{out}_{tid'_O}$|failed] to the third party. In the first two cases, the recipient does not obtain knowledge on item d_O [R. 12.5e]. If the third party outputs [$\mathsf{T_O}{}^{out}_{tid'_O}$|failed], then the originator started recovery by sending m_3, i.e., it obtained an output [$\mathsf{S_O}{}^{out}_{tid_O}$|sentSV]. However, for a correct originator, [R. 12.5b] implies that this results in an output [$\mathsf{T_O}{}^{out}_{tid'_O}$|sent] at T.

If participant R outputs [R^{out}_{tid}|failed] after not receiving m_1, it does not input [$\mathsf{S_R}{}^{in}_{tid_R}$|prepareG(...)] and therefore, R is still able to transfer the item [R. 12.8d]. If it outputs "[R^{out}_{tid}|failed]" upon not receiving [$\mathsf{R_O}{}^{out}_{tid}$| receivedSV: d_O, m_2] until time $t_V + \Delta_v$, [R. 12.5c] implies that in this case, the verification at the third party will output [$\mathsf{T_R}{}^{out}_{tid''_R}$|failed] and the third party will not input [$\mathsf{T_R}{}^{in}_{tid''_R}$|generate(R, O, d_R, m_2)]. Since R does not input [$\mathsf{S_R}{}^{in}_{tid'_R}$|transferG()] either, the recipient does not obtain knowledge about item d_R [R. 12.8d].

No Loss (R. 12.4c): Similar to "no surprises" using [R. 12.8e, R. 12.5f], except that R (the sender of the generatable item) is re-enabled to transfer the item after time $t_G + \Delta_g$.

Termination (R. 12.2d): Since all sub-protocols terminate [R. 12.5g,R. 12.8f] in a fixed time, the protocols terminate in a fixed time as well.

■

13.2 Fair Exchange of Revocable and Recipient-verifiable Items

The following exchange protocol requires that one of the items offers revocability whereas the other item offers recipient verifiability. The basic idea is that the participants first prepare the recipient-verifiable item. Then, they transfer the revocable item and the recipient-verifiable item. If the recipient-verifiable item is not sent, the third party is invoked to verify this fact and revoke the first transfer. The behavior of the protocol in the fault-less case is sketched in Figure 13.3, the recovery is depicted in Figure 13.4.

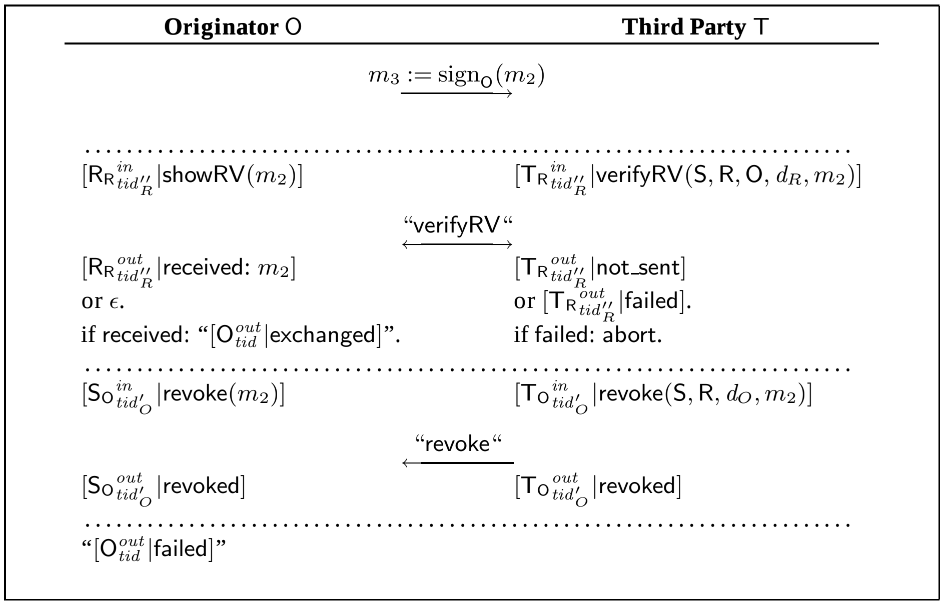
Scheme 13.2 (Exchange "R&RV")
Let Σ_O be the set of all transferable items with revocability and let Σ_R be the set of all transferable items with recipient verifiability.

Originator O		**Responder** R
“[$\mathsf{O}^{in}_{tid}\mid$exchange($\mathsf{R}, d_O, d'_R, xref$)]”		“[$\mathsf{R}^{in}_{tid}\mid$exchange($\mathsf{O}, d_R, d'_O, xref$)]”
	$m_1 := \text{sign}_{\mathsf{O}}(\langle \mathsf{T}, \mathsf{T_O}, \mathsf{T_R}\rangle, \mathsf{R}, d_O, d'_R, xref, tid)$ $\xrightarrow{\quad}$	
		$\neg m_1$ or not ok (e.g., $d_O \neq d'_O$): “[$\mathsf{O}^{out}_{tid}\mid$failed]”.
$tid_R := (tid, \mathsf{R})$.		$tid_R := (tid, \mathsf{R})$.
$tid_O := (tid, \mathsf{O})$.		$tid_O := (tid, \mathsf{O})$.
		$m_2 := \text{sign}_{\mathsf{R}}(m_1)$.
$t_R := t_V + \Delta_v$.		$t_V := t_0 + \Delta_p + \Delta_t^R + \Delta_t^{RV} + 3$.
	$m_2 := \text{sign}_{\mathsf{R}}(m_1)$ $\xleftarrow{\quad}$	
$\neg m_2$ or not ok: “[$\mathsf{O}^{out}_{tid}\mid$failed]”.		
[$\mathsf{R_R}{}^{in}_{tid_R}\mid$checkRV($\mathsf{T}, t_V, m_2$)]	“prepareRV“ $\xleftarrow{\quad}$	[$\mathsf{S_R}{}^{in}_{tid_R}\mid$prepareRV($\mathsf{O}, d_R, \mathsf{T}, t_V, m_2$)]
[$\mathsf{R_R}{}^{out}_{tid_R}\mid$providesRV: d_R]. if $\neg$providesRV: “[$\mathsf{O}^{out}_{tid}\mid$failed]”.		
[$\mathsf{S_O}{}^{in}_{tid_O}\mid$transferR($\mathsf{R}, d_O, \mathsf{T}, t_R, m_2$)]		[$\mathsf{R_O}{}^{in}_{tid_O}\mid$receiveR($\mathsf{T}, t_R, m_2$)]
	“transferR“ $\xrightarrow{\quad}$	
[$\mathsf{S_O}{}^{out}_{tid_O}\mid$sentR] or [$\mathsf{S_O}{}^{out}_{tid_O}\mid$failed]. if failed: “[$\mathsf{O}^{out}_{tid}\mid$failed]”.		[$\mathsf{R_O}{}^{out}_{tid_O}\mid$receivedR: d_O] if $\neg$receivedR: “[$\mathsf{R}^{out}_{tid}\mid$failed]”.
	“transferRV“ $\xleftarrow{\quad}$	[$\mathsf{S_R}{}^{in}_{tid'_R}\mid$transferRV($m_2$)]
[$\mathsf{R_R}{}^{out}_{tid'_R}\mid$received: m_2].		
if $\neg$received: resolve with T.		if failed or not_sent: “[$\mathsf{R}^{out}_{tid}\mid$failed]”.
“[$\mathsf{O}^{out}_{tid}\mid$exchanged]”		“[$\mathsf{R}^{out}_{tid}\mid$exchanged]” after time t_R.

Figure 13.3: Optimistic Fair Exchange Protocol of Scheme 13.2 for Exchanging the Recipient-verifiable Item I_R sent by R and the Revocable Item I_O sent by O.

A transfer-based fair exchange scheme $(\mathsf{O}, \mathsf{R}, \mathsf{T}, XREFs, TIDs)$ for Σ_O and Σ_R is defined as follows: Let $I_O = (\mathsf{S_O}, \mathsf{R_O}, \mathsf{T_O}, B_O, \{0,1\}^*, TIDs, \delta_R, (\Delta_t^R, \Delta_r))$ with $I_O \in \Sigma_O$ be a revocable item. Let $I_R = (\mathsf{S_R}, \mathsf{R_R}, \mathsf{T_R}, B_R, \{0,1\}^*, TIDs, \delta_{RV}, (\Delta_p, \Delta_t^{RV}, \Delta_v))$ with $I_R \in \Sigma_R$ be a recipient-verifiable item. Then, the behavior of O, R, and T is defined as follows:

Exchange by O*:* On input “[$\mathsf{O}^{in}_{tid}\mid$exchange($\mathsf{R}, d_O, d'_R, xref$)]” in round t_0, the machine sends $m_1 = \text{sign}_{\mathsf{O}}(\langle \mathsf{T}, \mathsf{T_O}, \mathsf{T_R}\rangle, \mathsf{R}, d_O, d'_R, xref, tid)$ and waits for a message $m_2 = \text{sign}_{\mathsf{R}}(m_1)$. If this message is received, it inputs [$\mathsf{R_R}{}^{in}_{tid_R}\mid$ checkRV(T, t_V, m_2)] with $t_V = t_0 + \Delta_p + \Delta_t^R + \Delta_t^{RV} + 3$ and waits for an output of the sub-protocol “prepareRV“. If the sub-protocol outputs [$\mathsf{R_R}{}^{out}_{tid_R}\mid$providesRV: d_R], it inputs [$\mathsf{S_O}{}^{in}_{tid'_O}\mid$transferR($\mathsf{R}, d_O, \mathsf{T}, t_R, m_2$)] with $t_R := t_V + \Delta_v$. Else, it outputs “[$\mathsf{O}^{out}_{tid}\mid$failed]”. If “transferR“ outputs [$\mathsf{S_O}{}^{out}_{tid'_O}\mid$failed], it outputs “[$\mathsf{O}^{out}_{tid}\mid$failed]”. If it receives the outputs [$\mathsf{S_O}{}^{out}_{tid_O}\mid$



Originator O		**Third Party** T
	$\xrightarrow{m_3 := \text{sign}_{\mathsf{O}}(m_2)}$	
$[\mathsf{R_R}{}^{in}_{tid''_R}\|\mathsf{showRV}(m_2)]$		$[\mathsf{T_R}{}^{in}_{tid''_R}\|\mathsf{verifyRV}(\mathsf{S},\mathsf{R},\mathsf{O},d_R,m_2)]$
	$\xleftrightarrow{\text{“verifyRV“}}$	
$[\mathsf{R_R}{}^{out}_{tid''_R}\|\text{received: } m_2]$ or ϵ. if received: “$[\mathsf{O}^{out}_{tid}\|\text{exchanged}]$”.		$[\mathsf{T_R}{}^{out}_{tid''_R}\|\text{not\_sent}]$ or $[\mathsf{T_R}{}^{out}_{tid''_R}\|\text{failed}]$. if failed: abort.
$[\mathsf{S_O}{}^{in}_{tid'_O}\|\mathsf{revoke}(m_2)]$		$[\mathsf{T_O}{}^{in}_{tid'_O}\|\mathsf{revoke}(\mathsf{S},\mathsf{R},d_O,m_2)]$
	$\xleftarrow{\text{“revoke“}}$	
$[\mathsf{S_O}{}^{out}_{tid'_O}\|\text{revoked}]$		$[\mathsf{T_O}{}^{out}_{tid'_O}\|\text{revoked}]$
“$[\mathsf{O}^{out}_{tid}\|\text{failed}]$”		

Figure 13.4: Recovery Protocol of Scheme 13.2.

sentR] and $[\mathsf{R_R}{}^{out}_{tid'_R}|\text{received: } m_2]$ it outputs “$[\mathsf{R}^{out}_{tid}|\text{exchanged}]$”. Else, it starts recovery with T.

Exchange by R*:* On input “$[\mathsf{R}^{in}_{tid}|\mathsf{exchange}(\mathsf{O}, d_R, d'_O, xref)]$” in round t_0, the responder waits for m_1 and verifies it. If it is incorrect, it outputs “$[\mathsf{R}^{out}_{tid}|$ failed]”. Else, it computes $m_2 := \text{sign}_{\mathsf{R}}(m_1)$ and sends it to O. Then, it executes the following sub-protocols:

Sub-Protocol “prepareRV“: Machine R inputs $[\mathsf{S_R}{}^{in}_{tid_R}|\mathsf{prepareRV}(\mathsf{O}, d_R, \mathsf{T}, t_V, m_2)]$ with $t_V := t_0 + \Delta_p + \Delta_t^R + \Delta_t^{RV} + 3$ and $tid_R = (tid, \mathsf{R})$ to the sender $\mathsf{S_R}$ of the item to be sent with recipient verifiability.

Sub-Protocol “transferR“: Then, it inputs $[\mathsf{R_O}{}^{in}_{tid_O}|\mathsf{receiveR}(\mathsf{T}, t_R, m_2)]$ and waits for an output of the “transferR“-protocol. If $\mathsf{R_O}$ does not output $[\mathsf{R_O}{}^{out}_{tid_O}|\text{receivedR: } d_O]$ with $tid_O = (tid, \mathsf{O})$ at time $t_0 + \Delta_p + \Delta_t^R + 2$, the responder outputs “$[\mathsf{R}^{out}_{tid}|\text{failed}]$”.

Sub-Protocol “transferRV“: The responder inputs $[\mathsf{S_R}{}^{in}_{tid'_R}|\mathsf{transferRV}(m_2)]$ and outputs “$[\mathsf{R}^{out}_{tid}|\text{exchanged}]$” at time t_R. If the sub-protocol “transferRV“ outputs $[\mathsf{S_R}{}^{out}_{tid'_R}|\text{failed}]$ or $[\mathsf{S_R}{}^{out}_{tid''_R}|\text{not\_sent: } m_2]$, the responder outputs “$[\mathsf{R}^{out}_{tid}|\text{failed}]$”.

Recovery by O*:* This recovery protocol is executed to revoke an item if the responder did not send its item. The originator sends a signed request $m_3 := \text{sign}_{\mathsf{O}}(m_2)$ to the third party in order to prove the deal the participants made. Then, the players input $[\mathsf{R_R}{}^{in}_{tid''_R}|\mathsf{showRV}(m_2)]$ and $[\mathsf{T_R}{}^{in}_{tid''_R}|$ $\mathsf{verifyRV}(\mathsf{S}, \mathsf{R}, \mathsf{O}, d_R, m_2)]$. If O obtains an output $[\mathsf{R_R}{}^{out}_{tid''_R}|\text{received: } m_2]$ during “verifyRV“, it outputs $[\mathsf{O}^{out}_{tid}|\text{exchanged}]$. If the third party obtains an output $[\mathsf{T_R}{}^{out}_{tid''_R}|\text{failed}]$, it aborts this run. Else, if it receives $[\mathsf{T_R}{}^{out}_{tid''_R}|$ not_sent], it inputs $[\mathsf{T_O}{}^{in}_{tid'_O}|\mathsf{revoke}(\mathsf{S}, \mathsf{R}, d_O, m_2)]$. At the same time t_R, O inputs $[\mathsf{So}^{in}_{tid'_O}|\mathsf{revoke}(m_2)]$. Finally, after an output $[\mathsf{So}^{out}_{tid'_O}|\mathsf{revoked}]$, machine O outputs "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$".

◇

Theorem 13.2 (Security of Scheme 13.2)
Scheme 13.2 is a secure transfer-based fair exchange protocol for revocable and recipient-verifiable items. □

Proof. We show that all requirements of Definitions 12.2 and 12.4 are fulfilled by Scheme 13.2 if I_O is revocable and I_R is recipient verifiable:

Correct Execution (R. 12.2a) and Availability (R. 12.4b): If both machines are correct and receive non-matching inputs, R outputs failed after receiving an m_1 with unexpected parameters, whereas O outputs failed after not receiving m_2. Else, they start the "prepareRV"-protocol with matching inputs.

If both machines are able to transfer the items to each other, the protocol outputs $[\mathsf{R_R}^{out}_{tid_R}|\mathsf{providesRV}\text{: } d_R]$ on inputs $[\mathsf{S_R}^{in}_{tid_R}|\mathsf{prepareRV}(\mathsf{O}, d_R, \mathsf{T}, t_V, m_2)]$ and $[\mathsf{R_R}^{in}_{tid_R}|\mathsf{checkRV}(\mathsf{T}, t_V, m_2)]$ [R. 12.7a]. Then, both input $[\mathsf{S_O}^{in}_{tid_O}|\mathsf{transferR}(\mathsf{R}, d_O, \mathsf{T}, t_R, m_2)]$ and $[\mathsf{R_O}^{in}_{tid_O}|\mathsf{receiveR}(\mathsf{T}, t_R, m_2)]$. Since O is able to transfer d_O, this leads to an output $[\mathsf{R_O}^{out}_{tid_O}|\mathsf{receivedR}\text{: } d_O]$ according to the definition of δ_R since $t_R > t_0 + \Delta_t^R$. Therefore, R will input $[\mathsf{S_R}^{in}_{tid'_R}|\mathsf{transferRV}(m_2)]$ and output "$[\mathsf{R}^{out}_{tid}|\mathsf{exchanged}]$". Since $\mathsf{S_R}$ is able to transfer d_R, this will lead to an output output $[\mathsf{R_R}^{in}_{tid'_R}|\mathsf{received}(m_2)]$ [R. 12.7a] and R will output "$[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$" as well.

Transfer (R. 12.2b): Participant O only outputs "$[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$" after receiving the expected transfer. From [R. 12.7c] follows, that it owns the item.

