UNIVERSITÄT
DES
SAARLANDES

# The Subfield Bilinear Collision Problem and its Application to Post-Quantum Signatures

Dissertation
zur Erlangung des Grades
des Doktors der Naturwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

vorgelegt von

Janik Huth

Saarbrücken, 2025

| | |
|---|---|
| Tag des Kolloquiums: | 20. März 2026 |
| Dekan: | Prof. Dr. Roland Speicher |
| Prüfungsausschuss: | |
| Vorsitzender: | Prof. Dr. Ingmar Weber |
| Berichterstattende: | Prof. Dr. Antoine Joux |
| | Prof. Dr. Julian Loss |
| | Prof. Dr. Markus Bläser |
| Akademischer Mitarbeiter: | Dr. Hendrik Waldner |

# Abstract

In this dissertation, we introduce the Subfield Bilinear Collision (SBC) problem as a new cryptographic hardness assumption and explore its application to post-quantum signature schemes.

First, we construct an MPC-in-the-Head (MPCitH) based identification protocol and digital signature scheme from SBC. The resulting signature scheme is more compact compared to previous post-quantum MPCitH proposals, while retaining efficient signing and verification algorithms. Building on this, we design a VOLE-in-the-Head (VOLEitH) signature scheme from SBC, further reducing communication costs and signature size.

In order to achieve these improvements, we develop new generic techniques, including a faster hypercube folding algorithm, the correlated GGM forest, and MinMax commitments for GGM trees. These optimizations are not tied to the SBC problem and can therefore be applied more broadly to improve the efficiency of schemes based on MPCitH and VOLEitH.

Finally, we extend the methodology of MPCitH beyond digital signatures to Fully Homomorphic Encryption (FHE). We propose Fherret, a novel proof system that leverages MPCitH techniques to provide correctness and honest evaluation guarantees for FHE while preserving circuit privacy. This approach protects against reaction-based attacks without requiring the heavy overhead of zk-SNARK-based methods.

# Zusammenfassung

In dieser Dissertation stellen wir das Subfield Bilinear Collision (SBC) Problem als neue kryptografische Annahme vor und untersuchen dessen Anwendung auf quantenresistente digitale Signaturen.
Zunächst konstruieren wir ein Identifikationsprotokoll, basierend auf MPC-in-the-Head (MPCitH) sowie ein darauf aufbauendes digitales Signaturschema aus dem SBC Problem. Die resultierende Signatur ist kompakter als bisherige quantenresistente MPCitH Schemata und ermöglicht effiziente Algorithmen zum Signieren und Verifizieren. Darauf aufbauend entwickeln wir zusätzlich ein VOLE-in-the-Head (VOLEitH) Signaturschema, basierend auf dem SBC Problem, welches die Kommunikationskosten und Signaturgröße nochmals reduziert.
Um diese Verbesserungen zu erreichen, entwickeln wir neue generische Techniken. Darunter einen schnelleren Algorithmus zum Berechnen des Hypercube-Foldings, den korrelierten GGM Forest sowie das MinMax Commitment-Verfahren für GGM Binärbäume.
Diese Optimierungen sind nicht an das SBC Problem gebunden und können daher zur Effizienzsteigerung von Signaturverfahren, basierend auf MPCitH und VOLEitH genutzt werden.
Abschließend erweitern wir die MPCitH Techniken über digitale Signaturen hinaus auf vollständig homomorphe Verschlüsselungsverfahren (FHE). Mit Fherret präsentieren wir ein neuartiges Beweissystem, welches ebenfalls Techniken aus MPCitH benutzt, um die Korrektheit und ehrliche Auswertung von FHE Verfahren zu gewährleisten und gleichzeitig die Circuit Privacy erhält.

# Publication History of this Dissertation

This dissertation is based on the following papers and their respective full versions. More details about the references can be found in the Bibliography. At the beginning of each chapter, we provide further details about the relevant reference and the changes made compared to the original version.

[HJ24a] **MPC in the Head using the Subfield Bilinear Collision Problem**
Janik Huth, Antoine Joux
CRYPTO 2024, ©IACR 2024

[HJ25] **VOLE-in-the-Head Signatures from Subfield Bilinear Collisions**
Janik Huth, Antoine Joux
ASIACRYPT 2025, ©IACR 2025

[HJS25] **Fherret: Proof of FHE Correct-and-Honest Evaluation with Circuit Privacy from MPCitH**
Janik Huth, Antoine Joux, Giacomo Santato
Cryptology ePrint Archive, 2025, Preprint

# Acknowledgments

The completion of this dissertation marks the end of a long journey that began more than 20 years ago when I first discovered my fascination for mathematics in elementary school. That curiosity grew stronger through my Bachelor's and Master's studies in mathematics and gradually led me from pure mathematics to the field of cryptography and this PhD.
I would like to express my deepest gratitude to all the people I met along the way who have supported, encouraged, and inspired me at different stages of this journey.
First and foremost, I want to thank my supervisor Antoine Joux for his exceptional mentorship, great advice, and continuous support throughout my PhD. His guidance made the transition from pure mathematics to cryptographic research both enjoyable and rewarding. I am deeply grateful for the many insightful discussions, his patience, and for the stimulating research environment that helped me to grow as a researcher and navigate the challenges and opportunities of a new field.
I also wish to thank my other coauthor Giacomo Santato for the great collaboration, which allowed me to explore new ideas beyond my initial research scope. Working together has been both productive and inspiring. My sincere thanks also goes to all my colleagues at CISPA for creating such an intellectually stimulating and friendly atmosphere. The many discussions during lunch breaks, internal seminars, and conference trips were a highlight of my time here. I am especially thankful for the friendships that have grown out of these encounters.
Moreover, I would like to thank all the teachers and professors I met throughout my studies who have fueled my excitement for mathematics, computer science, and cryptography. Their passion for teaching and research has shaped my academic path in many ways.