Participant R outputs "$[\mathsf{O}^{out}_{tid}|\mathsf{exchanged}]$" after receiving the expected transfer and not receiving an output $[\mathsf{S_R}^{out}_{tid'_R}|\mathsf{failed}]$ or $[\mathsf{S_R}^{out}_{tid''_R}|\mathsf{not\_sent}\text{: } m_2]$. From the correctness of the sender follows that T does not output $[\mathsf{S_R}^{out}_{tid'_R}|\mathsf{not\_sent}]$. As a consequence, it will not revoke the received transfer. From [R. 12.6c] then follows that R owns the item.

No Surprises (R. 12.2c) If participant R outputs "$[\mathsf{R}^{out}_{tid}|\mathsf{failed}]$", it either does not input $[\mathsf{S_R}^{in}_{tid'_R}|\mathsf{transferRV}(\dots)]$ or else it output $[\mathsf{S_R}^{out}_{tid'_R}|\mathsf{failed}]$ or $[\mathsf{S_R}^{out}_{tid''_R}|\mathsf{not\_sent}\text{: } m_2]$. From [R. 12.7d] follows that $\mathsf{R_R}$ does not obtain knowledge about d_R.

If "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$" is output before $[\mathsf{S_O}^{in}_{tid_O}|\mathsf{transferR}(\dots)]$ is input or if this input was answered with $[\mathsf{S_O}^{out}_{tid_O}|\mathsf{failed}]$, the recipient does not obtain knowledge about the item [R. 12.6d]. Else, if it output "$[\mathsf{O}^{out}_{tid}|\mathsf{failed}]$" after recovery, it obtained an output $[\mathsf{S_O}^{out}_{tid}|\mathsf{revoked}\text{: } m_2]$ and $\mathsf{R_O}$ does not obtain knowledge about the item d_R as well [R. 12.6b, R. 12.6d].

Termination (R. 12.2d): Since all sub-protocols terminate in a fixed time [R. 12.5g,R. 12.8f], the protocols terminate in a fixed time, too.

No Loss (R. 12.4c): Similar to "no surprises" using [R. 12.6e, R. 12.7e], except that the sender of the recipient-verifiable item is re-enabled to transfer the item after a delay of $t_V + \Delta_v$.

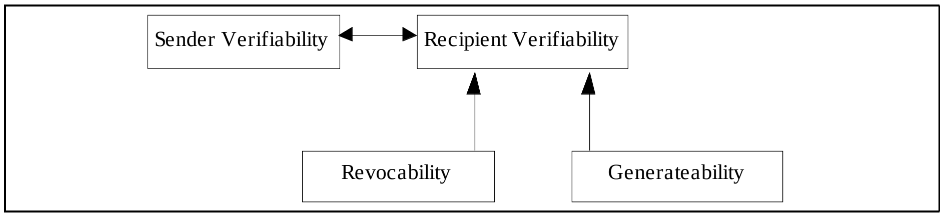


Figure 13.5: Simulations between Exchange-enabling Transfers.

Sender S		**Recipient** R
"$[\mathsf{S}^{in}_{tid}\mid\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$"		"$[\mathsf{R}^{in}_{tid}\mid\mathsf{checkRV}(\mathsf{T}, t_T, xref)]$"
	$m_1 := \mathrm{sign}_{\mathsf{S}}(\mathsf{R}, d, t_T, xref, tid)$ $\longrightarrow$	
		"$[\mathsf{R}^{out}_{tid}\mid\mathsf{providesRV}\colon d]$"

Figure 13.6: Preparation of Recipient Verifiability using Sender Verifiability.

■

13.3 Simulations of Exchange-enabling Transfers

We now describe simulations between different exchange-enabling transfers, i.e., how to use certain exchange-enabling properties to provide others. These simulations can be used to "map" a given property to the properties needed by our exchange protocols. This enables us to exchange any two items where at least one item is generatable or revocable. Second, it shows that on synchronous networks, both types of verifiability are equivalent, i.e., one can be used to simulate the other.

Our simulations are depicted in Figure 13.5: We show that the recipient and sender verifiability is weaker than generatability and revocability. We do this by describing protocols that provide verifiability given revocability or generatability. Furthermore, we show that on synchronous networks sender and recipient verifiability are equivalent, i.e., can be simulated with each other. These simulations are based on three-party disputes and cannot be used on asynchronous networks. Moreover, on asynchronous networks, we believe that sender and recipient verifiability are fundamentally different, i.e., neither of them can be used to simulate the other.

13.3.1 Recipient Verifiability using Sender Verifiability

We first show how to simulate recipient verifiability using sender verifiability. Basically, during "prepareRV", the sender promises to send an item. Then, during a "verifyRV"-protocol, the third party forces the sender to execute "verifySV" and decides against the sender if the sender is unable to show that the recipient obtained the item. The protocols are depicted in Figures 13.6, 13.7, and 13.8.

Lemma 13.1 (Simulation $SV \to RV$)
Let $I' = (\mathsf{S}', \mathsf{R}', \mathsf{T}', B, XREFs, TIDs, \delta'_{SV}, (\Delta^{SV}_t, \Delta^{SV}_v))$ be a transferable item with sender verifiability. Then, the item $I = (\langle \mathsf{S}, \mathsf{S}'\rangle, \langle \mathsf{R}, \mathsf{R}'\rangle\ \langle \mathsf{T}, \mathsf{T}'\rangle, B, XREFs,$

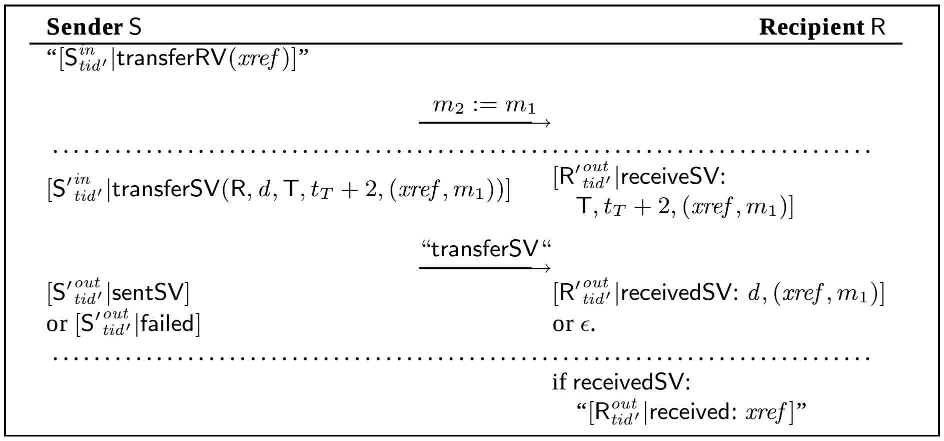

Sender S
Recipient R
“[$\mathsf{S}^{in}_{tid'}$|transferRV($xref$)]”
$m_2 := m_1$
[$\mathsf{S}'^{in}_{tid'}$|transferSV(R, d, T, $t_T + 2$, ($xref$, m_1))]
[$\mathsf{R}'^{out}_{tid'}$|receiveSV: T, $t_T + 2$, ($xref$, m_1)]
“transferSV“
[$\mathsf{S}'^{out}_{tid'}$|sentSV] or [$\mathsf{S}'^{out}_{tid'}$|failed]
[$\mathsf{R}'^{out}_{tid'}$|receivedSV: d, ($xref$, m_1)] or ϵ.
if receivedSV: “[$\mathsf{R}^{out}_{tid'}$|received: $xref$]”

Figure 13.7: Transfer of Recipient Verifiability using Sender Verifiability.

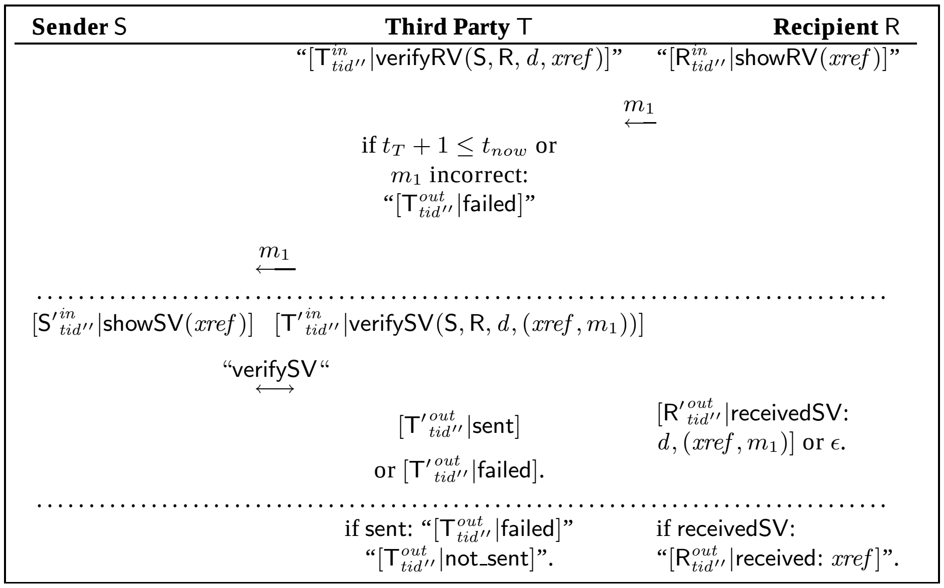

Sender S
Third Party T
Recipient R
“[$\mathsf{T}^{in}_{tid''}$|verifyRV(S, R, d, $xref$)]”
“[$\mathsf{R}^{in}_{tid''}$|showRV($xref$)]”
m_1
if $t_T + 1 \leq t_{now}$ or m_1 incorrect: “[$\mathsf{T}^{out}_{tid''}$|failed]”
m_1
[$\mathsf{S}'^{in}_{tid''}$|showSV($xref$)]
[$\mathsf{T}'^{in}_{tid''}$|verifySV(S, R, d, ($xref$, m_1))]
“verifySV“
[$\mathsf{T}'^{out}_{tid''}$|sent] or [$\mathsf{T}'^{out}_{tid''}$|failed].
[$\mathsf{R}'^{out}_{tid''}$|receivedSV: d, ($xref$, m_1)] or ϵ.
if sent: “[$\mathsf{T}^{out}_{tid''}$|failed]” “[$\mathsf{T}^{out}_{tid''}$|not_sent]”.
if receivedSV: “[$\mathsf{R}^{out}_{tid''}$|received: $xref$]”.

Figure 13.8: Verification of Recipient Verifiability using Sender Verifiability.

$TIDs$, δ_{RV}, $(\Delta_p, \Delta_t, \Delta_v)$) as defined below offers recipient verifiability. I is optimistic if item I' is optimistic.

The behavior of I is defined by the behavior of S, R, *and* T *as follows:*

Protocol “prepareRV“: *On input “*[S^{in}_{tid}|prepareRV(R, d, T, t_T, $xref$)]*” and “*[R^{in}_{tid}| checkRV(T, t_T, $xref$)]*” machine* S *sends* $m_1 := \text{sign}_\mathsf{S}$(R, d, t_T, $xref$, tid) *to* R *who outputs “*[R^{out}_{tid}|providesRV: d]*”.*

Protocol “transferRV“: *On input “*[$\mathsf{S}^{in}_{tid'}$|transferRV($xref$)]*”, the sender sends $m_2 := m_1$ and inputs* [$\mathsf{S}'^{in}_{tid'}$|transferSV(R, d, T, $t_T + 2$, ($xref$, m_1))] *(sender verifiability needs to be guaranteed at time $t_T + 2$ since a recipient verification started at time t_T will start the sub-protocol for sender verification at time $t_T + 2$). Upon receipt of m_1, the recipient inputs* [$\mathsf{R}'^{in}_{tid'}$|receiveSV(T, $t_T + 2$, ($xref$, m_1))].

On output of $[\mathsf{R'}^{out}_{tid'}|\mathsf{receivedSV}\textit{: } d, (\mathit{xref}, m_1)]$*, the recipient outputs* "$[\mathsf{R}^{out}_{tid'}|\mathsf{received}\textit{: } \mathit{xref}]$".

Protocol "verifyRV": *On input* "$[\mathsf{R}^{in}_{tid''}|\mathsf{showRV}(\mathit{xref})]$" *and* "$[\mathsf{T}^{in}_{tid''}|\mathsf{verifyRV}(\mathsf{S}, \mathsf{R}, d, \mathit{xref})]$"*, the recipient sends* m_1 *to* T *who checks that the request is in time (i.e., whether* $t_T \leq t_{now} + 1$*) and that the parameters contained in* m_1 *match the input ones. If this is the case,* T *sends* m_1 *to* S *in Round 2. If this is not the case, it outputs* "$[\mathsf{T}^{out}_{tid''}|\mathsf{failed}]$"*. Upon receipt of* m_1*, machine* S *inputs* $[\mathsf{S'}^{in}_{tid''}|\mathsf{showSV}(\mathit{xref})]$ *in Round 3. Furthermore the third party inputs* $[\mathsf{T'}^{in}_{tid''}|\mathsf{verifySV}(\mathsf{S}, \mathsf{R}, d, (\mathit{xref}, m_1))]$ *in Round 3.*

If the "verifySV"*-protocol outputs* $[\mathsf{T'}^{out}_{tid''}|\mathsf{failed}]$*, the third party outputs* "$[\mathsf{T}^{out}_{tid''}|\mathsf{not\_sent}]$" *(recall that* R*'s goal was to show that it was unable to receive the item). If it outputs* $[\mathsf{T'}^{out}_{tid''}|\mathsf{sent}]$*, the third party outputs* "$[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$".

If the recipient obtains an output $[\mathsf{R'}^{out}_{tid''}|\mathsf{receivedSV}\textit{: } d, (\mathit{xref}, m_1)]$*, it outputs* "$[\mathsf{R}^{out}_{tid''}|\mathsf{received}\textit{: } \mathit{xref}]$".

We define $\Delta_p := 1$*,* $\Delta_t := \Delta_t^{SV} + 1$*,* $\Delta_v := \Delta_v^{SV} + 2$*.* □

Proof. Item I is as optimistic as I' by construction. We show that I fulfills the requirements defined in Definition 12.7.

Correct Execution (R. 12.7a): The inputs "$[\mathsf{S}^{in}_{tid}|\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_T, \mathit{xref})]$" and "$[\mathsf{R}^{in}_{tid}|\mathsf{checkRV}(\mathsf{T}, t_T, \mathit{xref})]$" always lead to an output "$[\mathsf{R}^{out}_{tid}|\mathsf{providesRV}\text{: } d]$" if the players are correct.

Correct Verification (R. 12.7b): If the recipient output "$[\mathsf{R}^{out}_{tid}|\mathsf{providesRV}\text{: } d]$" it received a correct m_1. Therefore, the inputs "$[\mathsf{T}^{in}_{tid''}|\mathsf{verifyRV}(\mathsf{S}, \mathsf{R}, d, \mathit{xref})]$" and "$[\mathsf{R}^{in}_{tid''}|\mathsf{showRV}(\mathit{xref})]$" before time t_T lead to an output "$[\mathsf{T}^{out}_{tid''}|\mathsf{not\_sent}]$" except if the sub-protocol "verifySV" outputs $[\mathsf{T'}^{out}_{tid''}|\mathsf{sent}]$. However, if sent is output, a correct recipient output "$[\mathsf{R}^{out}_{tid''}|\mathsf{received}\text{: } \mathit{xref}]$" [R. 12.5c].

Note that "verifySV" may output "$[\mathsf{T}^{out}_{tid''}|\mathsf{not\_sent}]$" even if R output "$[\mathsf{R}^{out}_{tid''}|\mathsf{received}\text{: } \mathit{xref}]$" if an incorrect machine S refuses to participate in "verifyRV". This, however, does not contradict R. 12.7b.

Transfer (R. 12.7c): This follows directly from [R. 12.5d].

No Surprises (R. 12.7d): If the sender does not input "$[\mathsf{S}^{in}_{tid'}|\mathsf{transferRV}(\mathit{xref})]$", $[\mathsf{S'}^{in}_{tid'}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T + 2, (\mathit{xref}, m_1))]$ is not input either. No surprises then follows from [R. 12.5e].

If the third party outputs "$[\mathsf{T}^{out}_{tid''}|\mathsf{not\_sent}]$", then a correct S executed "verifySV" before $t_T + 2$, which output $[\mathsf{T'}^{out}_{tid''}|\mathsf{failed}]$. From [R. 12.5b] follows that no output $[\mathsf{S'}^{out}_{tid''}|\mathsf{sentSV}]$ was made on input of $[\mathsf{S}^{in}_{tid'}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T + 2, (\mathit{xref}, m_1))]$. From [R. 12.5g] follows that S output $[\mathsf{S'}^{out}_{tid'}|\mathsf{failed}]$ in the transfer. From [R. 12.5e] then follows that no R is able to obtain knowledge about the item.

No Loss (R. 12.7e): Follows similarly from [R. 12.5f].

Termination (R. 12.7f): Follows directly from [R. 12.5g].

We now show that $\delta_{RV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ iff $\delta_{SV}(\sigma_{S'}, \sigma_{R'}, d) = \mathsf{true}$ for the sub-machines: Let us first assume that $\delta_{RV} = \mathsf{true}$ holds. If prepareRV and checkRV are input, m_1 is sent in "“prepareRV“" and a subsequent input of transferRV leads to the input of transferSV and receiveSV. This results in an output receivedSV that leads to an output "received", i.e., $\delta_{RV} = \mathsf{true}$ holds as well. Let us now assume that $\delta_{RV} = \mathsf{true}$. Then prepareRV, checkRV, and transferRV leads to an output received. As a consequence, the underlying protocol “transferSV“ output sent, i.e., $\delta_{SV} = \mathsf{true}$ holds as well. ∎

13.3.2 Sender Verifiability using Recipient Verifiability

We now sketch how to simulate sender verifiability using recipient verifiability. The main problem of this simulation is that the sender is unable to find out whether the recipient obtained the item or not without executing “verifyRV“. Therefore, in case of faults, this protocol is needed for recovery.