Ich möchte diese Gelegenheit nutzen, um auch meinem privaten Umfeld von ganzem Herzen zu danken. Durch meine Freunde und Familie hatte ich während der gesamten Promotionszeit nicht nur Unterstützung, sondern auch die nötige Ablenkung vom Studium und Forschungsalltag.
Ein ganz besonderer Dank gilt dabei meinen Eltern, die mich mein ganzes Leben lang bedingungslos unterstützt haben. Sie haben mir die Freiheit gegeben, meinen eigenen Weg zu gehen, und mir gleichzeitig stets Rückhalt und Zuversicht vermittelt. Ohne ihre Geduld, Ermutigung und Liebe wäre diese Dissertation nicht möglich gewesen.

UNIVERSITÄT DES SAARLANDES

## Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

## Declaration of original authorship

I hereby declare that this dissertation is my own original work except where otherwise indicated. All data or concepts drawn directly or indirectly from other sources have been correctly acknowledged. This dissertation has not been submitted in its present or similar form to any other academic institution either in Germany or abroad for the award of any other degree.

Saarbrücken, ***November 2025***

gez. / signed

**Janik Huth**

# Contents

CHAPTER 1

# Introduction

In the digital age, the ability to communicate securely, authenticate identities and verify the integrity of data across untrusted channels is essential. Applications reach from the confidentiality of online banking to the authenticity of software updates in critical infrastructure and are all based on cryptographic primitives. The goal of these primitives is to enable modern communication systems to operate safely at a global scale while being convenient to use.

In a nutshell, cryptography turns hard mathematical problems into practical security of a system. When we encrypt a message or digitally sign a document, we rely on the fact that certain computations are beyond the reach of any feasible adversary. We can formalize this concept using the notion of *hardness assumptions*: we assume that a particular computational problem cannot be solved efficiently, and build the security proof of our scheme based on this assumption.

Since the starting point of public key cryptography in the paper from Whitfield Diffie and Martin Hellman in 1976 [DH76], various hardness assumptions have been proposed. For instance, the introduction of RSA in 1977 by Rivest, Shamir, and Adleman [RSA78] demonstrated how results from number theory could be transformed into an efficient public-key cryptosystem. Elliptic curve cryptography (ECC), which was developed in the 1980s and 1990s, illustrated how the same security level could be achieved with smaller keys by relying on the hardness of the *Discrete Logarithm Problem* (DLP) in certain elliptic curve groups.

Throughout the history of cryptographic research, there are examples of assumptions that were believed secure which were later significantly weakened or even broken by algorithmic advances. One such development that poses a possible threat to many applications used today is *quantum computing*. While it is still in early stages, quantum computation poses an existential threat to the number-theoretic assumptions that underlie most of the public key systems deployed today.

Peter Shor's algorithms [Sho94] for factoring and for computing discrete logarithms show that a sufficiently powerful quantum computer could break RSA, DSA and ECC in polynomial time. This threat is particularly problematic for digital signatures and key exchange protocols that secure long lived data: even if quantum computers arrive only decades from now, adversaries might intercept information today and decrypt or forge it in the future once sufficient advancements in technology are made.

To prevent this threat, there is an ongoing effort of the scientific community to design *post-quantum cryptography* (PQC), containing cryptographic algorithms that are secure against both classical and quantum adversaries. Based on this effort, there is a larger focus on mathematical hardness assumptions that are currently considered post-quantum secure. In particular, algorithms involving lattices, error correcting codes, multivariate systems and isogenies are among the major candidates for post-quantum secure signature schemes. In line with this trend, the National Institute of Standards and Technology

(NIST) issued multiple calls for digital signature proposals to be considered in their PQC standardization process.

## 1.1 Digital Signatures

Digital signatures [DH76] allow a user, who is in possession of a secret key `sk`, to sign a message `msg` by computing a signature $\sigma$. This signature can be verified by anyone given the corresponding public key `pk` of the signer. However, it should not be feasible to generate valid signatures for new messages without knowing the secret key `sk`, which is a property called *unforgeability*.
Digital signatures are a fundamental building block of public-key cryptography. Classical constructions [RSA78, Sch90, Ame05] rely on the hardness of factoring and of the discrete logarithm problem. In the quest for post-quantum digital signatures, ongoing research focuses on identifying candidates that achieve strong security while maintaining efficiency and compactness.
Within the second round of the NIST PQC standardization process, the proposed candidates for signature schemes can be classified into six categories:

- Code-based signatures
- Isogeny signatures
- Lattice-based signatures
- MPC-in-the-Head signatures
- Multivariate signatures
- Symmetric-based signatures

In the MPC-in-the-Head signatures category, there are a total of five submissions [AAB$^+$24, BBFR24, ABB$^+$24a, ABB$^+$24c, ABB$^+$24b]. Since the NIST call in 2022, there has been a lot of progress and optimization in the realm of MPC-in-the-Head (MPCitH) signatures, introducing faster and more compact signatures along the way.