The resulting simulation is similar to a generalization of an optimistic certified mail protocol: During “transferSV“, the protocols “prepareRV“ and “transferRV“ are executed and acknowledged by the recipient. If an incorrect recipient refuses to acknowledge the receipt of a transfer, the third party forces the recipient to execute “verifyRV“ and decides against the recipient if the recipient is unable to show that it did not obtain the item. During verification at the third party, the sender either shows the acknowledgment from the recipient or from the third party.

The protocols are depicted in Figure 13.9, 13.10, and 13.11.

Lemma 13.2 (Simulation $RV \rightarrow SV$)
Let $I' = (\mathsf{S}', \mathsf{R}', \mathsf{T}', B, \mathit{XREFs}, \mathit{TIDs}, \delta_{RV}, (\Delta_p, \Delta_t^{RV}, \Delta_v^{RV}))$ be a transferable item with recipient verifiability. Then, the item $I = (\langle \mathsf{S}, \mathsf{S}' \rangle, \langle \mathsf{R}, \mathsf{R}' \rangle\ \langle \mathsf{T}, \mathsf{T}' \rangle, B, \mathit{XREFs}, \mathit{TIDs}, \delta_{SV}, , (\Delta_t, \Delta_v))$ as defined below offers sender verifiability. I is optimistic if item I' is optimistic.

The behavior of I is defined by the behavior of S, R, *and* T *as follows:*

Protocol “transferSV“: *On input of* “$[\mathsf{S}_{tid}^{in}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T, \mathit{xref})]$” *and* “$[\mathsf{R}_{tid}^{in}|\mathsf{receiveSV}(\mathsf{T}, t_T, \mathit{xref})]$”, *the machines* S *and* R *inputs* $[\mathsf{S'}_{tid}^{in}|\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_v, \mathit{xref})]$ *and* $[\mathsf{S'}_{tid}^{in}|\mathsf{checkRV}(\mathsf{T}, t_v, \mathit{xref})]$ *with* $t_v := t_0 + \Delta_p + \Delta_t^{RV} + 4$. *On output* $[\mathsf{R'}_{tid}^{out}|\mathsf{providesRV}: d]$, *the recipient sends* $m_1 := \mathrm{sign}_{\mathsf{R}}(\mathsf{T}, d, t_T, \mathit{xref}, \mathit{tid}, \mathcal{H}(r))$ *for a random* r[7]. *If* S *does not receive* m_1 *it outputs* “$[\mathsf{S}_{tid}^{out}|\mathsf{failed}]$”. *Else, it inputs* $[\mathsf{S'}_{tid'}^{in}|\mathsf{transferRV}(\mathit{xref})]$. *On output* $[\mathsf{R'}_{tid'}^{out}|\mathsf{received}: \mathit{xref}]$, *the recipient sends* $m_2 := r$ *and outputs* “$[\mathsf{R}_{tid}^{out}|\mathsf{receivedSV}: d, \mathit{xref}]$”. *If the* “transferRV“*-protocol outputs* $[\mathsf{S'}_{tid'}^{out}|\mathsf{failed}]$, *the sender outputs* “$[\mathsf{S}_{tid}^{out}|\mathsf{failed}]$”.

If S *receives a correct* m_2, *it outputs* “$[\mathsf{S}_{tid}^{out}|\mathsf{sentSV}]$”. *Else, it starts the sub-protocol* “resolveSV“.

Sub-protocol “resolveSV“: *For resolving, the sender sends* $\mathrm{sign}_{\mathsf{S}}(m_1)$, *which is forwarded by* T *in Round 2 to* R *who inputs* $[\mathsf{R'}_{tid''}^{in}|\mathsf{showRV}(\mathit{xref})]$

[7] $\mathcal{H}()$ is a secure one-way function.

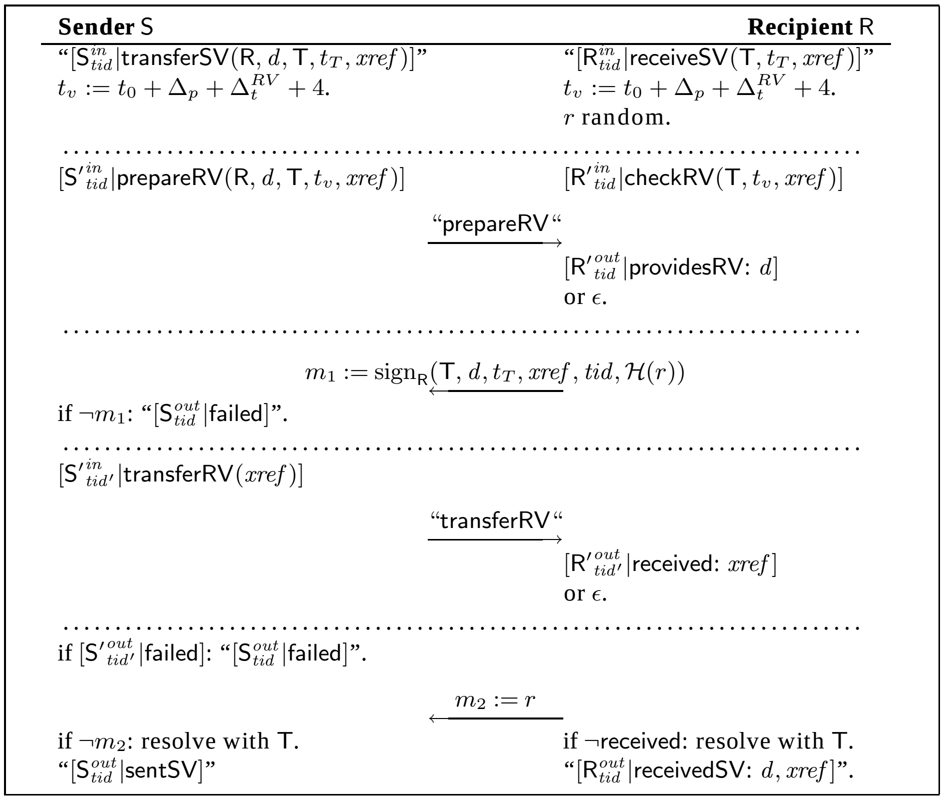

Sender S
Recipient R
“$[\mathsf{S}^{in}_{tid}|\mathsf{transferSV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$”
$t_v := t_0 + \Delta_p + \Delta_t^{RV} + 4.$
“$[\mathsf{R}^{in}_{tid}|\mathsf{receiveSV}(\mathsf{T}, t_T, xref)]$”
$t_v := t_0 + \Delta_p + \Delta_t^{RV} + 4.$
r random.
$[\mathsf{S}'^{in}_{tid}|\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_v, xref)]$
$[\mathsf{R}'^{in}_{tid}|\mathsf{checkRV}(\mathsf{T}, t_v, xref)]$
$\xrightarrow{\text{“prepareRV“}}$
$[\mathsf{R}'^{out}_{tid}|\mathsf{providesRV}\text{: } d]$
or ϵ.
$\xleftarrow{m_1 := \mathrm{sign}_{\mathsf{R}}(\mathsf{T}, d, t_T, xref, tid, \mathcal{H}(r))}$
if $\neg m_1$: “$[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$”.
$[\mathsf{S}'^{in}_{tid'}|\mathsf{transferRV}(xref)]$
$\xrightarrow{\text{“transferRV“}}$
$[\mathsf{R}'^{out}_{tid'}|\mathsf{received}\text{: } xref]$
or ϵ.
if $[\mathsf{S}'^{out}_{tid'}|\mathsf{failed}]$: “$[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$”.
$\xleftarrow{m_2 := r}$
if $\neg m_2$: resolve with T.
“$[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$”
if $\neg$received: resolve with T.
“$[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\text{: } d, xref]$”.

Figure 13.9: Sender Verifiability using Recipient Verifiability.

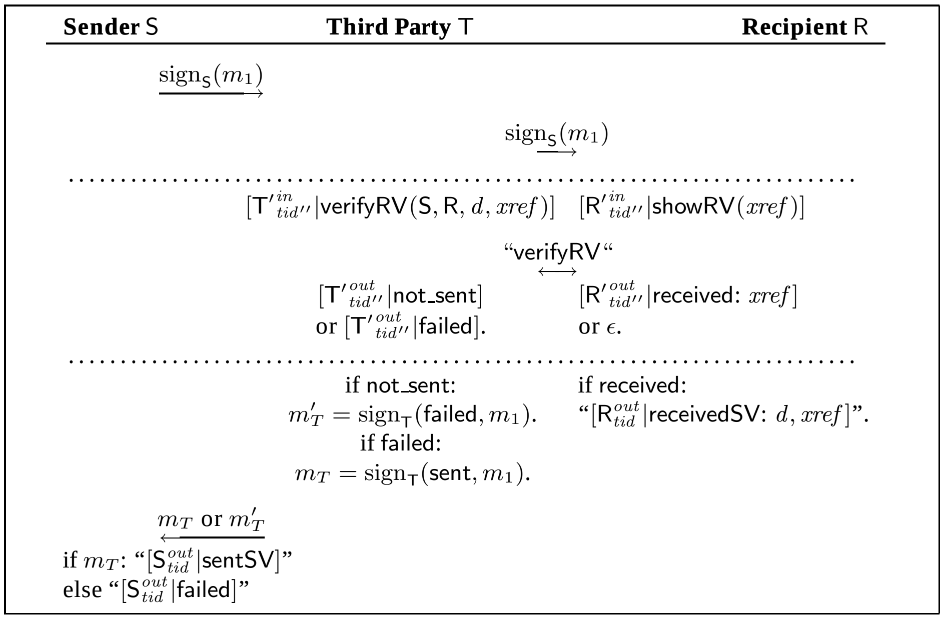

Sender S
Third Party T
Recipient R
$\xrightarrow{\mathrm{sign}_{\mathsf{S}}(m_1)}$
$\xrightarrow{\mathrm{sign}_{\mathsf{S}}(m_1)}$
$[\mathsf{T}'^{in}_{tid''}|\mathsf{verifyRV}(\mathsf{S}, \mathsf{R}, d, xref)]$
$[\mathsf{R}'^{in}_{tid''}|\mathsf{showRV}(xref)]$
$\xleftrightarrow{\text{“verifyRV“}}$
$[\mathsf{T}'^{out}_{tid''}|\mathsf{not\_sent}]$
or $[\mathsf{T}'^{out}_{tid''}|\mathsf{failed}]$.
$[\mathsf{R}'^{out}_{tid''}|\mathsf{received}\text{: } xref]$
or ϵ.
if not_sent:
$m'_T = \mathrm{sign}_{\mathsf{T}}(\mathsf{failed}, m_1)$.
if failed:
$m_T = \mathrm{sign}_{\mathsf{T}}(\mathsf{sent}, m_1)$.
if received:
“$[\mathsf{R}^{out}_{tid}|\mathsf{receivedSV}\text{: } d, xref]$”.
$\xleftarrow{m_T \text{ or } m'_T}$
if m_T: “$[\mathsf{S}^{out}_{tid}|\mathsf{sentSV}]$”
else “$[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$”

Figure 13.10: Recovery of Sender Verifiability using Recipient Verifiability.

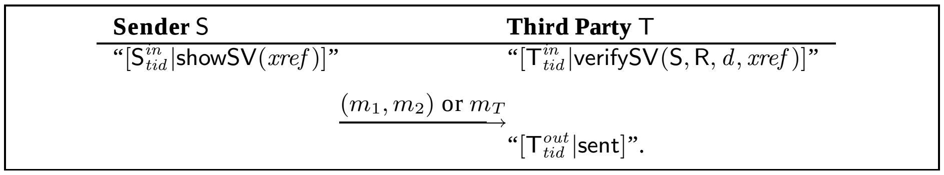

Sender S
Third Party T
“[S^{in}_{tid}|showSV($xref$)]”
“[T^{in}_{tid}|verifySV(S, R, d, $xref$)]”
$\xrightarrow{(m_1, m_2) \text{ or } m_T}$
“[T^{out}_{tid}|sent]”.

Figure 13.11: Verification of Sender Verifiability using Recipient Verifiability.

in Round 3. On input [$\mathsf{T'}^{in}_{tid''}$|verifyRV(S, R, d, $xref$)]*, machine* T *waits for an output* [$\mathsf{T'}^{out}_{tid''}$|not_sent] *or* [$\mathsf{T'}^{out}_{tid''}$|failed]*. If* not_sent *is output,* T *sends* m'_T:=sign_T(failed, m_1) *and, upon receipt, the sender outputs* “[S^{out}_{tid}| failed]”*. If* failed *is output, it sends* m_T := sign_T(sent, m_1) *and the sender outputs* “[S^{out}_{tid}|sentSV]”*.*

Protocol “verifySV“: *On input* “[S^{in}_{tid}|showSV($xref$)]” *and* “[T^{in}_{tid}|verifySV(S, R, d, $xref$)]”*, the sender either sends* (m_1, m_2) *or* m_T*. If one of these messages is received correctly, the third party outputs* “[T^{out}_{tid}|sent]” *and* “[T^{out}_{tid}| failed]”*, else.*

We define $\Delta_t := \Delta_p + \Delta_t^{RV} + \Delta_v^{RV} + 5$*,* $\Delta_v := 1$*.* □

Proof. We show that I fulfills the requirements defined in Definition 12.5:

Correct Execution (R. 12.5a): If a correct sender is able to transfer d to R, a correct recipient will obtain an output [$\mathsf{R'}^{out}_{tid'}$|received: $xref$] and will send m_2. Therefore, the sender will output “[S^{out}_{tid}|sentSV]”.

If a correct sender cannot transfer d to R, S′ will output [$\mathsf{S'}^{out}_{tid'}$|failed] on input of [$\mathsf{S'}^{in}_{tid'}$|transferRV($xref$)] and will output “[S^{out}_{tid}|failed]”.

Sender Verifiability (R. 12.5b): If the sender outputs “[S^{out}_{tid}|sentSV]”, it either received (m_1, m_2) or m_T, which both lead to an output “[T^{out}_{tid}|sent]”.

Correct Verification (R. 12.5c): If the third party output sent on receipt of (m_1, m_2), then the correct recipient obtained an output [$\mathsf{R'}^{out}_{tid'}$|received: $xref$] (otherwise, it would not send m_2). This leads to an output “[R^{out}_{tid}| receivedSV: $d, xref$]”.

If the third party output sent on receipt of m_T, then the execution of “prepareRV“ output [$\mathsf{R'}^{out}_{tid}$|providesRV: d] because T requires m_1 and m_1 is only sent with these parameters. Therefore, protocol “verifyRV“ output [$\mathsf{T'}^{out}_{tid''}$|failed] and a correct recipient output [$\mathsf{R'}^{out}_{tid'}$|received: $xref$] [R. 12.7b] since the verification was started before time t_v. This leads to an output “[R^{out}_{tid}|receivedSV: $d, xref$]”.

Transfer (R. 12.5d): Follows directly from [R. 12.7c].

No Surprises (R. 12.5e): If no input transferRV is made, if “[S^{out}_{tid}|failed]” is output without receiving m_1, or if [$\mathsf{S'}^{out}_{tid'}$|failed] is output by “transferRV“, this follows directly from [R. 12.7c].

If “[S^{out}_{tid}|failed]” was output after receiving m'_T, the third party output [$\mathsf{T'}^{out}_{tid''}$|not_sent] on input [$\mathsf{T'}^{in}_{tid''}$|verifyRV(S, R, d, $xref$)] with the parameters fixed in $\mathrm{sign}_\mathsf{S}(m_1)$. Therefore, the recipient is unable to obtain knowledge about the item [R. 12.7d].

Sender S		**Recipient** R
“[S^{in}_{tid}|prepareRV(R, d, T, t_T, $xref$)]” $xref' :=$ (S, R, $t_T + 1$, $xref$).		“[R^{in}_{tid}|checkRV(T, t_T, $xref$)]” $xref' :=$ (S, R, $t_T + 1$, $xref$).
....	....	....
[$\mathsf{S'}^{in}_{tid}$|prepareG(R, d, T, $t_T + 1$, $xref'$)]		[$\mathsf{R'}^{in}_{tid}$|checkG(T, $t_T + 1$, $xref'$)]
	$\xrightarrow{\text{“prepareG“}}$	
		[$\mathsf{R'}^{out}_{tid}$|providesG: d] or ϵ.
....	....	....
		if providesG: “[R^{out}_{tid}|providesRV: d]”

Figure 13.12: Preparation of Recipient Verifiability using Generatability.

No Loss (R. 12.5f): Reasoning similar based on [R. 12.7e].

Termination (R. 12.5g): Since all sub-protocols terminate in a fixed time, the protocols terminate in a fixed time, too.

If I is optimistic and the parties are correct no recovery is needed and only “prepareRV“ and “transferRV“ are executed. Therefore, the third party does not participate, i.e., the resulting exchange is optimistic, too.