## 1.2 MPC-in-the-Head Signatures

The MPC-in-the-Head paradigm [IKOS07] is a versatile tool for designing Zero-Knowledge (ZK) protocols and digital signatures. It allows the construction of proofs of knowledge for NP-relations by emulating a compatible multi-party computation "in the head" of a prover. In this setting, the prover commits to simulated views of the parties and the verifier checks the correctness by inspecting all but one of them, which by design does not reveal any information about the witness of the NP-relation. This results in ZK proofs that can be made non-interactive using the Fiat-Shamir transform [FS87]. In recent years, MPCitH has been extensively studied and many new signature schemes based on different hardness assumptions have been proposed [FJR22, FR23, BKPV23, ABB$^+$23, Fen22]. However, compared to classical signatures, their signature size is relatively large, which has motivated ongoing research into improving their efficiency. At the same time, the MPCitH paradigm has shown remarkable flexibility, as it can be instantiated from various different hardness assumptions.

## 1.3 VOLE-in-the-Head Signatures

An important refinement of the MPCitH paradigm is called *Vector Oblivious Linear Evaluation in the Head* (VOLEitH) [BBD+23]. While the fundamental concepts are similar to the MPCitH paradigm, the main advantage of using VOLEitH is that it often leads to an improvement of signature sizes and running times. The key idea is to replace the generic MPC protocol with a lighter primitive based on *Vector Oblivious Linear Evaluation* (VOLE). This VOLE functionality is induced by the GGM tree structure [GGM86], which is already used to instantiate the puncturable PRF in order to reduce the communication of MPCitH schemes. Using it often yields simplified protocols and leads to a reduction in both signature size and running time of the corresponding digital signature schemes.

## 1.4 Fully Homomorphic Encryption

*Fully Homomorphic Encryption* (FHE) [Riv87, Gen09b] is a cryptographic primitive that enables computation on encrypted data without requiring decryption. Specifically, a client encrypts a message $m$ and sends the ciphertext to a server. The server, having chosen a function $f$, evaluates it homomorphically on the ciphertext and returns a result that decrypts to $f(m)$. In particular, the client does not reveal the plaintext message $m$. This property makes FHE a cornerstone for applications such as secure cloud computing, privacy-preserving machine learning, and genomic data analysis. When interacting with an untrusted evaluator, the client needs additional mechanisms to ensure that computations are performed correctly and honestly. Moreover, many practical use cases require that the evaluated circuit remains private, which gives rise to the notion of *circuit privacy*.

## 1.5 Overview of Results

This thesis focuses on the design of post-quantum digital signatures in the MPC-in-the-Head paradigm, building on a new hardness assumption that we call the *Subfield Bilinear Collision (SBC) problem*. The SBC problem originates from algebraic structures that occur in algorithms for solving the discrete logarithm problem in small characteristic finite fields. By leveraging the properties of SBC, we construct simple and efficient MPCitH signatures, reducing the state-of-the-art signature sizes significantly. The resulting interactive protocol is designed specifically for the SBC problem, and the improvements in terms of signature size mainly come from the fact that the SBC protocol fits the MPCitH paradigm very well. To push this line of research further, we adopt a refinement of the MPCitH paradigm called *Vector Oblivious Linear Evaluation in the Head (VOLEitH)* paradigm. This construction builds on the same high level idea of simulating an MPC protocol "in the head" of a prover. However, it uses techniques based on vector oblivious linear evaluation induced by the GGM tree structure used in the MPCitH scheme. Using these techniques allows for a significant reduction of the communication cost and therefore of the overall signature size of the SBC protocol. The compactness of both the MPCitH and VOLEitH signature schemes based on the SBC problem comes from the fact that the particular structure of SBC suits these paradigms very well.
Apart from the improvements tied to the SBC problem, we also introduce new techniques that can be applied to any MPCitH style protocol and are not tied to any specific assumption. These techniques result in improvements of both signature size and running times

of MPCitH and VOLEitH schemes. In particular, we introduce a fast hypercube folding algorithm with running time $\mathcal{O}(N)$ to improve the performance of MPCitH schemes in the hypercube setting.
We further introduce the *correlated GGM forest*, by combining the correlated GGM tree construction [GYW+23] and the hypercube folding technique [AGH+23] to correlate GGM trees across multiple rounds of the protocol. Using the correlated GGM forest allows us to reduce the communication cost since the need for offset values corresponding to the internal bits of the GGM trees becomes redundant. To further reduce the communication cost, we propose a new, implicit commitment to the cGGM forest called *MinMax commitment*.
Lastly, we explore how techniques from the MPCitH paradigm can be combined with FHE to provide lightweight proofs of correct and honest evaluation. In particular, we introduce *Fherret*, a generic construction that extends MPCitH techniques to the FHE setting. Fherret offers verifiable homomorphic evaluation while preserving circuit privacy and protects FHE schemes against reaction-based attacks.