We now show that $\delta_{RV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ iff $\delta_{SV}(\sigma_{S'}, \sigma_{R'}, d) = \mathsf{true}$ for the sub-machines: If we assume that $\delta_{RV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ then the inputs by the “transferSV“-protocol to the sub-machines result in an output [$\mathsf{R'}^{out}_{tid'}$|received: $xref$]. This leads to an output “[R^{out}_{tid}|receivedSV: $d, xref$]”, i.e., $\delta_{SV}(\sigma_{S'}, \sigma_{R'}, d) = \mathsf{true}$. If we assume that $\delta_{SV}(\sigma_{S'}, \sigma_{R'}, d) = \mathsf{true}$ then the “transferSV“-protocol executed with matching parameters output “[R^{out}_{tid}| receivedSV: $d, xref$]”. By construction, the underlying “transferRV“-protocol output [$\mathsf{R'}^{out}_{tid'}$|received: $xref$], i.e., $\delta_{RV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ holds. ■

Remark 13.1. For simulating sender verifiability using revocability directly, one could pursue a similar approach, i.e., one could transfer the item with revocability and revoke it if the recipient does not send a receipt. To prevent incorrect revocation, one either needs to ask the recipient whether it is willing to re-send the receipt, or else the third party needs to store which runs have been revoked and which have not. ○

13.3.3 Recipient Verifiability using Generatability

We now sketch how to simulate recipient verifiability using generatability. The underlying idea is that during “verifyRV“, the item is always generated and therefore, an output “[T^{out}_{tid}|not_sent]” will never occur. The protocols are depicted in Figure 13.12, 13.13, and 13.14.

Lemma 13.3 (Simulation $G \rightarrow RV$)
Let $I' = (\mathsf{S'}, \mathsf{R'}, \mathsf{T'}, B, XREFs, TIDs, \delta_G, (\Delta^G_p, \Delta^G_t, \Delta_g))$ be a transferable item with generatability. Then, the item $I = (\langle \mathsf{S}, \mathsf{S'} \rangle, \langle \mathsf{R}, \mathsf{R'} \rangle\ \langle \mathsf{T}, \mathsf{T'} \rangle, B, XREFs, TIDs, \delta_{RV}, (\Delta_p, \Delta_t, \Delta_v))$ as defined below offers recipient verifiability. I is optimistic if item I' is optimistic.

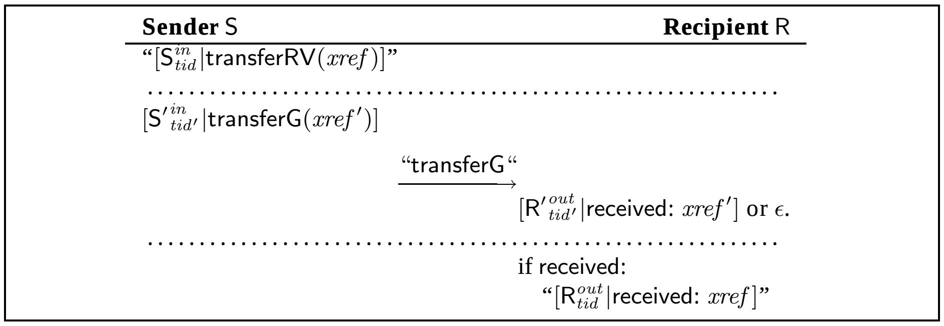

Sender S
Recipient R
“[S^{in}_{tid}|transferRV($xref$)]”
[$\mathsf{S'}^{in}_{tid'}$|transferG($xref'$)]
“transferG“
[$\mathsf{R'}^{out}_{tid'}$|received: $xref'$] or ϵ.
if received:
“[R^{out}_{tid}|received: $xref$]”

Figure 13.13: Transfer of Recipient Verifiability using Generatability.

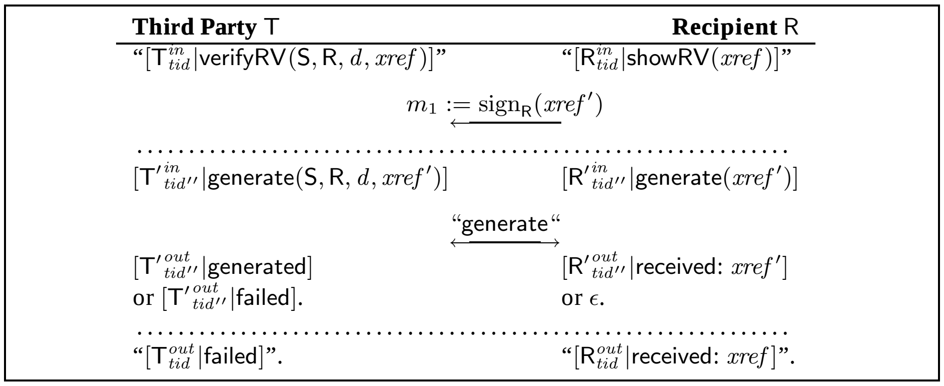

Third Party T
Recipient R
“[T^{in}_{tid}|verifyRV(S, R, $d, xref$)]”
“[R^{in}_{tid}|showRV($xref$)]”
$m_1 := \text{sign}_\mathsf{R}(xref')$
[$\mathsf{T'}^{in}_{tid''}$|generate(S, R, $d, xref'$)]
[$\mathsf{R'}^{in}_{tid''}$|generate($xref'$)]
“generate“
[$\mathsf{T'}^{out}_{tid''}$|generated]
or [$\mathsf{T'}^{out}_{tid''}$|failed].
[$\mathsf{R'}^{out}_{tid''}$|received: $xref'$]
or ϵ.
“[T^{out}_{tid}|failed]”.
“[R^{out}_{tid}|received: $xref$]”.

Figure 13.14: Verification of Recipient Verifiability using Generatability.

The behavior of I is defined by the behavior of S, R, *and* T *as follows:*

Protocol “prepareRV“: *On input* “[S^{in}_{tid}|prepareRV(R, d, T, t_T, $xref$)]” *and* “[R^{in}_{tid}|checkRV(T, t_T, $xref$)]”*, the machines compute* $xref' :=$ (S, R, $t_T + 1$, $xref$) *and input* [$\mathsf{S'}^{in}_{tid}$|prepareG(R, d, T, $t_T + 1, xref'$)] *and* [$\mathsf{R'}^{in}_{tid}$|checkG(T, $t_T + 1, xref'$)]*, respectively. If* [$\mathsf{R'}^{out}_{tid}$|providesG*:* d] *is output, the recipient outputs* “[R^{out}_{tid}|providesRV*:* d]”*.*

Protocol “transferRV“: *On input* “[S^{in}_{tid}|transferRV($xref$)]”*, the sender inputs* [$\mathsf{S'}^{in}_{tid'}$|transferG($xref'$)].

On output of [$\mathsf{R'}^{out}_{tid'}$|received*:* $xref'$] *by* R′*, the recipient outputs* “[R^{out}_{tid}| received*:* $xref$]”*.*

Protocol “verifyRV“: *On input* “[R^{in}_{tid}|showRV($xref$)]” *and* “[T^{in}_{tid}|verifyRV(S, R, $d, xref$)]”*, the recipient sends* $m_1 := \text{sign}_\mathsf{R}(xref')$ *to* T*. Then, the parties input* [$\mathsf{R'}^{in}_{tid''}$|generate($xref'$)] *and* [$\mathsf{T'}^{in}_{tid''}$|generate(S, R, $d, xref'$)] *and* T *outputs* “[T^{out}_{tid}|failed]”*. Machine* S *ignores the output* [$\mathsf{S'}^{out}_{tid''}$|generated*:* $xref'$].

We define $\Delta_p := \Delta^G_p$*,* $\Delta_t := \Delta^G_t$*, and* $\Delta_v := \Delta_g + 1$. □

Proof. We show that I fulfills the requirements defined in Definition 12.7:

Correct Execution (R. 12.7a): Follows from [R. 12.8a]: With $\delta_{SV} = \delta_G$ and the definition of δ_G follows that the input of [$\mathsf{S'}^{in}_{tid}$|prepareG(R, d, T, t_T +

Sender S		**Recipient** R
“[S^{in}_{tid}|prepareRV($\mathsf{R}, d, \mathsf{T}, t_T, xref$)]”		“[R^{in}_{tid}|checkRV($\mathsf{T}, t_T, xref$)]”
	$m_1 := \text{sign}_{\mathsf{S}}(\mathsf{R}, d, t_T, xref, tid)$ $\longrightarrow$	
		“[R^{out}_{tid}|providesRV: d]”

Figure 13.15: Preparations of Recipient Verifiability using Revocability.

$1, xref')$] and [$\mathsf{S'}^{in}_{tid}$|checkG($\mathsf{T}, t_T + 1, xref'$)] leads to an output [$\mathsf{R'}^{out}_{tid}$| providesG: d] and R will output “[R^{out}_{tid}|providesRV: d]”.

Correct Verification (R. 12.7b): If a correct recipient output “[R^{out}_{tid}|providesRV: d]”, an execution of “verifyRV“ always leads to an output “[R^{out}_{tid}|received: $xref$]” [R. 12.8b].

Transfer (R. 12.7c): This follows directly from [R. 12.8c].

No Surprises (R. 12.7d): If “[S^{in}_{tid}|prepareRV($\mathsf{R}, d, \mathsf{T}, t_T, xref$)]” or else neither “[$\mathsf{S}^{in}_{tid}$|transferRV($xref$)]” nor “[$\mathsf{T}^{in}_{tid}$|verifyRV($\mathsf{S}, \mathsf{R}, d, xref$)]” was input, this follows from [R. 12.8d]. Since the protocol never outputs “[S^{out}_{tid}|failed]” or “[T^{out}_{tid}|not_sent]”, this covers all cases.

No Loss (R. 12.7e): Follows similarly from [R. 12.8e].

Termination (R. 12.7f): Follows directly from [R. 12.8f].

Since I' only calls the corresponding sub-protocols for generatability, item I' is as optimistic as item I.

For the composite machines, $\delta_{RV}(\sigma_S, \sigma_R, d) =$ true holds iff $\delta_G(\sigma_{S'}, \sigma_{R'}, d) =$ true holds for the sub-machines: If we assume that $\delta_G(\sigma_{S'}, \sigma_{R'}, d) =$ true holds then the inputs of [$\mathsf{S'}^{in}_{tid}$|prepareG($\mathsf{R}, d, \mathsf{T}, t_T + 1, xref'$)] and [$\mathsf{R'}^{in}_{tid}$|checkG($\mathsf{T}, t_T + 1, xref'$)] with matching parameters lead to an output [$\mathsf{R'}^{out}_{tid}$|providesG: d]. From [R. 12.8a] follows that a subsequent input of [$\mathsf{S'}^{in}_{tid}$|transferG($xref'$)] leads to an output [$\mathsf{R'}^{out}_{tid}$|received: $xref'$], i.e., $\delta_{RV}(\sigma_S, \sigma_R, d) =$ true. If now assume that $\delta_{RV}(\sigma_S, \sigma_R, d) =$ true holds, then the input of “[S^{in}_{tid}|prepareRV($\mathsf{R}, d, \mathsf{T}, t_T, xref$)]”, “[$\mathsf{R}^{in}_{tid}$|checkRV($\mathsf{T}, t_T, xref$)]” and “[$\mathsf{S}^{in}_{tid}$|transferRV($xref$)]” lead to an output “[$\mathsf{R}^{out}_{tid}$|received: $xref$]”. From [R. 12.7a] follows that the scheme output “[R^{out}_{tid}|providesRV: d]”, which implies that $\delta_G(\sigma_{S'}, \sigma_{R'}, d) =$ true. ■

13.3.4 Recipient Verifiability using Revocability

We now sketch how to simulate recipient verifiability using revocability. The basic idea of this simulation is that during “verifyRV“, the third party enables the sender to revoke the item and outputs [$\mathsf{T'}^{out}_{tid'}$|not_sent]. Thus, if the sender is correct, the recipient cannot obtain the item.

The protocols are depicted in Figure 13.15, 13.16, and 13.17.

Lemma 13.4 (Simulation $R \rightarrow RV$)
Let $I' = (\mathsf{S'}, \mathsf{R'}, \mathsf{T'}, B, XREFs, TIDs, \delta_R, (\Delta^R_t, \Delta_r))$ be a transferable item with revocability. Then, the item $I = (\langle \mathsf{S}, \mathsf{S'}\rangle, \langle \mathsf{R}, \mathsf{R'}\rangle\ \langle \mathsf{T}, \mathsf{T'}\rangle, B, XREFs, TIDs, \delta_{RV},$

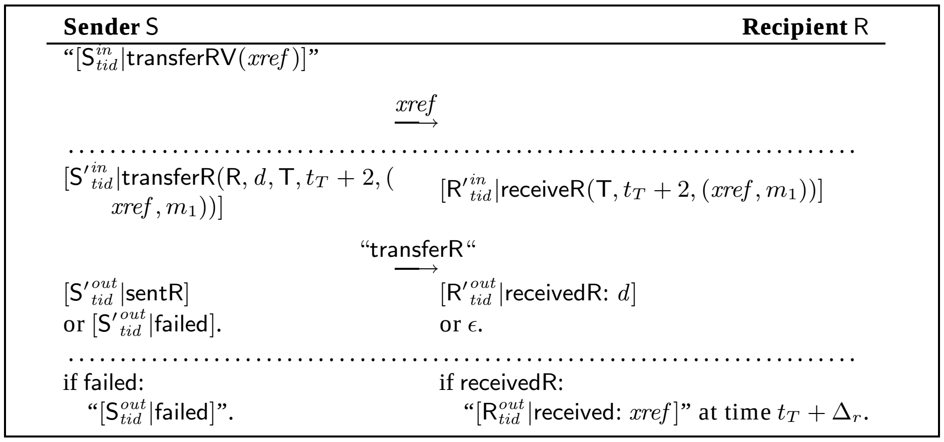

Sender S
Recipient R
“$[\mathsf{S}^{in}_{tid}|\mathsf{transferRV}(xref)]$”
$\xrightarrow{xref}$
$[\mathsf{S}'^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T + 2, (xref, m_1))]$
$[\mathsf{R}'^{in}_{tid}|\mathsf{receiveR}(\mathsf{T}, t_T + 2, (xref, m_1))]$
“transferR“ $\longrightarrow$
$[\mathsf{S}'^{out}_{tid}|\mathsf{sentR}]$ or $[\mathsf{S}'^{out}_{tid}|\mathsf{failed}]$.
$[\mathsf{R}'^{out}_{tid}|\mathsf{receivedR}\text{: } d]$ or ϵ.
if failed: “$[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$”.
if receivedR: “$[\mathsf{R}^{out}_{tid}|\mathsf{received}\text{: } xref]$” at time $t_T + \Delta_r$.

Figure 13.16: Transfer of Recipient Verifiability using Revocability.

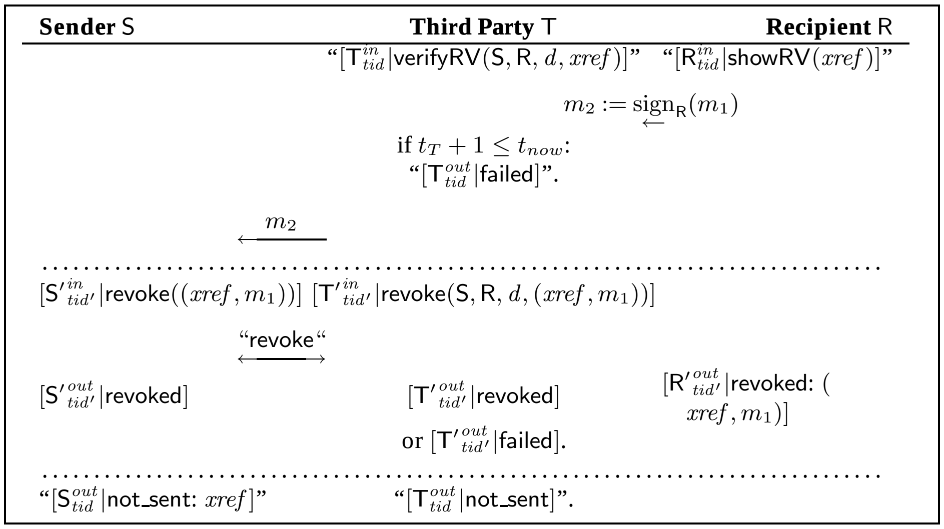

Sender S
Third Party T
Recipient R
“$[\mathsf{T}^{in}_{tid}|\mathsf{verifyRV}(\mathsf{S}, \mathsf{R}, d, xref)]$”
“$[\mathsf{R}^{in}_{tid}|\mathsf{showRV}(xref)]$”
$m_2 := \mathrm{sign}_{\mathsf{R}}(m_1)$ $\longleftarrow$
if $t_T + 1 \leq t_{now}$: “$[\mathsf{T}^{out}_{tid}|\mathsf{failed}]$”.
$\xleftarrow{m_2}$
$[\mathsf{S}'^{in}_{tid'}|\mathsf{revoke}((xref, m_1))]$
$[\mathsf{T}'^{in}_{tid'}|\mathsf{revoke}(\mathsf{S}, \mathsf{R}, d, (xref, m_1))]$
“revoke“ $\longleftrightarrow$
$[\mathsf{S}'^{out}_{tid'}|\mathsf{revoked}]$
$[\mathsf{T}'^{out}_{tid'}|\mathsf{revoked}]$ or $[\mathsf{T}'^{out}_{tid'}|\mathsf{failed}]$.
$[\mathsf{R}'^{out}_{tid'}|\mathsf{revoked}\text{: } (xref, m_1)]$
“$[\mathsf{S}^{out}_{tid}|\mathsf{not\_sent}\text{: } xref]$”
“$[\mathsf{T}^{out}_{tid}|\mathsf{not\_sent}]$”.

Figure 13.17: Verification of Recipient Verifiability using Revocability.

$(\Delta_p, \Delta_t, \Delta_v))$ as defined below offers recipient verifiability. The item I is optimistic if item I' is optimistic.

The behavior of I is defined by the behavior of S, R, *and* T *as follows:*

Protocol “prepareRV“: *On input of “$[\mathsf{S}^{in}_{tid}|\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$” and “$[\mathsf{R}^{in}_{tid}|\mathsf{checkRV}(\mathsf{T}, t_T, xref)]$”,* S *sends $m_1 := \mathrm{sign}_{\mathsf{S}}(\mathsf{R}, d, t_T, xref, tid)$ to* R *who outputs “$[\mathsf{R}^{out}_{tid}|\mathsf{providesRV}\text{: } d]$”.*

Protocol “transferRV“: *On input “$[\mathsf{S}^{in}_{tid}|\mathsf{transferRV}(xref)]$”, the sender sends $xref$ and inputs $[\mathsf{S}'^{in}_{tid}|\mathsf{transferR}(\mathsf{R}, d, \mathsf{T}, t_T + 2, (xref, m_1))]$. If the protocol outputs $[\mathsf{S}'^{out}_{tid}|\mathsf{failed}]$, the sender outputs “$[\mathsf{S}^{out}_{tid}|\mathsf{failed}]$”.*

Upon receipt of $xref$ the recipient inputs $[\mathsf{R}'^{in}_{tid}|\mathsf{receiveR}(\mathsf{T}, t_T + 2, (xref, m_1))]$. On output $[\mathsf{R}'^{out}_{tid}|\mathsf{receivedR}\text{: } d]$, the recipient outputs “$[\mathsf{R}^{out}_{tid}|\mathsf{received}\text{: } xref]$” at time $t_T + \Delta_r$.

Protocol “verifyRV“: *On input “$[\mathsf{R}^{in}_{tid}|\mathsf{showRV}(xref)]$” and “$[\mathsf{T}^{in}_{tid}|\mathsf{verifyRV}(\mathsf{S}, \mathsf{R},$*

$d, xref)]$*", the recipient sends* $m_2 := \mathrm{sign}_{\mathsf{R}}(m_1)$ *to* T*. If this message is received until time* $t_T + 1$*, it is forwarded to* S *who inputs* $[\mathsf{S}'^{in}_{tid'}|\mathsf{revoke}((xref, m_1))]$ *in Round 3. Furthermore,* T *inputs* $[\mathsf{T}'^{in}_{tid'}|\mathsf{revoke}(\mathsf{S}, \mathsf{R}, d, (xref, m_1))]$ *and outputs "*$[\mathsf{T}^{out}_{tid}|\mathsf{not\_sent}]$*".*

Upon output of $[\mathsf{S}'^{out}_{tid'}|\mathsf{revoked}]$ *machine* S *outputs "*$[\mathsf{S}^{out}_{tid}|\mathsf{not\_sent}\colon xref]$*". Machine* R *ignores the output of* $[\mathsf{R}'^{out}_{tid'}|\mathsf{revoked}\colon (xref, m_1)]$.

We define $\Delta_p := 1$*,* $\Delta_t := \Delta_t^R$*, and* $\Delta_v := \Delta_r + 2$. □

Proof. We now show that I fulfills the requirements defined in Definition 12.7:

Correct Execution (R. 12.7a): The inputs "$[\mathsf{S}^{in}_{tid}|\mathsf{prepareRV}(\mathsf{R}, d, \mathsf{T}, t_T, xref)]$" and "$[\mathsf{R}^{in}_{tid}|\mathsf{checkRV}(\mathsf{T}, t_T, xref)]$" always lead to an output "$[\mathsf{R}^{out}_{tid}|\mathsf{providesRV}\colon d]$".

Correct Verification (R. 12.7b): If the recipient output "$[\mathsf{R}^{out}_{tid}|\mathsf{providesRV}\colon d]$", it knows m_1 needed to start "verifyRV", which will always output "$[\mathsf{T}^{out}_{tid}|\mathsf{not\_sent}]$".

Transfer (R. 12.7c): If R outputs "$[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$" at time $t_T + \Delta_r$ and "$[\mathsf{R}^{in}_{tid}|\mathsf{showRV}(xref)]$" is not input, m_2 is not sent and the third party does not input $[\mathsf{T}'^{in}_{tid'}|\mathsf{revoke}(\mathsf{S}, \mathsf{R}, d, (xref, m_1))]$ (note that only the R fixed in m_1 can produce a correct m_2). Therefore, R owns the item d [R. 12.6c].

No Surprises (R. 12.7d): If "$[\mathsf{S}^{in}_{tid}|\mathsf{transferRV}(xref)]$" is not input, then $[\mathsf{S}'^{in}_{tid'}|\mathsf{transferR}(\ldots)]$ is not input and no surprises follows from [R. 12.6d].

If "$[\mathsf{S}^{in}_{tid}|\mathsf{transferRV}(xref)]$" is input and "$[\mathsf{S}^{in}_{tid}|\mathsf{failed}()]$" is output, the underlying transfer output $[\mathsf{S}'^{out}_{tid}|\mathsf{failed}]$ and no surprises follows from [R. 12.6d].

If the third party outputs $[\mathsf{T}'^{out}_{tid'}|\mathsf{not\_sent}]$, then a correct S executed "revoke" before $t_T + 2$. From [R. 12.6b] follows that it output $[\mathsf{T}'^{out}_{tid}|\mathsf{revoked}]$. With [R. 12.6d], this implies that the recipient does not obtain knowledge about the item d.

No Loss (R. 12.7e): Follows similarly from [R. 12.6e].

Termination (R. 12.7f): Follows directly from [R. 12.6f].

During the optimistic phase, the third party is not contacted. Therefore, the simulation is as optimistic as the underlying item.

For the composite machines, $\delta_{RV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ holds, iff $\delta_R(\sigma_{S'}, \sigma_{R'}, d) = \mathsf{true}$ holds for the sub-machines: Let us first assume that $\delta_R(\sigma_{S'}, \sigma_{R'}, d) = \mathsf{true}$. Then, the "transferRV"-protocol output "$[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$" and $\delta_{RV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ holds as well. If we now assume that $\delta_{RV}(\sigma_S, \sigma_R, d) = \mathsf{true}$ holds then the "transferRV"-protocol output "$[\mathsf{R}^{out}_{tid}|\mathsf{received}\colon xref]$" and the underlying "transferR"-protocol output $[\mathsf{R}'^{out}_{tid}|\mathsf{receivedR}\colon d]$, i.e., $\delta_R(\sigma_{S'}, \sigma_{R'}, d) = \mathsf{true}$ holds as well. ■

Remark 13.2. Following this pattern, one should also be able to simulate sender verifiability using generatability directly: Before actually transferring the item, the recipient is first enabled to generate the item with the third party. Then, during "verifySV", the third party enables the recipient to execute "generate" and outputs "$[\mathsf{T}_{tid}^{out}|\mathsf{sent}]$". However, since this does not enable the exchange of new types of transfers with the given fair exchange protocols, we do not elaborate on this efficiency improvement. ◦

Chapter 14

The *SEMPER* Fair Exchange Framework

In Section 11.2, we have sketched the basic concepts of the general *SEMPER* Framework for Electronic Commerce. This framework contains the *SEMPER* Fair Exchange Framework as a central part.

We now describe this Fair Exchange Framework in more detail based on the protocols and definitions presented in the previous chapters. This framework is based on the exchange protocols and the exchange-enabling properties described in the earlier chapters. The design extends the design of the transfer layer as described in [Beil 98, Semp 98].

In Section 14.1, we first describe the static view of the transfer and exchange layer of *SEMPER*, i.e., its class hierarchy and the services of the most important classes. This includes the "wrapping" of the exchange-enabling properties (Section 12.2) and the generic fair exchange protocols (Section 13) into classes. Furthermore, we recall some details of the transfer framework as described in [Semp 98].

Finally, in Section 14.2, we describe the dynamic behavior of the transfer- and fair exchange framework. This is done in two parts: First, we describe what a running exchange looks like, i.e., which objects are instantiated and how they interact during the execution of an exchange protocol. Then, we describe the negotiations and procedure to create such a configuration.

14.1 Class Hierarchy of the Transfer and Exchange Layer

We now describe a static view on the transfer and exchange layer, i.e., we describe all classes and the use of their objects. The interaction among the objects and the execution of protocols will then be described in Section 14.2.

Figure 14.1 shows the main classes of the transfer and exchange layer. In the following subsections, we will explain these classes in more detail: Transactions implementing interactive protocols, attributes for specifying their behavior, descriptions of business items for negotiating and selecting appropriate transfer transactions, and exchange descriptions for negotiating exchange

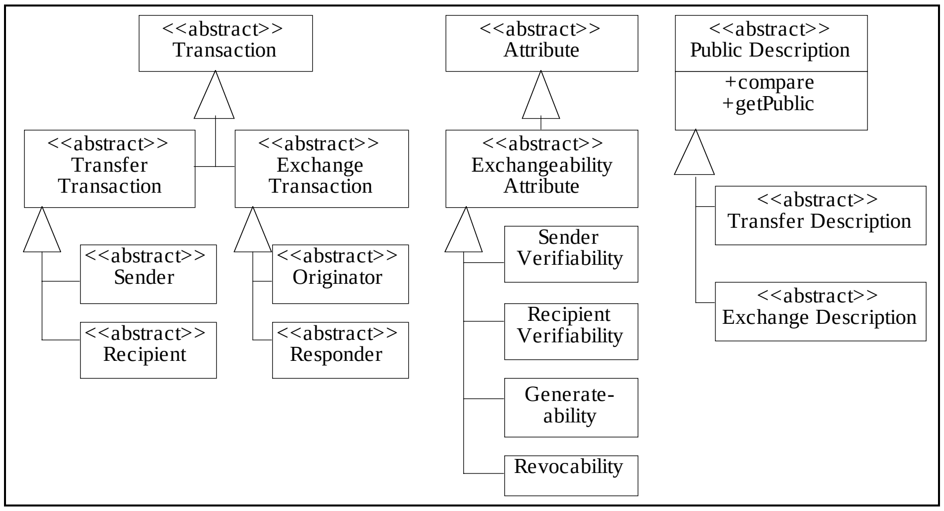


Figure 14.1: Class Hierarchy and Selected Methods of the Transfer and Fair Exchange Layer (written in UML notation; see [FoSc 97]).

protocols.

Note that we only depicted the central methods of our design for fair exchange. In practice, each class needs additional methods and services. The class Transaction, for example, provides methods for persistency and fault tolerance [Beil 98] as well as so-called observer objects [GHJV 95] to observe and control the execution of protocols.

14.1.1 Transactions Implementing Protocols

All protocols are implemented by so-called *transaction role objects*. A particular kind of transaction, such as transfer or exchange, defines the required roles as abstract classes. A protocol is then implemented by one transaction object sub-classing each of the required roles.

Exchange transactions require at least originator and responder role objects. Each optimistic exchange protocol from Section 13 provides an additional object implementing the third party. For instance, the originator of the "S-Verifiable/Generatable" exchange transaction implements machine O of Scheme 13.1 on page 199.

Transfer transaction require at least a sender and a recipient. The transfer transactions depicted in Figure 14.3 define the interfaces of each role of a transfer with exchange-enabling properties. Each exchange-enabling transfer protocol from Section 12.2 provides an additional object implementing the third party. The actual implementors of business items are then required to provide subclasses of these roles that can be instantiated (see [Semp 98, Beil 98] for a more detailed description of the transfer layer).

Note that selecting an appropriate scheme and machine can be a complex task and need not be done by the user of the transfer and exchange layer. These negotiations are explained in detail in Section 14.1.3.

In order to "convert" machines into transaction role objects, we implement each machine of our model (such as S, R, O, or T) in a transaction object sub-

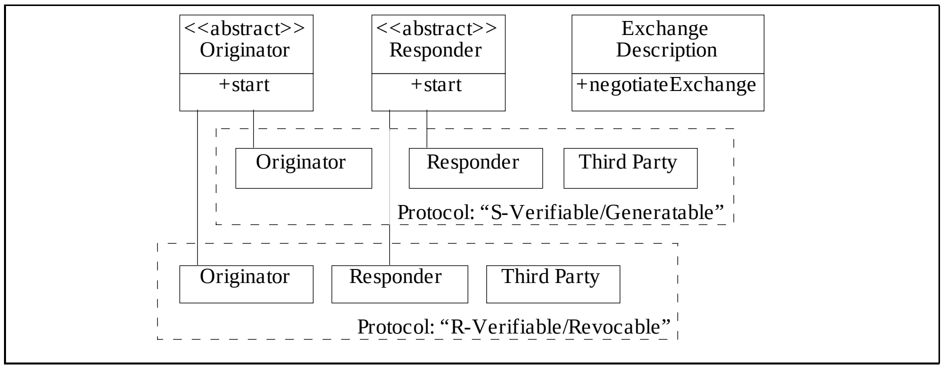


Figure 14.2: Classes of the Exchange Block.

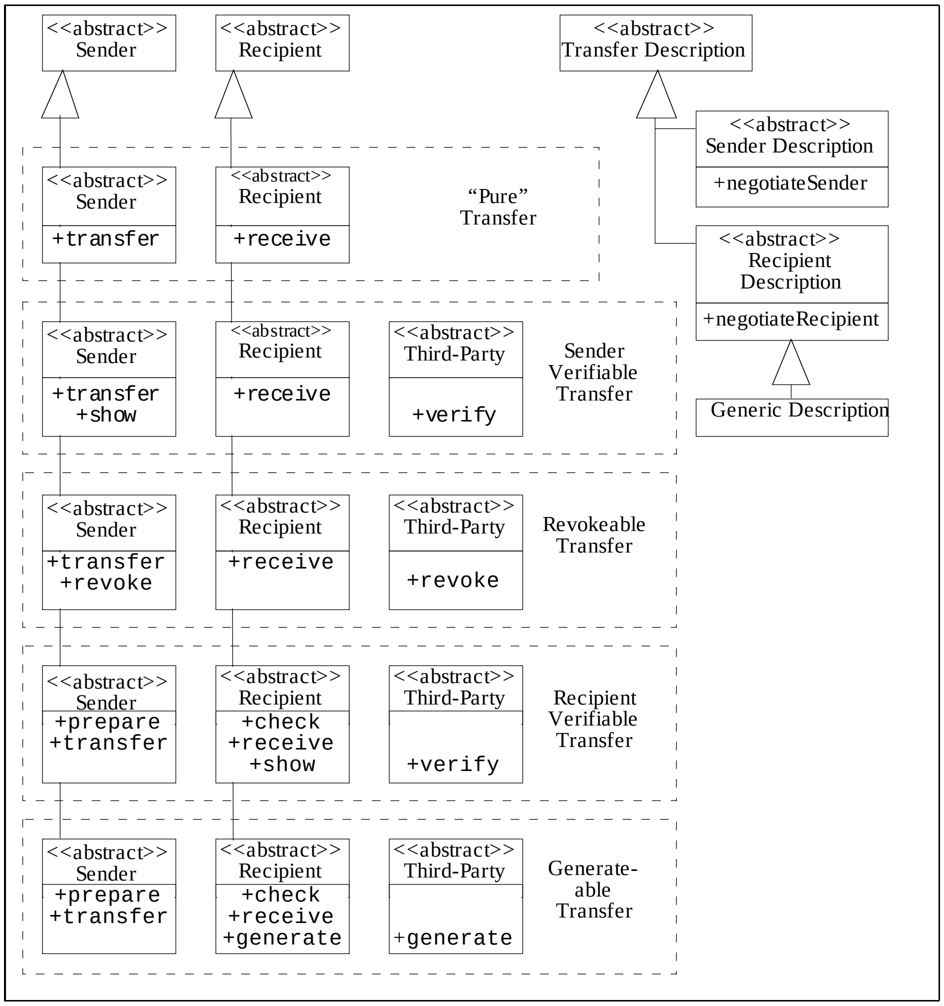


Figure 14.3: Classes of the Transfer Block.

classing the particular role of the corresponding service. The interface of each transaction object directly reflects the functionality of the corresponding machine as defined in Chapter 12: The tid is input when instantiating a transaction. The method names are a prefix of the input command and in principle accept the same arguments than the inputs. An input of, e.g., $[\mathsf{S}_{tid}^{in}|\mathsf{prepareRV}(args)]$ corresponds to calling the method $\mathsf{prepare}(tid, args)$ of the sender object of a transaction with recipient verifiability. The returned result tag and the output parameters correspond to the return parameters of the method that started the protocol[1]. For protocols that do not require an input by a particular machine in our definitions, the design introduces empty methods starting the protocol and returning the defined result (e.g., even though "prepareG" does not take any inputs from the recipient, the corresponding generatability recipient role object provides a method verify() to execute the recipient's side of this protocol and to return the output of the "prepareG" protocol).

14.1.2 Attributes Specifying Security Services

An attribute is an object describing required services of transactions, i.e., inputting an attribute to a negotiation specifies the behavior of the negotiated transaction so that the transaction guarantees the corresponding service defined by this attribute.

Transfer transactions accept attributes for the exchange-enabling properties as well as non-repudiation, i.e., the property that this transaction can later be proven to a judge.

Note that these attributes are not specific to particular kinds of items, i.e., any particular transfer transaction may provide any subset of the exchange-enabling attributes.

14.1.3 Descriptions as Transaction Factories for Negotiating Protocols

For transferring an item, a sender and a recipient transfer-transaction role object is needed, which are able to provide the services corresponding to the desired attributes. For an exchange, an originator and a responder exchange-transaction role object is needed.

The appropriate transaction role objects are negotiated and selected by so-called transaction factories [GHJV 95]: A transaction factory is able to create a transaction object and return it to the caller. The kind of transaction returned (i.e., the protocol and the role to be executed) can be based on the results of a negotiation. Note that these negotiations require interaction with the peer since each player does not have sufficient information on the set-up of the peer. E.g., for paying \$5, one does not know which payment systems exist on the peer's machine.

In our design, descriptions describe protocols and are used as factories.

Negotiating Transfers: Each business item to be transferred is represented by its description. I.e., descriptions of the items are input and output at the inter-

[1]Note that each protocol only produces one output for each party. Otherwise observer-objects may be used to return intermediate results.

face instead of the actual items. The rationale is that most items (i.e., payments, rights, or signatures) are transferred by interactive protocols and therefore, a user cannot input these protocols but only describe them.