## 1.6 Structure of this Dissertation

In Chapter 2, we provide the cryptographic background, including basic tools and formal definitions needed throughout this dissertation. In Chapter 3, we present the Subfield Bilinear Collision (SBC) Problem as a new hardness assumption and analyze its properties. We use the SBC problem in Chapter 4 to construct an identification protocol and a signature scheme based on MPCitH using the Fiat-Shamir transform [FS87]. We refine this approach in Chapter 5 by using the VOLEitH paradigm to achieve improved signature sizes and efficiency. In Chapter 6, we show how to use methods from MPCitH schemes to protect FHE schemes against reaction-based attacks in the circuit privacy setting. Finally, Chapter 7 concludes.

CHAPTER 2

# Preliminaries

## 2.1 Notation

We introduce general notation and conventions used throughout this dissertation. Many of these notations are common for mathematical and cryptographic literature, see for example [KL14, Gol01].

**Sets.** We denote the set of integers by $\mathbb{Z}$. For any positive integer $m \in \mathbb{Z}^+$, we denote the set $\{0, \ldots, m-1\}$ by $[m]$. For $m > 1$, we denote the set $\{1, \ldots, m-1\}$ by $[m]^*$. For a prime or prime power $q$, we denote the finite field with $q$ elements by $\mathbb{F}_q$. For any field $\mathbb{F}_q$, we denote its multiplicative group of units by $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$. By $\{0,1\}^*$, we denote the set of all finite strings of bits. In particular, $\{0,1\}^* = \bigcup_{i \geq 0} \{0,1\}^i$. For any finite set $S$, we denote the cardinality of $S$ by $|S|$. The empty set is $\emptyset$.

**Matrices and Vectors.** Let $\mathbb{F}$ be a finite field and let $k, n$ be positive integers. We denote the set of all $n \times k$ matrices over $\mathbb{F}$ by $\mathbb{F}^{n \times k}$ and the set of all vectors of size $n$ with coordinates in $\mathbb{F}$ by $\mathbb{F}^n$. We denote vectors using arrow notation, for example $\vec{x}$. For a fixed vector $\vec{x} \in \mathbb{F}^n$, the subspace generated by $\vec{x}$ is denoted by $\langle \vec{x} \rangle$ and the projective space over $\mathbb{F}$ of dimension $n$ is denoted by $\mathbb{P}_n(\mathbb{F})$. We denote the dot product of two vectors $\vec{u} \in \mathbb{F}^n$ and $\vec{x} \in \mathbb{F}^n$ by $\vec{u} \cdot \vec{x} = \sum_{i=0}^{n-1} u_i x_i$. Let $\vec{a} \in \mathbb{F}^{n_1}$ and $\vec{b} \in \mathbb{F}^{n_2}$ be two vectors of dimensions $n_1$ and $n_2$, respectively. We denote the vector of dimension $n_1 + n_2$ resulting from the concatenation of the coordinates of $\vec{a}$ and $\vec{b}$ by $(\vec{a}, \vec{b})$. Furthermore, we use the notation $|\cdot|$ to denote the length of a vector or a binary sequence. For a bit string $x \in \{0,1\}^*$, $|x|$ denotes the number of bits of $x$.

**Polynomials.** We denote the polynomial ring over $\mathbb{F}$ in $X$ by $\mathbb{F}[X]$ and the polynomial ring in two variables $X, Y$ by $\mathbb{F}[X, Y]$. Let $R$ be a polynomial ring and let $f, g \in R$. We say that $f$ divides $g$, using the notation $f \,|\, g$, if there is a polynomial $s \in R$ with $g = s \cdot f$. Let $n \geq k \geq 0$ be two integers. We denote the binomial coefficient of $n$ over $k$ by $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$.

**Probability.** If $S$ is a finite set, then the notation $s \leftarrow_\$ S$ means that $s$ is uniformly sampled at random from $S$. By using the notation $s \xleftarrow{r}_\$ S$, we indicate that $s$ is sampled pseudorandomly from $S$ based on the seed $r$. Let $D$ denote any probability distribution. Then the notation $s \leftarrow D$ indicates that $s$ is sampled from $D$. Let $t, \varepsilon \colon \mathbb{N} \to \mathbb{R}$. We say that two distributions $\{D_\lambda\}_\lambda$ and $\{E_\lambda\}_\lambda$ indexed by $\lambda$ are $(t, \varepsilon)$ indistinguishable, if for any algorithm $\mathcal{A}$ with running time at most $t(\lambda)$ we have that

$$\left| \Pr\left[\mathcal{A}^{D_\lambda}(\cdot) = 1\right] - \Pr\left[\mathcal{A}^{E_\lambda}(\cdot) = 1\right] \right| \leq \varepsilon(\lambda).$$

The two distributions are called

- *computationally indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every $t \in \mathsf{poly}(\lambda)$,
- *statistically indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every (unbounded) $t$,

- *perfectly indistinguishable* if $\varepsilon = 0$.