The actual items as well as the appropriate protocols for transferring them are handled by the business items layer, i.e., a description is like a reference to one or more actual business items managed by the business items layer.

We distinguish three kinds of descriptions for transfers: A *Sender Description* is a sender's side description of a transfer protocol to be executed. A *Recipient Description* is the recipient's side of an expected transfer. Since both of these descriptions may contain internal information, such as account numbers, about the system that should not be revealed, a *Public Description* is a public version that can be extracted from any description using the getPublic method. This method is defined by the class *Public Description* and is inherited by all other descriptions.

If a recipient has no expectation at all, i.e., accepts any transfer, it may use the so-called *Generic Description*, which is a recipient description.

The factory for the sender role of a transfer transaction is the sender description object of the item to be transferred. By calling the method negotiateSender on input of the recipient name[2] and the required security attributes, an appropriate protocol is selected and a sender-role object is returned.

The recipient uses the method negotiateRecipient of the receiver description of the item it expects while inputting the desired security attributes and, if desired, the name of the expected sender. If it does not expect any particular item, it uses a generic-description object for its negotiation.

More information on transfer transactions can be found in [Semp 98]. A more detailed description of payment negotiations is given in [AsSt 00].

Negotiating Exchanges: The exchange transactions are selected by a so-called exchange description. This description is created using the peer name, a sender description of the item to be sent and a recipient description of the item to be received. Its method negotiateExchange negotiates an appropriate exchange transaction together with its peer Exchange Description. For selecting an appropriate exchange transaction, the exchange description class keeps a static list of attributes for exchange-enabling properties as well as lists of installed exchange transactions together with their required properties.

This negotiation is described in more detail in Section 14.2.2.

14.2 The Transfer and Exchange Layer in Action

We now describe how the transfer and exchange layer works. First, we describe the goal, i.e., what a running exchange protocol looks like. Then, we explain how we get there.

14.2.1 Exchanges in Action

Figure 14.4 shows the set-up, i.e., all instantiated objects, of one party participating in a running exchange: A running exchange transaction exchanges

[2]We assume that the name can internally be mapped to a network address.

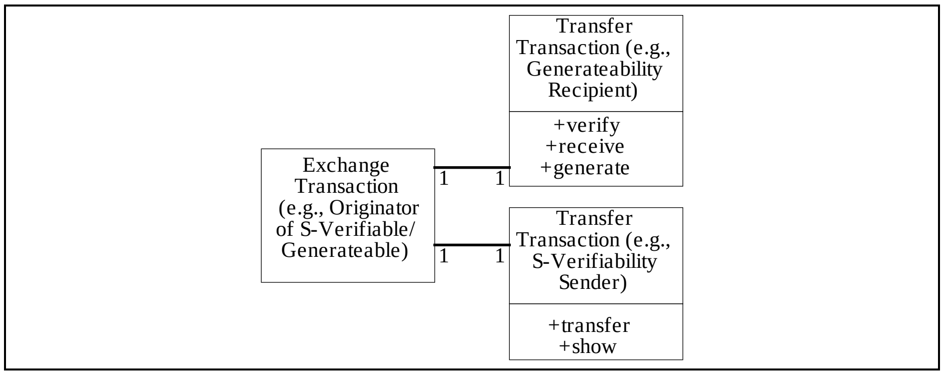

Transfer Transaction (e.g., Generateability Recipient)
+verify
+receive
+generate
Exchange Transaction (e.g., Originator of S-Verifiable/ Generateable)
1
1
1
1
Transfer Transaction (e.g., S-Verifiability Sender)
+transfer
+show

Figure 14.4: Fair Exchange in Action: Objects and their Services in an Ongoing Exchange

two transfers. It sends / receives items by calling the `transfer` and `receive` methods of the transfer transactions. Furthermore, it uses particular methods (such as verify, generate, or show) for accessing the functionality of the exchange-enabling properties.

In the depicted example, the exchange transaction is an originator of a fair exchange of a sender-verifiable for a generatable transfer (see Section 12.2). It interacts with a sending sender-verifiable transfer (i.e., a Sender Transfer Transaction providing Sender Verifiability) and a receiving generatable transfer (i.e., a Recipient Transfer Transaction providing Generatability).

Figures 14.5, 14.6, and 14.7 provide a closer look at how the protocol is actually executed. Figure 14.5 depicts the interaction diagram for the originator of a sender-verifiable / generatable exchange transaction. It consists of the following steps:

1. After the exchange has been started by calling its `start`-method, the exchange transaction first signs an agreement with its peer. This is done by extracting the public descriptions of the input descriptions and sending them in m_1 and m_2. If the players disagree, the originator aborts.

2. Then, the originator's exchange transaction starts the preparation subprotocol for verifying the generatability provided by the transfer received from the responder. This is done by calling the `check`-method of the receiving transfer transaction. Note that here and in the sequel, the input identity of the peer and the description of the item is only used for verification purposes, i.e., to verify whether the agreement messages contain the same identities and descriptions than initially used for negotiating the transfers (see Section 14.2.2).

3. Then, the originator's exchange transaction sends its transfer by calling the `transfer`-method of the sender-verifiable transfer transaction.

4. Then, it receives a transfer by calling the `receive`-method of the generatable transfer.

5. Finally, if the expected item was received, it outputs the description of the item received.

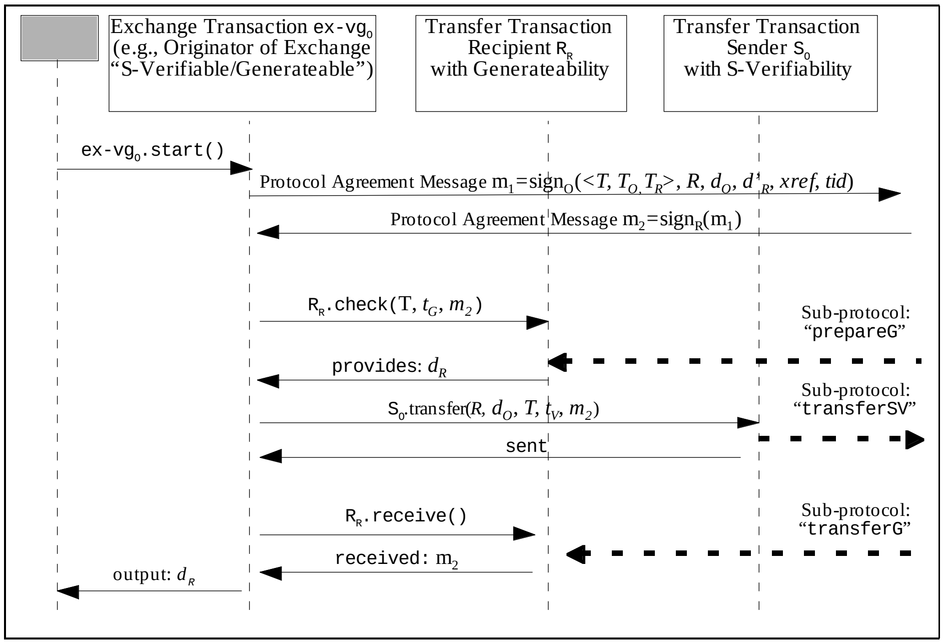

Exchange Transaction ex-vg$_O$ (e.g., Originator of Exchange "S-Verifiable/Generateable")
Transfer Transaction Recipient R$_R$ with Generateability
Transfer Transaction Sender S$_O$ with S-Verifiability
ex-vg$_O$.start()
Protocol Agreement Message m$_1$=sign$_O$(<T, T$_O$, T$_R$>, R, d$_O$, d'$_R$, xref, tid)
Protocol Agreement Message m$_2$=sign$_R$(m$_1$)
R$_R$.check(T, t$_G$, m$_2$)
Sub-protocol: "prepareG"
provides: d$_R$
Sub-protocol: "transferSV"
S$_O$.transfer(R, d$_O$, T, t$_V$, m$_2$)
sent
R$_R$.receive()
Sub-protocol: "transferG"
received: m$_2$
output: d$_R$

Figure 14.5: Fair Exchange: Interaction Diagram for the Originator (See Scheme 13.1 on page 199 for the behavior in detail).

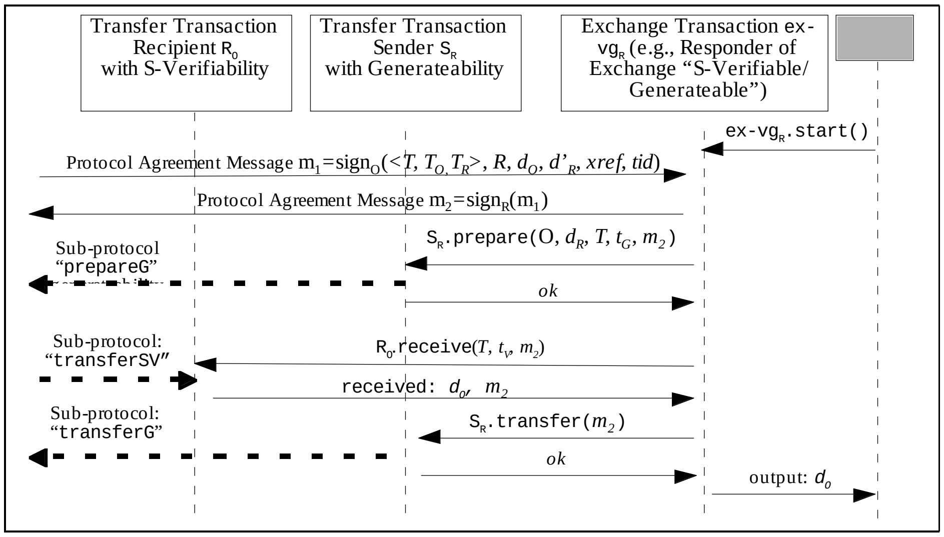

Transfer Transaction Recipient R$_O$ with S-Verifiability
Transfer Transaction Sender S$_R$ with Generateability
Exchange Transaction ex-vg$_R$ (e.g., Responder of Exchange "S-Verifiable/ Generateable")
ex-vg$_R$.start()
Protocol Agreement Message m$_1$=sign$_O$(<T, T$_O$, T$_R$>, R, d$_O$, d'$_R$, xref, tid)
Protocol Agreement Message m$_2$=sign$_R$(m$_1$)
S$_R$.prepare(O, d$_R$, T, t$_G$, m$_2$)
Sub-protocol "prepareG"
ok
Sub-protocol: "transferSV"
R$_O$.receive(T, t$_V$, m$_2$)
received: d$_O$, m$_2$
Sub-protocol: "transferG"
S$_R$.transfer(m$_2$)
ok
output: d$_O$

Figure 14.6: Fair Exchange: Interaction Diagram for the Responder.

The corresponding steps for the responder are depicted in Figure 14.6. If the originator transferred its item in Step 3 without receiving a transfer from the responder in Step 4. In this case, the originator starts the recovery protocol with the third party. This protocol is depicted in Figure 14.7.

The design of the other generic fair exchange protocol follows the same pattern: Again, the protocols described in Section 13 are each implemented by one object for the originator, the responder and the third party, respectively.

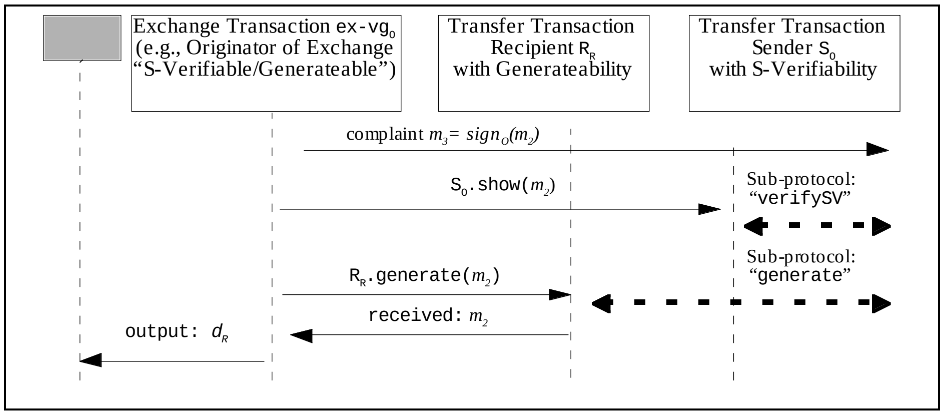

Exchange Transaction ex-vg$_O$ (e.g., Originator of Exchange "S-Verifiable/Generateable")
Transfer Transaction Recipient R$_R$ with Generateability
Transfer Transaction Sender S$_O$ with S-Verifiability
complaint $m_3 = sign_O(m_2)$
S$_O$.show(m_2)
Sub-protocol: "verifySV"
Sub-protocol: "generate"
R$_R$.generate(m_2)
received: m_2
output: d_R

Figure 14.7: Fair Exchange: Originator's view of the Recovery with the Third Party.

14.2.2 Negotiating Exchanges

The fair exchange negotiation is performed by the negotiateExchange method between two fair exchange descriptions containing the peer name, a sender description of the item to be sent, and a recipient description of the item to be received. They output one initialized role-object of an exchange protocol at the end.

In addition to selecting an appropriate exchange protocols and roles, it also negotiates the underlying transfers, i.e., at the end, a fixed exchange protocol together with two transfer protocols have been selected so that the transfer protocols guarantee the required exchange-enabling properties.

The negotiation is based on a list of installed fair exchange protocols and their required properties as known to the exchange description class[3]. Each role of each exchange protocol requires one exchange-enabling property attribute for the sending and the receiving transfer.
The negotiation is nested and works as follows (Figure 14.8):

1. The user starts the negotiation by calling the `negotiateExchange` method of the fair exchange description. The fair exchange description then retrieves the table of registered fair exchange protocols and their required exchangeability attributes.

2. The fair exchange description collects a list of exchange-enabling properties for which exchange protocols exists. It starts by selecting the first attribute in this list.

3. It asks its sender description to negotiate with the peer recipient description whether such a transfer is possible.

   If this is the case, the sender description returns an appropriate transfer transaction sender. If not, it returns a failure and the fair exchange description selects the next attribute in Step 2. If no untried exchangeability attribute exists, the negotiation failed.

[3]In Java, this can be implemented by a static variable of the class, which can be accessed by all its objects.

If the negotiation of a transfer was unsuccessful at the first try, the transfer descriptions may still try to provide the requested property using one of the simulations described in Section 13.3, i.e., the transfer descriptions select a simulation that guarantees the desired property and then start a negotiation of this simulation (i.e., the simulation again negotiates with its peer simulation whether a transfer with its required property can be provided directly). If this succeeds, the negotiated simulating transfer transaction role objects are returned.

4. Given the attribute of the first transfer, the fair exchange description checks for what other exchangeability attributes exchange protocols exist. It selects one of them.

5. It asks the input recipient description to negotiate with the peer sender description whether such a transfer is possible.

   If this is the case, the sender description returns an appropriate transfer transaction. If not, it returns a failure and the fair exchange description selects the next attribute in Step 4.

   If the fair exchange description failed to negotiate any transfer recipient with an exchangeability attribute for which an exchange protocol exists, it selects the next attribute for the first transfer, i.e., goes back to Step 2.

6. Then, it instantiates an appropriate role-object of the selected fair exchange transaction. During this instantiation, it inputs references to both transfers, the addresses of the players, and the recipient descriptions into the fair exchange protocol.

7. The fair exchange description returns the fair exchange transaction role which is ready to run, i.e., it will execute the actual exchange when both users call the start() method.

Remark 14.1. In Steps 2, 4 the fair exchange has the choice which property or protocol to try first. Therefore, the sequence of properties and protocols can be set by the preferences input by the human user.

Remark 14.2. Steps 3 and 4 can be performed in parallel, i.e., the exchange description negotiates pairs of exchange-enabled transfers for a particular exchange in parallel. ○

14.3 Extending the Transfer and Exchange Layer

14.3.1 Transfer Block

In order to add a new transferable item, one has to implement the item, its sender and recipient descriptions, and appropriate transfer transactions. If one wants to exchange the item fairly, some of the provided transfer transactions must be able to guarantee exchange-enabling properties. For payments, for example, the implementor would be required to add a new purse, payment items, payment sender and recipient descriptions as well as a payment transaction which, e.g., implements generatability.

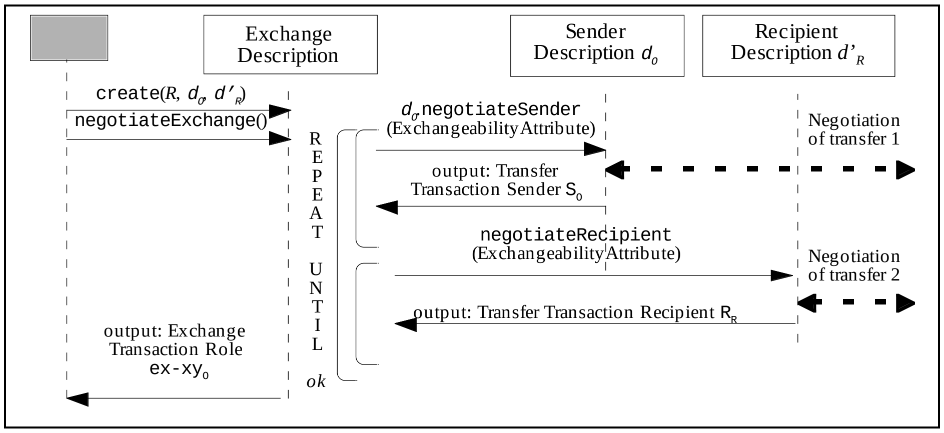

Exchange Description
Sender Description d_o
Recipient Description d'_R
create(R, d_σ, d'_R)
negotiateExchange()
REPEAT UNTIL ok
d_σ.negotiateSender (ExchangeabilityAttribute)
Negotiation of transfer 1
output: Transfer Transaction Sender S_o
negotiateRecipient (ExchangeabilityAttribute)
Negotiation of transfer 2
output: Transfer Transaction Recipient R_R
output: Exchange Transaction Role ex-xy$_o$

Figure 14.8: Fair Exchange: Negotiating an Exchange Protocol for the Originator.