We use the notation $\{D_\lambda\}\approx_c\{E_\lambda\}$ to indicate that the two distributions $\{D_\lambda\}_\lambda$ and $\{E_\lambda\}_\lambda$ are computationally indistinguishable.
For two events $A$ and $B$, we denote the conditional probability of $A$ given $B$ by $\Pr[A|B]$. In our proofs, we often use the union bound for probabilities, which states that $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$ for any events $A$ and $B$.
**Functions.** A function $\mu\colon \mathbb{N} \to \mathbb{R}$ is called *negligible* if for every positive polynomial p, there exists an integer $N_p > 0$ such that for all $x > N_p$, we have $|\mu(x)| < 1/p(x)$. We denote an (arbitrary) negligible function in $\lambda$ by $\mathsf{negl}(\lambda)$ and a polynomial function in $\lambda$ by $\mathsf{poly}(\lambda)$. By $\mathsf{poly}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, we denote a polynomial function in the variables $\lambda_1, \ldots, \lambda_n$.
Throughout this dissertation, we denote the base-2 logarithm by log. We use the standard big-O notation $\mathcal{O}$ to describe asymptotic behavior. By $\lambda$, we denote the bit security level. Every algorithm implicitly gets the unary encoding $1^\lambda$ of $\lambda$ as an input.
**Algorithms.** When dealing with a nondeterministic algorithm $\mathcal{A}(\text{input})$, we sometimes write $\mathcal{A}(\sigma; \text{input})$ to consider explicitly the random coins $\sigma$ of the algorithm and treat it as deterministic. Let $\mathcal{E}$ be a function or an algorithm. We use the notation $\mathcal{A}^{\mathcal{E}}$ to indicate that $\mathcal{A}$ gets oracle access to $\mathcal{E}$. In particular, $\mathcal{A}$ can send an input $x$ to $\mathcal{E}$ and receives the output $\mathcal{E}(x)$ in return.

## 2.2 Cryptographic Tools

### 2.2.1 Hash Functions

A hash function is an algorithm that converts input data of any length into a fixed-size bitstring. Intuitively, a cryptographic hash function should behave like a digital fingerprint for input data. The goal is that even a tiny change to the input results in a drastic change of the output and that it should be practically infeasible to invert the function.
Two of the most important security notions for hash functions are collision resistance and preimage resistance. We define these formally:

**Definition 2.1** (Collision Resistant Hash Functions)**.** A family of functions

$$\left\{\mathcal{H}_k\colon \{0,1\}^* \to \{0,1\}^{l(k)} \,\middle|\, k \in \{0,1\}^{m(\lambda)}\right\}_\lambda$$

indexed by a security parameter $\lambda$ is called *collision resistant* for some $l, m\colon \mathbb{N} \to \mathbb{N}$, if there exists a negligible function $\mu$ such that, for any PPT algorithm $\mathcal{A}$, we have

$$\Pr\left[\mathcal{H}_k(x) = \mathcal{H}_k(x') \wedge x \neq x' \,\middle|\, \begin{array}{l} k \leftarrow \{0,1\}^{m(\lambda)} \\ (x, x') \leftarrow \mathcal{A}(k) \end{array}\right] \leq \mu(\lambda).$$

Collision resistance means that it is computationally infeasible to find any two inputs that map to the same output. This property is crucial for many cryptographic applications like commitment schemes and digital signatures, which we work with in this dissertation. Finding a collision in the respective hash function used in those applications would undermine correctness or security.

**Definition 2.2** (Preimage Resistant Hash Functions)**.** A family of functions

$$\left\{\mathcal{H}_k\colon \{0,1\}^* \to \{0,1\}^{l(k)} \,\middle|\, k \in \{0,1\}^{m(\lambda)}\right\}_\lambda$$

indexed by a security parameter $\lambda$ is called *preimage resistant* for some $l, m\colon \mathbb{N} \to \mathbb{N}$, if there exists a negligible function $\mu$ such that, for any PPT algorithm $\mathcal{A}$, we have

$$\Pr\left[\mathcal{H}_k(x') = y \,\middle|\, \begin{array}{c} k \leftarrow \{0,1\}^{m(\lambda)}, x \leftarrow_\$ \{0,1\}^*, y \leftarrow \mathcal{H}_k(x) \\ x' \leftarrow \mathcal{A}(k, y) \end{array}\right] \leq \mu(\lambda).$$

Preimage resistance means that given a hash output, it is hard to find any input that maps to it. This ensures that hash functions cannot be easily inverted, which is a property that is essential for hiding information in many cryptographic protocols.

**The Random Oracle Model.** In the security analysis of our constructions, we rely on the random oracle model [BR93]. In this model, hash functions $\mathcal{H}\colon \mathcal{D} \to \mathcal{R}$ are given by idealized, truly random functions. In the schemes as well as in the security games, every algorithm gets oracle access to $\mathcal{H}$. We assume that the function $\mathcal{H}$ is implemented using the *lazy sampling* technique: for any query $x \in \mathcal{D}$, the oracle checks if $\mathcal{H}(x) = \perp$. If this is the case, the oracle samples a value $\mathcal{H}(x) \leftarrow_\$ \mathcal{R}$ uniformly at random and returns this value. Otherwise, it returns $\mathcal{H}(x)$ directly. This abstraction allows for simplified proofs, assuming that hash functions behave like idealized black boxes that are given to each algorithm. In particular, in each scheme and in each security game, all algorithms, including the adversary, have oracle access to $\mathcal{H}$.

We also use the technique of *domain separation* to simulate multiple random oracles from a single one. In practice, domain separation is realized by prepending a distinct prefix depending on the context of the query to the input of each hash. This prevents collisions between inputs that are intended for different purposes, even when the same underlying hash function is used.

### 2.2.2 Pseudorandom Generators

Pseudorandom generators (PRGs) are a fundamental building block in many cryptographic applications. Intuitively, a PRG takes short input values and expands them into longer output values that look random to any efficient adversary. This means that the output of a PRG should look computationally indistinguishable from a truly random string of the same length, even though it was generated deterministically from a short input seed. This ability to expand randomness is essential in many applications like key generation or the construction of pseudorandom functions. We now introduce this notion formally:

**Definition 2.3** (Pseudorandom Generator (PRG))**.** A deterministic function $G\colon \{0,1\}^\lambda \to \{0,1\}^m$ is a $(t, \varepsilon)$-secure pseudorandom generator (PRG) if

- $\lambda < m$ and
- the distributions

  $$\{G(x) \,|\, x \leftarrow_\$ \{0,1\}^\lambda\} \text{ and } \{r \,|\, r \leftarrow_\$ \{0,1\}^m\}$$

  are $(t, \varepsilon)$ computationally indistinguishable.

In other words, no PPT adversary can efficiently distinguish the output of $G$ from a truly random $m$-bit string with advantage more than $\varepsilon \in \mathsf{negl}(\lambda)$.
For our goal of building identification schemes and signatures based on the MPC-in-the-Head paradigm, it is common to use a primitive with an additional property, called puncturable pseudorandom functions (PPRFs). A PPRF is a special kind of function that allows us to "puncture" it at one point. In particular, we can hide its value for a specific input, while still being able to reveal it everywhere else. The key property that we want from a PPRF is that even if an adversary learns every value of the function except at one point, it cannot distinguish the hidden value from a truly random one.

**Definition 2.4** (Puncturable Pseudorandom Function (PPRF))**.** A family $F$ of *puncturable PRFs* on $[N]$ is a set of functions $F$ indexed by a key $K$ with domain $[N]$ satisfying the following properties:

- For each key $K$ and index $i \in [N]$ there exists a punctured key $K_{i^*}$ and an algorithm $\mathcal{A}$ such that

$$\text{for each } j \in [N] \setminus \{i\} : \mathcal{A}(K_{i^*}, j) = F_K(j).$$

- Given the punctured key $K_{i^*}$, the value $F_K(i)$ is indistinguishable from a random value.

In the design of MPC-in-the-Head and VOLE-in-the-Head schemes, PPRFs are particularly useful. In these protocols, the prover simulates multiple parties and hides one of them. In such a setting, the properties of a PPRF ensure that revealing partial information does not leak anything about the missing (punctured) value. A state-of-the-art method to instantiate PPRFs in practice is the binary tree based construction from Goldwasser, Goldreich and Micali [GGM86], called GGM trees.
**The GGM tree construction.** Starting from a root seed, the idea is to construct a binary tree with $N = 2^D$ leaves by repeatedly applying a length-doubling PRG to obtain the left and right child of a given node. An illustration of this construction is given in Figure 2.1. In this construction, to reveal all leaves except for a specific one indexed by $i^* \in [N]$, we disclose the values of the sibling nodes along the path from the root to leaf $i^*$. Using the PRG, this allows reconstruction of every leaf except at position $i^*$ by communicating only $\log(N) = D$ tree nodes, rather than sending $N - 1$ leaf values. We provide an example of a PPRF key corresponding to a GGM tree in Figure 2.2.

### 2.2.3 Commitment Schemes

A commitment scheme is the digital analog of sealing a message in an envelope: the sender commits to a value while keeping it hidden, but cannot change it later. It provides two main guarantees:

- hiding, meaning that the commitment does not reveal the message,
- binding, meaning that once committed, the message cannot be changed.

These properties are essential in a wide range of cryptographic protocols, including Zero-Knowledge proofs and secure multiparty computation.
We formally define a commitment scheme below:

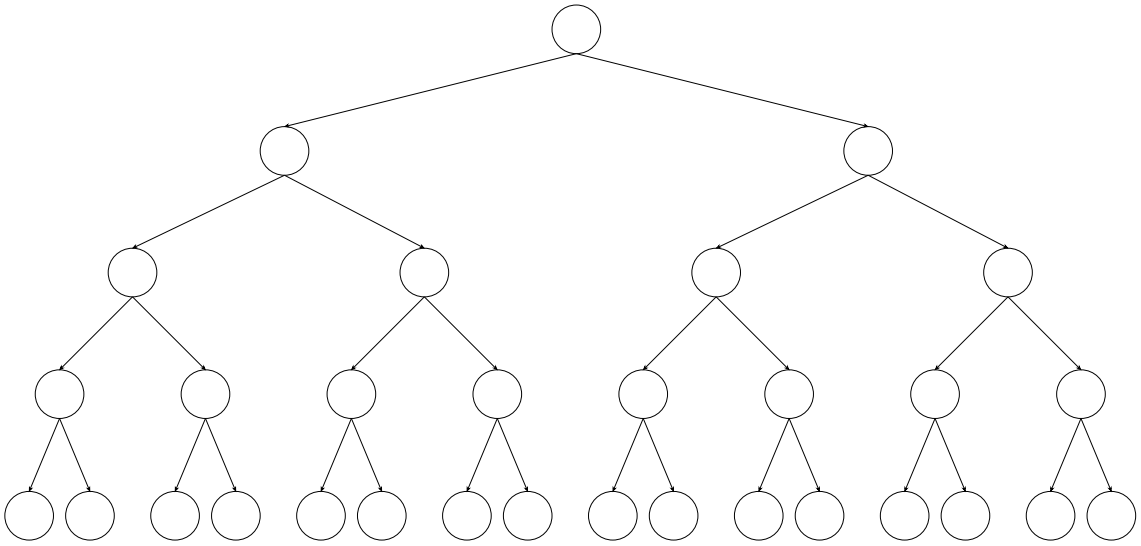

**Figure 2.1:** Illustration of the GGM tree construction for $N = 16$ leaves. For any given node, the left and right child are computed using $\mathsf{left} \,\|\, \mathsf{right} \leftarrow \mathrm{PRG}(\mathsf{node})$ using a length doubling PRG.