14.3.2 Exchange Block

For the given set of properties, the current design supplies two protocols, which, given the current set of exchange-enabling properties and the given set of simulations, are sufficient to exchange any two items fairly. However, in practice it may be desirable to install more efficient versions of protocols for these properties (i.e., without requiring simulations). In order to do this, one may install new fair exchange transactions for each role and register them at the exchange description class[4]. In the preferences, one should tell the exchange description to prefer this protocol compared to the old protocol. Naturally, the protocol is only executed if the peer installed it, too.

The more interesting case, however, is to extend the exchange layer in a more general way. An example of such an extension would be to add gradual fair exchange protocols which do not guarantee deterministic fairness but which also do not need a third party. This can be done as follows: Again, one defines new properties of transfers and implements the appropriate transfer transactions for the items that shall be exchanged using the new protocol (such as micro-transferable for probabilistically fair gradual exchange protocols). Finally, one has to provide fair exchange protocols for these new properties or a mixture between new and old properties.

This mechanism for extending the transfer and exchange layer should also work for particular instances of fair exchange. In this case, one would implement specific exchange transactions for the appropriate items. Furthermore, one would either tell the exchange description that if both items are of the exchangeable type (e.g., signatures for contract signing), it may select the contract signing protocol. An alternative would be to define a new property "contractable" which is only provided by the particular items needed by this contract signing protocol and which then leads to the selection of this particular exchange protocol by the exchange description.

Note that this should not require changes to the fair exchange description: After installation of the new property attributes and protocols, the fair exchange description considers and selects them during negotiation if appro-

[4]Recall, the registered transactions are kept in a static variable of the class.

priate.

Chapter 15

Conclusion and Outlook

In this part, we have described synchronous protocols and a general design of a framework for generic and optimistic two-party fair exchange.

Our protocols have been designed for the synchronous network model commonly used in cryptography. However, for increased availability and robustness it is desirable to extend the protocols towards asynchronous networks. Even though we feel that few fundamental changes are required, our exchange-enabling properties as well as our generic fair exchange protocols need to be refined. This does not require an extension of the given framework since asynchronous as well as synchronous services can already co-exist.

In our definitions of the exchange-enabling properties, we describe integrity requirements by means of listing required protocols as well as requirements on their user in- and outputs (e.g., in protocol X, an input Y should result in an output Z). Unfortunately, this approach is insufficient for secrecy of business items: Here, we are required to formalize what it means that an incorrect party does not obtain any useful additional information about the item. One approach towards a formalization of these secrecy requirements was pursued in Part II by comparing a given implementation with the trusted host specifying the intended service. Even though this formalization is useful for evaluating particular protocols, it cannot easily be applied to our generic notion of transfers: Since trusted-host-based specifications tend to over-specify the behavior of a system, we would be required to specify one trusted host for each kind of items. Without a notion of action refinement, building a trusted host for substantially different items is still an open problem. Therefore, a large number of trusted hosts is needed, which again leads to a large number of exchange protocols. As a consequence of aiming at a design rather than a formal model, we therefore omitted a formalization of the secrecy requirements. Promising future approaches towards formalizing the privacy requirements seem to be:

- A notion of action refinement for trusted hosts may enable more generic trusted-host-based specifications.
- A specific semantics for "not gaining knowledge about items" may be built based on the "use"-protocols formalizing the use of an item.
- The exchange-enabling properties may later be defined in relation to the

"pure" transfer, i.e., one may formalize the notion that with the additional protocols for exchange-enabling properties, the item is as secure as without them.

The used two-party approach towards fair exchange should cover most of today's need for fair exchange in electronic commerce. In the future, multi-party fair exchanges, such as reliable and certified broadcast, may be needed as well [Waid6 98]. Designing such protocols should be straightforward by combining the ideas of transfer-based exchanges with existing protocols for multi-party fair exchange [ABSW 98, ABSW 99].

Up to now, we only implemented the transfer layer and an exchange prototype. An implementation of the complete framework will lead to a more thorough validation as well as further refinements and extensions. One aspect of such an implementation is to adapt additional business items based on existing products to provide exchange-enabling transfers.

Bibliography

[Abad 98] Martín Abadi: Protection in Programming-Language Translations; 25th International Colloquium on Automata, Languages and Programming (ICALP), LNCS 1443, Springer-Verlag, Berlin 1998, 868-883.

[ABPP 00] N. Asokan, Birgit Baum-Waidner, Torben P. Pedersen, Birgit Pfitzmann, Matthias Schunter, Michael Steiner, Michael Waidner: Architecture; Chapter 6, pp. 45-64 of [Semp 00].

[ABSW 98] N. Asokan, Birgit Baum-Waidner, Matthias Schunter, Michael Waidner: Optimistic Synchronous Multi-Party Contract Signing; IBM Research Report RZ 3089 (#93135) 12 / 14 / 1998, IBM Research Division, Zürich, Dec. 1998.

[ABSW 99] N. Asokan, Birgit Baum-Waidner, Matthias Schunter, Michael Waidner: Optimistische Mehrparteienvertragsunterzeichnung; Verläßliche IT-Systeme, GI-Fachtagung VIS '99, DuD Fachbeiträge, Vieweg, Braunschweig 1999, 49-66.

[AJSW 97] N. Asokan, Phillipe A. Janson, Michael Steiner, Michael Waidner: The State of the Art in Electronic Payment Systems; IEEE Computer 30 / 9 (1997) 28-35.

[AnMS 96] Ross Anderson, Charalampos Manifavas, Chris Sutherland: NetCard - A Practical Electronic Cash System; International Security Protocols Workshop 1996, LNCS 1189, Springer-Verlag, Berlin 1997, 49-58.

[AnNe1 95] Ross Anderson, Roger Needham: Robustness Principles for Public Key Protocols; Crypto '95, LNCS 963, Springer-Verlag, Berlin 1995, 236-247.

[Asok 98] N. Asokan: Fairness in Electronic Commerce; PhD Thesis, University of Waterloo, 1998.

[AsSt 00] N. Asokan, Michael Steiner: Example of a Block Design — The Payment Framework; Chapter 11, pp. 185-212 of [Semp 00].

[AsSW 97] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange; 4th ACM Conference on Computer and Communications Security, Zürich, April 1997, 6-17.

[AsSW 98] N. Asokan, Victor Shoup, Michael Waidner: Asynchronous Protocols for Optimistic Fair Exchange; 1998 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Los Alamitos 1998, 86-99.

[AsSW1 98] N. Asokan, Victor Shoup, Michael Waidner: Optimistic Fair Exchange of Digital Signatures; Eurocrypt '98, LNCS 1403, Springer-Verlag, Berlin 1998, 591-606.

[AsSW2 96] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Multi-Party Fair Exchange; IBM Research Report RZ 2892, IBM Zurich Research Laboratory, Zürich, November 1996.

[AsSW3 96] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange; IBM Research Report RZ 2858, IBM Zurich Research Laboratory, Zürich, February 1996.

[BaDM 98] Feng Bao, Robert Deng, Wenbo Mao: Efficient and Practical Fair Exchange Protocols with Off-Line TTP; 1998 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Los Alamitos 1998, 77-85.

[BaTy 94] Alireza Bahreman, J. D. Tygar: Certified Electronic Mail; 1994 Symposium on Network and Distributed Systems Security (NDSS 94), Internet Society, IEEE Press, 1994.

[BaWa 98] Birgit Baum-Waidner, Michael Waidner: Optimistic Asynchronous Multi-Party Contract Signing; IBM Research Report RZ 3078 (#93124) 11/23/1998, IBM Research Division, Zürich, Nov. 1998.

[BBCM1 94] Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Mjølsnes, Frank Muller, Torben Pedersen, Birgit Pfitzmann, Peter de Rooij, Berry Schoenmakers, Matthias Schunter, Luc Vallée, Michael Waidner: The ESPRIT Project CAFE - High Security Digital Payment Systems; ESORICS 94 (Third European Symposium on Research in Computer Security), Brighton, LNCS 875, Springer-Verlag, Berlin 1994, 217-230.

[BCDG 88] Ernest F. Brickell, David Chaum, Ivan B. Damgård, Jeroen van de Graaf: Gradual and verifiable release of a secret; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 156-166.

[BDPR 98] Mihir Bellare, Anand Desai, David Pointcheval, Phillip Rogaway: Relations Among Notions of Security for Public-Key Encryption Schemes; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 26-45.

[Beav 96] Donald Beaver: Equivocable oblivious transfer; Eurocrypt '96, LNCS 1070, Springer-Verlag, Berlin 1996, 119-130.

[Beav5 91] Donald Beaver: Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority; Journal of Cryptology 4/2 (1991) 75-122.

[BeCK1 98] Mihir Bellare, Ran Canetti, Hugo Krawczyk: A modular approach to the design and analysis of authentication and key exchange protocols; 13th Symposium on Theory of Computing (STOC) 1998, ACM, New York 1998, 419-428.

[Beil 98] Thomas Beiler: Design einer Architektur zum Austausch elektronischer Güter; Diplomarbeit, Institut für Informatik, Universität Hildesheim, Februar 1998.

[BGHH 95] Mihir Bellare, Juan A. Garay, Ralf Hauser, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, Michael Waidner: iKP - A Family of Secure Electronic Payment Protocols; 1st USENIX Workshop on Electronic Commerce, 1995, 89-106.

[BGMR 85] Michael Ben-Or, Oded Goldreich, Silvio Micali, Ronald L. Rivest: A Fair Protocol for Signing Contracts; 12th International Colloquium on Automata, Languages and Programming (ICALP), LNCS 194, Springer-Verlag, Berlin 1985, 43-52.

[BGMR 90] Michael Ben-Or, Oded Goldreich, Silvio Micali, Ronald L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36 / 1 (1990) 40-46.

[BlMi 82] Manuel Blum, Silvio Micali: How To Generate Cryptographically Strong Sequences Of Pseudo Random Bits; 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 112-117.

[Blu1 83] Manuel Blum: Coin Flipping By Telephone, A Protocol for Solving Impossible Problems; ACM SIGACT News 15 / 1 (1983) 23-27.

[Blu2 83] Manuel Blum: How to Exchange (Secret) Keys; ACM Transactions on Computer Systems 1 / 2 (1983) 175-193.

[Blum 81] Manuel Blum: Three Applications of the Oblivious Transfer; Version 2: September 18, 1981, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkley, Ca. 94720.

[Blum 83] Manuel Blum: How to exchange (secret) keys; 15th Symposium on Theory of Computing (STOC) 1983, ACM, New York 1983, 440-447.

[BlVV 84] Manuel Blum, Umesh V. Vazirani, Vijay V. Vazirani: Reducibility Among Protocols; Crypto ’83, Plenum Press, New York 1984, 137-146.

[BoCP 88] Jurjen Bos, David Chaum, George Purdy: A Voting Scheme; unpublished manuscript, presented at the rump session of Crypto ’88.

[BrCC 88] Gilles Brassard, David Chaum, Claude Crépeau: Minimum Disclosure Proofs of Knowledge; Journal of Computer and System Sciences 37 (1988) 156-189.

[BrDo 84] Andrei Z. Broder, Danny Dolev: Flipping coins in many pockets (Byzantine agreement on uniformly random values); 25th Symposium on Foundations of Computer Science (FOCS) 1984, IEEE Computer Society, 1984, 157-170.

[BüPf 89] Holger Bürk, Andreas Pfitzmann: Digital Payment Systems Enabling Security and Unobservability; Computers & Security 8/5 (1989) 399-416.

[BüPf 90] Holger Bürk, Andreas Pfitzmann: Value Exchange Systems Enabling Security and Unobservability; Computers & Security 9/8 (1990) 715-721.

[Camp 96] Linda Jean Camp: Privacy and Reliability in Internet Commerce; Ph. D. thesis, Carnegie Mellon University, Pittsburgh, August 1996.

[Cane 96] Ran Canetti: Studies in Secure Multiparty Computation and Applications; Thesis, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1995, revised March 1996.

[Cane 98] Ran Canetti: Security and Composition of Multi-party Cryptographic Protocols; Theory of Cryptography Library 98-18, June 1998, last revision September 1999.

[CaGo 99] Ran Canetti, Shafi Goldwasser: An Efficient Threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack; Eurocrypt '99, LNCS 1592, Springer-Verlag, Berlin 1999, 90-106.

[CGMA 85] Benny Chor, Shafi Goldwasser, Silvio Micali, Baruch Awerbuch: Verifiable secret sharing and achieving simultaneity in the presence of faults; 26th Symposium on Foundations of Computer Science (FOCS) 1985, IEEE Computer Society, 1985, 383-395.

[Chau 89] David Chaum: Privacy Protected Payments - Unconditional Payer and/or Payee Untraceability; SMART CARD 2000: The Future of IC Cards, IFIP WG 11.6 International Conference; Laxenburg (Austria) 1987, North-Holland, Amsterdam 1989, 69-93.

[ChDw 89] Benny Chor, Cynthia Dwork: Randomization in Byzantine Agreement; JAI Press, Advances in Computing Research Vol. 5, Greenwich (Connecticut) 1989, 443-497.

[ChHP 92] David Chaum, Eugène van Heijst, Birgit Pfitzmann: Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 470-484.

[ChPe1 93] David Chaum, Torben Pryds Pedersen: Wallet Databases with Observers; Crypto '92, LNCS 740, Springer-Verlag, Berlin 1993, 89-105.

[CHTY1 96] Jean Camp, Michael Harkavy, J. D. Tygar, Bennet Yee: Anonymous Atomic Transactions; 2nd USENIX Workshop on Electronic Commerce, 1996, 123-134.

[Clev 90] Richard Cleve: Controlled gradual disclosure schemes for random bits and their applications; Crypto '89, LNCS 435, Springer-Verlag, Berlin 1990, 573-588.

[CoSa 96] Tom Coffey, Puneet Saidha: Non-Repudiation with Mandatory Proof of Receipt; Computer Communication Review 26/1 (1996) 6-17.

[CoTS 95] Benjamin Cox, J. D. Tygar, Marvin Sirbu: NetBill Security and Transaction Protocol; 1st USENIX Workshop on Electronic Commerce, 1995, 77-88.

[CrSh1 98] Ronald Cramer, Victor Shoup: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 13-25.

[Damg 88] Ivan Bjerre Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

[Damg 94] Ivan Bjerre Damgård: Practical and Provably Secure Release of a Secret and Exchange of Signatures; Eurocrypt '93, LNCS 765, Springer-Verlag, Berlin 1994, 200-217.

[Damg 95] Ivan Bjerre Damgård: Practical and Provably Secure Release of a Secret and Exchange of Signatures; Journal of Cryptology 8/4 (1995) 201-222.

[DGLW 96] Robert H. Deng, Li Gong, Aurel A. Lazar, Weiguo Wang: Practical Protocols for Certified Electronic Mail; Journal of Network and Systems Management 4/3 (1996) 279-297.

[DiHe 76] Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

[DoDN 91] Danny Dolev, Cynthia Dwork, Moni Naor: Non-Malleable Cryptography; 23rd Symposium on Theory of Computing (STOC) 1991, ACM, New York 1991, 542-552.

[DoSc 98] Assia Doudou, André Schiper: Muteness Detectors for Consensus with Byzantine Processes; 17th Symposium on Principles of Distributed Computing (PODC), ACM, New York 1998, 315.

[DoSc1 98] Assia Doudou, André Schiper: Muteness failure detectors for consensus with byzantine processes; Technical Report 97/230, École Polytechnique Fédérale de Lausanne, Switzerland, October 1997.

[Even 83] Shimon Even: A Protocol for Signing Contracts; ACM SIGACT News 15/1 (1983) 34-39.

[EvGL 83] Shimon Even, Oded Goldreich, Abraham Lempel: A randomized protocol for signing contracts; Crypto '82, Plenum Press, New York 1983, 205-210.

[EvGL 85] Shimon Even, Oded Goldreich, Abraham Lempel: A Randomized Protocol for Signing Contracts; Communications of the ACM 28/6 (1985) 637-647.

[EvYa 80] Shimon Even, Yacov Yacobi: Relations Among Public Key Signature Systems; Technical Report Nr. 175, March 1980, Computer Science Department, Technion, Haifa, Israel.

[FiHM 99] Matthias Fitzi, Martin Hirt, Ueli Maurer: General Adversaries in Unconditional Multi-party Computation; Asiacrypt '99, LNCS 1716, Springer-Verlag, Berlin 1999, 232-246.

[FoSc 97] Martin Fowler, Kendall Scott: UML Distilled - Applying the Standard Object Modeling Language; Addison-Wesley, Reading 1997.

[FrRe 97] Matthew K. Franklin, Michael K. Reiter: Fair Exchange with a Semi-Trusted Third Party; 4th ACM Conference on Computer and Communications Security, Zürich, April 1997, ACM Press, New York 1997, 1-5.

[FrTs 98] Matt Franklin, Gene Tsudik: Secure Group Barter: Multi-party Fair Exchange with Semi-Trusted Neutral Parties; 2nd International Conference on Financial Cryptography (FC '98), LNCS 1465, Springer-Verlag, Berlin 1998, 90-102.

[GaJM 99] Juan Garay, Markus Jakobsson, Philip MacKenzie: Abuse-Free Optimistic Contract Signing; Crypto '99, LNCS 1666, Springer-Verlag, Berlin 1999, 449-484.