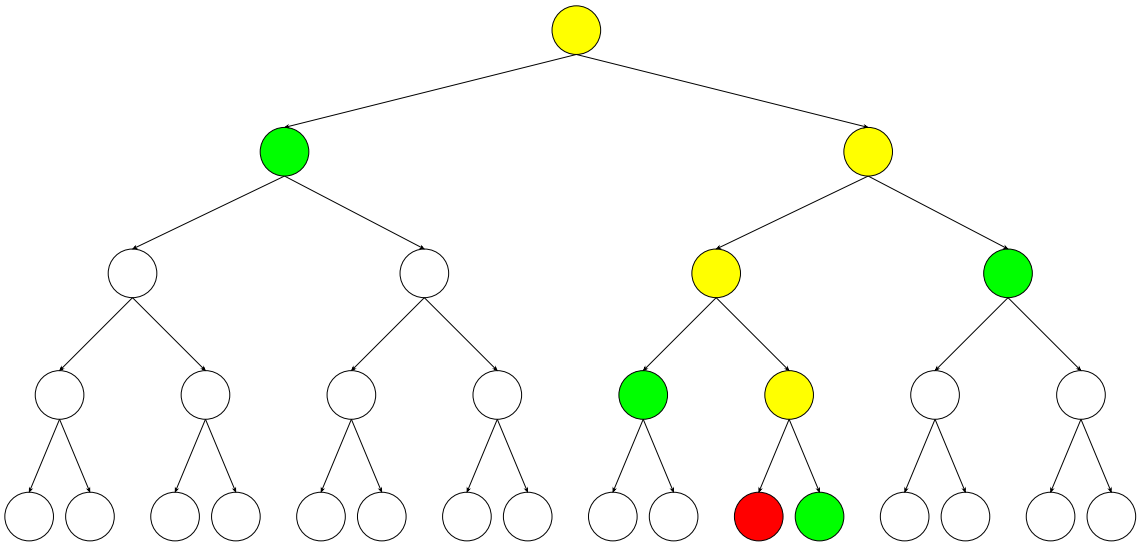

**Figure 2.2:** Example of a PPRF key for $i^* = 10$ (red leaf). The yellow nodes correspond to the path from the root of the tree to the leaf at position $i^*$. The green nodes correspond to the siblings of these nodes and are included in the PPRF key. From the green nodes, every leaf except leaf $i^*$ can be recomputed.

**Definition 2.5** (Commitment Scheme)**.** A *commitment scheme* is a tuple of algorithms $(\mathtt{KeyGen}, \mathtt{Commit}, \mathtt{Verify})$ such that

- $\mathtt{KeyGen}(1^\lambda) \to \mathtt{ck}$ is a PPT algorithm that takes as input the security parameter $1^\lambda$ and outputs a commitment key $\mathtt{ck}$ containing the definition of the message space $\mathcal{M}_{\mathtt{ck}}$ and the randomness space $\mathcal{R}_{\mathtt{ck}}$.
- $\mathtt{Commit}(\mathtt{ck}, \mathsf{msg}, r) \to c$ is a deterministic polynomial time algorithm that takes as

input a commitment key $\mathtt{ck}$, a message $\mathsf{msg} \in \mathcal{M}_{\mathtt{ck}}$ and the randomness $r \in \mathcal{R}_{\mathtt{ck}}$ and outputs a commitment $c \in \{0,1\}^*$.

- $\mathtt{Verify}(\mathtt{ck}, \mathsf{msg}, r, c) \to b$ is a deterministic polynomial time algorithm that takes as input a commitment key $\mathtt{ck}$, a message $\mathsf{msg} \in \mathcal{M}_{\mathtt{ck}}$, the randomness $r \in \mathcal{R}_{\mathtt{ck}}$ and a commitment $c \in \{0,1\}^*$ and outputs a bit $b \in \{0,1\}$.

Furthermore, the following correctness and security properties should hold:

- **Correctness.** For any $\mathtt{ck} \leftarrow \mathtt{KeyGen}(1^\lambda)$, any message $\mathsf{msg} \in \mathcal{M}_{\mathtt{ck}}$ and any randomness $r \in \mathcal{R}_{\mathtt{ck}}$, we have that

$$\Pr\left[\mathtt{Verify}(\mathtt{ck}, \mathsf{msg}, r, c) = 1 \,|\, c \leftarrow \mathtt{Commit}(\mathtt{ck}, \mathsf{msg}, r)\right] = 1.$$

  This property ensures that correctly computed commitments always verify successfully when opened with the original message and randomness.