[GeMi 95] Rosario Gennaro, Silvio Micali: Verifiable Secret Sharing as Secure Computation; Eurocrypt '95, LNCS 921, Springer-Verlag, Berlin 1995, 168-182.

[GHJV 95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns — Elements of Object-Oriented Software; Addison-Wesley-Longman, Reading 1995.

[GMW 87] Oded Goldreich, Silvio Micali, Avi Wigderson: How to play any mental game — or — a completeness theorem for protocols with honest majority; 19th Symposium on Theory of Computing (STOC) 1987, ACM, New York 1987, 218-229.

[Gol1 84] Oded Goldreich: Sending Certified Mail using Oblivious Transfer and a Threshold Scheme; Technion - Israel Institute of Technology, Computer Science Department, Technical Report, 1984.

[Gol4 83] Oded Goldreich: An Approximation Protocol for Signing Contracts; Technion - Israel Institute of Technology, Department of Computer Science, Haifa, Israel, Technical Report, 1983.

[Gold 82] Oded Goldreich: A Protocol for Sending Certified Mail; Technion - Israel Institute of Technology, Computer Science Department, Technical Report, 1982.

[Gold 83] Oded Goldreich: A simple protocol for signing contracts; Crypto '83, Plenum Press, New York 1984, 133-136.

[Gold 93] Oded Goldreich: A Uniform-Complexity Treatment of Encryption and Zero-Knowledge; Journal of Cryptology 6 / 1 (1993) 21-53.

[Gold 98] Oded Goldreich: Secure Multi-Party Computation; Working Draft, Version June 11, 1998, available from the author's home page.

[Gold 99] Oded Goldreich: Modern Cryptography, Probabilistic Proofs and Pseudo-randomness; Algorithms and Combinatorics 17, Springer - Verlag, Berlin Heidelberg 1999.

[Gold1 95] Oded Goldreich: Foundations of Cryptography (Fragments of a Book); Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, February 23, 1995 (from `http://www.wisdom.weizmann.ac.il/~oded`).

[GoLe 91] Shafi Goldwasser, Leonid Levin: Fair Computation of General Functions in Presence of Immoral Majority; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 77-93.

[GoMi 82] Shafi Goldwasser, Silvio Micali: Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information; 14th Symposium on Theory of Computing (STOC) 1982, ACM, New York 1982, 365-377.

[GoMi 84] Shafi Goldwasser, Silvio Micali: Probabilistic Encryption; Journal of Computer and System Sciences 28 (1984) 270-299.

[GoMR 88] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17 / 2 (1988) 281-308.

[GoMR 89] Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof Systems; SIAM Journal on Computing 18 / 1 (1989) 186-207.

[HaLi1 93] Lein Harn, Hung-Yu Lin: An Oblivious Transfer Protocol and Its Application for the Exchange of Secrets; Asiacrypt '91, LNCS 739, Springer-Verlag, Berlin 1993, 312-320.

[Han 96] Yongfei Han: Investigations of Non-Repudiation Protocols; First Australasian Conference on Information Security and Privacy (ACISP '96), LNCS 1172, Springer-Verlag, Berlin 1996, 38-45.

[HaSh 85] Johan Håstad, Adi Shamir: The Cryptographic Security of Truncated Linearly Related Variables; 17th Symposium on Theory of Computing (STOC) 1985, ACM, New York 1985, 356-362.

[HaSW 96] Ralf Hauser, Michael Steiner, Michael Waidner: Micro-payments based on iKP; IBM Research Report RZ 2791 (#89269) 02 / 12 / 96, IBM Research Division, Zürich, Feb. 1996. Also appeared at SECURICOM 96, 14th Worldwide Congress on Computer and Communications Security and Protection, Jun. 5-6, 1996, Paris, 67-82.

[HaTs 96] Ralf Hauser, Gene Tsudik: On Shopping Incognito; 2nd USENIX Workshop on Electronic Commerce, 1996, 251-258.

[Herd2 95] Siegfried Herda: Non-repudiation: Constituting evidence and proof in digital cooperation; Computer Standards & Interfaces 17 (1995) 69-79.

[HiMa 97] Martin Hirt, Ueli Maurer: Player Simulation and General Adversary Structures in Perfect Multi-Party Computation; Manuskript, Swiss Federal Institute of Technology (ETH), Zurich 1998.

[ISO13888-1 97] ISO / IEC: Information technology - Security techniques - Non repudiation - Part 1: General; ISO / IEC International Standart 13888-1, 1st Edition, 12.01.1997.

[ISO13888-2 98] ISO / IEC JTC 1 / SC 27, N1106: Information technology - Security techniques - Non-repudiation - Part 2: Mechanisms using symmetric techniques; ISO / IEC International Standard 13888-2, first edition, 1998.

[ISO13888-3 97] ISO / IEC: Information technology - Security techniques - Non-repudiation Part 3: Mechanisms using asymmetric techniques; ISO / IEC International Standard 13888-1, 1st Edition, 12.01.1997.

[Jako1 95] Markus Jakobsson: Ripping Coins for a Fair Exchange; Eurocrypt '95, LNCS 921, Springer-Verlag, Berlin 1995, 220-230.

[KeGa 96] Steven P. Ketchpel, Hector Garcia-Molina: Making Trust Explicit in Distributed Commerce Transactions; 16th International Conference on Distributed Computing Systems, 1996, IEEE Computer Society, 1996, 270-281.

[Lamp 78] Leslie Lamport: Time, Clocks, and the Ordering of Events in a Distributed System; Communications of the ACM 21 / 7 (1978) 558-565.

[LMWF 94] Nancy Lynch, Michael Merritt, William Weihl, Alan Fekete: Atomic Transactions; Morgan Kaufmann, San Mateo 1994.

[LuMR 83] Michael Luby, Silvio Micali, Charles Rackoff: How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin; 24th Symposium on Foundations of Computer Science (FOCS) 1983, IEEE Computer Society, 1983, 11-21.

[Lync 96] Nancy A. Lynch: Distributed Algorithms; Morgan Kaufmann, San Francisco 1996.

[Mana1 95] Mark S. Manasse: The Millicent Protocols for Electronic Commerce; 1st USENIX Workshop on Electronic Commerce, 1995, 117-124.

[Merk 78] Ralph C. Merkle: Secure Communication Over Insecure Channels; Communications of the ACM 21 / 4 (1978) 294-299.

[Mica 97] Silvio Micali: Certified E-Mail with Invisible Post Offices - or - A Low-Cost, Low-Congestion, and Low-Liability Certified E-Mail System; presented at RSA 97; sent by S. Micali to Asokan and M. Schunter, May 2, 1997.

[Mica1 97] Silvio Micali: Simultaneous Electronic Transactions with Visible Trusted Parties; United States Patent, Patent Number 5,629,982, Date of Patent May 13, 1997.

[Mica2 97] Silvio Micali: Simultaneous Electronic Transactions; United States Patent, Patent Number 5,666,420, Date of Patent Sep 9, 1997.

[MiRo 92] Silvio Micali, Phillip Rogaway: Secure Computation; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 392-404.

[OkOh 94] Tatsuaki Okamoto, Kazuo Ohta: How to Simultaneously Exchange Secrets by General Assumptions; 2nd ACM Conference on Computer and Communications Security, Fairfax, November 1994, ACM Press, New York 1994, 184-192.

[Oser 99] Philipp H. Oser: A Secure Service Platform for Electronic Commerce; Diploma Thesis, École Polytechnique Fédérale de Lausanne, February 1999 (Advisors: Sebastian Staamann, Allan Coignet).

[Pede 92] Torben Pryds Pedersen: Non-Interactive and Information-Theoretically Secure Verifiable Secret Sharing; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 129-140.

[Pede 95] Torben P. Pedersen: Electronic Payments of Small Amounts; DAIMI PB-495, Computer Science Department Aarhus University, August 1995.

[Pede 97] Torben P. Pedersen: Electronic Payments of Small Amounts; Security Protocols 1996, LNCS 1189, Springer-Verlag, Berlin 1997, 59-68.

[PeSL 80] Marshall Pease, Robert Shostak, Leslie Lamport: Reaching Agreement in the Presence of Faults; Journal of the ACM 27/2 (1980) 228-234.

[Pfit4 93] Birgit Pfitzmann: Sorting Out Signature Schemes; 1st ACM Conference on Computer and Communications Security, Fairfax, November 1993, ACM Press, New York 1993, 74-85.

[PfSW 00] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Cryptographic Security of Reactive Systems; DERA/RHUL Workshop on Secure Architectures and Information Flow, Electronic Notes in Theoretical Computer Science (ENTCS), March 2000, http://www.elsevier.nl/locate/entcs/volume32.html.

[PfSW 98] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Optimal Efficiency of Optimistic Contract Signing; 17th Symposium on Principles of Distributed Computing (PODC), ACM, New York 1998, 113-122.

[PfSW1 00] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Secure Reactive Systems; IBM Research Report RZ 3206 (#93252) 02 / 14 / 2000, IBM Research Division, Zürich, May 2000.

[PfSW1 98] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Optimal Efficiency of Optimistic Contract Signing; IBM Research Report RZ 2994 (#93040) 20 / 04 / 98, IBM Research Division, Zürich, April 1998.

[PfSW2 00] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Provably Secure Certified Mail; IBM Research Report RZ 3207 (#93253) 02 / 14 / 2000, IBM Research Division, Zürich, ??.

[Pfit8 96] Birgit Pfitzmann: Digital Signature Schemes - General Framework and Fail-Stop Signatures; LNCS 1100, Springer-Verlag, Berlin 1996.

[PfWa 00] Birgit Pfitzmann, Michael Waidner: Composition and Integrity Preservation of Secure Reactive Systems; IBM Research Report RZ 3234 (#93280) 06 / 12 / 00, IBM Research Division, Zürich, June 2000; accepted for 7th ACM Conference on Computer and Communications Security, Athens, November 2000.

[PfWa 94] Birgit Pfitzmann, Michael Waidner: A General Framework for Formal Notions of "Secure" System; Hildesheimer Informatik-Berichte 11 / 94, ISSN 0941-3014, Institut für Informatik, Universität Hildesheim, April 1994.

[PfWa3 98] Birgit Pfitzmann, Michael Waidner: Extensions to Multi-Party Computations; Slides, presented at: The 1998 Weizmann Workshop on Cryptography, June 16-18th, Weizmann Institute of Science, Rehovot, Israel.

[PWP 90] Birgit Pfitzmann, Michael Waidner, Andreas Pfitzmann: Rechtssicherheit trotz Anonymität in offenen digitalen Systemen; Datenschutz und Datensicherung DuD 14 / 5-6 (1990) 243-253, 305-315.

[Rab1 83] Michael O. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27 / (1983) 256-267.

[Rabi 81] Michael O. Rabin: Transaction Protection by Beacons; Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138, Technical Report TR-29-81, November 1981.

[Rabi 83] Michael O. Rabin: Randomized Byzantine Generals; 24th Symposium on Foundations of Computer Science (FOCS) 1983, IEEE Computer Society, 1983, 403-409.

[RaSi 92] Charles Rackoff, Daniel R. Simon: Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 433-444.

[RiSh 97] Ronald L. Rivest, Adi Shamir: Payword and MicroMint: Two simple micropayment schemes; Security Protocols 1996, LNCS 1189, Springer-Verlag, Berlin 1997, 69-88.

[RoPa 93] Kurt Rothermel, Stefan Pappe: Open Commit Protocols Tolerating Commission Failures; ACM Transactions on Database Systems 18 / 2 (1993) 289-332.

[RSA 78] Ronald L. Rivest, Adi Shamir, Leonard Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21 / 2 (1978) 120-126, reprinted: 26 / 1 (1983) 96-99.

[Schn3 98] Steve Schneider: Formal analysis of a non-repudiation protocol; 11th Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos 1998, 54-65.

[Schu 00] Matthias Schunter: Fair Exchange — A New Paradigm for Electronic Commerce; Chapter 10, pp. 155-284 of [Semp 00].

[ScWW 98] Matthias Schunter, Michael Waidner, Dale Whinnett: A status report on the *SEMPER* framework for secure electronic commerce; Computer Networks and ISDN Systems 30 / (1998) 1501-1510.

[ScWW1 99] Matthias Schunter, Michael Waidner, Dale Whinnett: The *SEMPER* Framework for Secure Electronic Commerce; Wirtschaftsinformatik 41 / 3 (1999), 238-247.

[Semp 00] Gérard Lacoste, Birgit Pfitzmann, Michael Steiner, Michael Waidner (eds.): *SEMPER*: Secure Electronic Marketplace for Europe; LNCS 1854, Springer-Verlag, Berlin 2000.

[Semp 98] *SEMPER* Consortium: Architecture, services and protocols. Deliverable D10 of ACTS Project AC026, December 1998.

[SET 97] Mastercard Inc., Visa Inc.: Secure Electronic Transactions (SET) Version 1.0 - Book 1: Business Descriptions, Book 2: Programmer's Guide, Book 3: Formal Protocol Specification; Report, May 31, 1997.

[Sham 79] Adi Shamir: How to Share a Secret; Communications of the ACM 22 / 11 (1979) 612-613.

[ShBD 98] Gadi Shamir, Michael Ben-Or, Danny Dolev: BARTER — a Backbone ARchitecture for Trade of ElectRonic content; Trends in Distributed Systems for Electronic Commerce (TREC), LNCS 1402, Springer-Verlag, Berlin 1998, 65-79.

[ShMi 99] V. Shmatikov, J. Mitchell: Analysis of a Fair Exchange Protocol, In: Nevin Heinze, Edmund Clarke (eds.): Formal Methods and Security Protocols, Proceedings of the 1999 Federated Logic Conference Workshop, Trento IT, July 05, 1999.

[Shou 99] Victor Shoup: On Formal Models for Secure Key Exchange; IBM Research Report RZ 3076 (##93122), IBM Research Division, Zürich, November 1998.

[SiKS 97] A. Silberschatz, H. Korth, S. Sudarshan: Database system concepts; 3rd ed., McGraw-Hill 1997.

[SuTy 96] Jiawen Su, J. D. Tygar: Building Blocks for Atomicity in Electronic Commerce; 6th USENIX Security Symposium, 1996, 97-104.

[Syve2 98] Paul Syverson: Weakly Secret Bit Commitment: Applications to Lotteries and Fair Exchange; 11th Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos 1998, 2-13.

[Tang 96] Lei Tang: Verifiable Transaction Atomicity For Electronic Payment Protocols; 16th International Conference on Distributed Computing Systems, 1996, IEEE Computer Society, 1996, 261-269.

[Tang1 96] Lei Tang: A Framework for Building an Electronic Currency System; 6th USENIX Security Symposium, 113-122.

[Tedr 84] Tom Tedrick: How to exchange half a bit; Crypto '83, Plenum Press, New York 1984, 147-151.

[Tedr 85] Tom Tedrick: Fair Exchange of Secrets; Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 434-438.

[VaVa 83] Umesh V. Vazirani, Vijay V. Vazirani: Trapdoor Pseudo-random Number Generators, with Applications to Protocol Design; 24th Symposium on Foundations of Computer Science (FOCS) 1983, IEEE Computer Society, 1983, 23-30.

[Veit 99] Jörg Veit: Formal Fairness Proofs for Optimistic Contract Signing Protocols; Diploma thesis; Saarland University, 10 / 15 / 1999.

[Waid6 98] Michael Waidner: Open Issues in Secure Electronic Commerce; IBM Research Report RZ 3070 26 / 10 / 1998, IBM Research Division, Zürich, Oct. 1998.

[Weid 97] Christoph Weidenbach; SPASS Version 0.49; Journal of Automated Reasoning 18 (1997) 247-252.

[Weid 98] Christoph Weidenbach: Sorted Unification and Tree Automata; in: W. Bibel, P. H. Schmidt (ed.): Automated deduction - a basis for applications, Vol. I, Kluwer Academic Publishers, The Netherlands 1998, 291-320.

[Yao 82] Andrew C. Yao: Protocols for Secure Computations; 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 160-164.

[Yao1 82] Andrew C. Yao: Theory and Applications of Trapdoor Functions; 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 80-91.

[Yao 86] Andrew Chi-Chih Yao: How to Generate and Exchange Secrets; 27th Symposium on Foundations of Computer Science (FOCS) 1986, IEEE Computer Society, 1986, 162-167.

[ZhGo 96] Jianying Zhou, Dieter Gollmann: A Fair Non-repudiation Protocol; IEEE Symposium on Research in Security and Privacy 1996, Oakland 55-61.

[ZhGo 97] JianYing Zhou, Dieter Gollmann: An Efficient Non-repudiation Protocol; 10th Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos 1997, 126-132.

[ZhGo1 96] Jianying Zhou, Dieter Gollmann: Certified Electronic Mail; ESORICS '96 (4th European Symposium on Research in Computer Security), Rome, LNCS 1146, Springer-Verlag, Berlin 1996, 160-171.

[ZhGo2 96] Jianying Zhou, Dieter Gollmann: Observations on Non-repudiation; Asiacrypt '96, LNCS 1163, Springer-Verlag, Berlin 1996, 365-375 133-144.

[ZhGo3 96] Jianying Zhou, Dieter Gollmann: Evidence and Non-Repudiation; Draft, University of London, 1997.

[Zhou 96] Jianying Zhou: Non-Repudiation; PhD Thesis, University of London, 1997.

[ZhSh 96] N. Zhang, Q. Shi: Achieving Non-repudiation of Receipt; The Computer Journal 39 / 10 (1996) 844-853.

[Zwiß 97] Sonja Zwißler: Ein Protokoll zur Risikoreduktion bei elektronischer Auslieferung; Datenschutz und Datensicherheit DuD 21 / 7 (1997) 411-415.

Index