- **Hiding.** For any key $\mathtt{ck}$ and any messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}_{\mathtt{ck}}$, the distributions

$$\{c \,|\, r \leftarrow_\$ \mathcal{R}_{\mathtt{ck}}, c \leftarrow \mathtt{Commit}(\mathtt{ck}, \mathsf{msg}_0, r)\} \text{ and } \{c \,|\, r \leftarrow_\$ \mathcal{R}_{\mathtt{ck}}, c \leftarrow \mathtt{Commit}(\mathtt{ck}, \mathsf{msg}_1, r)\}$$

  are indistinguishable. This means that the commitment hides all information about the message: An adversary cannot distinguish whether the commitment corresponds to $\mathsf{msg}_0$ or $\mathsf{msg}_1$.

- **Computationally Binding.** There exists a negligible function $\mu$ such that for any PPT algorithm $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{c} \mathtt{Verify}(\mathtt{ck}, \mathsf{msg}_0, r_0, c) = 1 \\ \wedge\, \mathtt{Verify}(\mathtt{ck}, \mathsf{msg}_1, r_1, c) = 1 \\ \wedge\, \mathsf{msg}_0 \neq \mathsf{msg}_1 \end{array} \middle| \begin{array}{c} \mathtt{ck} \leftarrow \mathtt{KeyGen}(1^\lambda) \\ (\mathsf{msg}_0, \mathsf{msg}_1, r_0, r_1, c) \leftarrow \mathcal{A}(\mathtt{ck}) \end{array} \right] \leq \mu(\lambda).$$

  This ensures that the sender cannot open a valid commitment in two different ways to two different messages. In other words, the commitment binds the sender to the committed message.

One way to instantiate a commitment scheme is by using a cryptographic hash function $\mathcal{H}$. We can commit to a value $\mathsf{msg}$ by sampling the randomness $r$ and by computing

$$c \leftarrow \mathcal{H}(\mathsf{msg} \,\|\, r).$$

To open the commitment $c$, we can send the values $\mathsf{msg}$ and $r$ to the verifier, who can recompute the hash and check if it matches $c$. Since we use a random string $r$, the hiding property of this hash based commitment scheme follows from the preimage resistance of $\mathcal{H}$, while the binding property follows from the collision resistance of $\mathcal{H}$.

### 2.2.4 Zero Knowledge Proofs

In this dissertation, we consider interactive protocols between two parties, which are usually called *prover* $\mathcal{P}$ and *verifier* $\mathcal{V}$. *Zero-Knowledge* (ZK) proofs allow the prover to convince the verifier that a certain statement is true. Typically, the prover wants to convince the verifier of the knowledge of a secret witness to some language. Zero-Knowledge

proofs allow the prover to convince the verifier in such a way that the verifier does not learn any information about the witness, except for the fact that it is in the language. After receiving their respective inputs $\mathtt{in}_{\mathcal{P}}$ and $\mathtt{in}_{\mathcal{V}}$, the parties can interact with each other by exchanging messages. The first message sent by the prover is called a commitment and denoted by $\mathtt{com}$. The verifier responds by sending a first *challenge* $\mathtt{ch}_1$ to the prover, to which the prover responds by sending a *response* $\mathtt{res}_1$ to the verifier. In the next step, the verifier sends the second challenge $\mathtt{ch}_2$ and in turn receives a second response $\mathtt{res}_2$. The communication continues in this way until the verifier receives the last response, say $\mathtt{res}_n$, and computes the final output: a bit $b \in \{0,1\}$, corresponding to $\mathtt{accept}(b = 1)$ or $\mathtt{reject}(b = 0)$. Such an interactive proof is called a *sigma protocol* with $m = 2n + 1$ rounds. The *transcript* of such an interactive protocol consists of the sequence of all messages that are exchanged during an execution. In particular, we denote the transcript by

$$\mathrm{View}(\mathtt{in}_{\mathcal{P}}, \mathtt{in}_{\mathcal{V}}) = (\mathtt{com}, \mathtt{ch}_1, \mathtt{res}_1, \ldots, \mathtt{ch}_n, \mathtt{res}_n)$$

and the final output bit $b \in \{0,1\}$ of the transcript is denoted using the notation

$$\mathtt{out}(\mathrm{View}(\mathtt{in}_{\mathcal{P}}, \mathtt{in}_{\mathcal{V}})) \to b.$$

We say that an algorithm has *rewindable black-box access* to $\mathcal{P}$ if it can copy the state of $\mathcal{P}$ at any given time during this interaction, rewind $\mathcal{P}$ to an earlier state in the interaction and query $\mathcal{P}$ with a new challenge. We say that a value $x$ is *extractable* from $\mathcal{P}$ if there is a PPT algorithm $\mathcal{E}$, called *extractor*, that can return the value $x$ after a polynomial number of queries to $\mathcal{P}$, given rewindable black-box access to $\mathcal{P}$.

**Definition 2.6** (Proof of Knowledge)**.** Consider a language $L$ in $\mathsf{NP}$ and let $x \in L$ be a statement. Let $W(x)$ be the set of witnesses for $x$ and consider the relation

$$\mathcal{R} = \{(x, w) \,|\, x \in L, w \in W(x)\}.$$

An interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is a *proof of knowledge* for $\mathcal{R}$ with *soundness error* $\varepsilon$ if the following properties hold:

- **Correctness.** For each $(x, w) \in \mathcal{R}$, an honest prover who knows a witness $w$ for $x$ convinces the verifier of his knowledge:

  $$\Pr[\mathtt{out}(\mathrm{View}((x, w)_{\mathcal{P}}, x_{\mathcal{V}})) \to \mathtt{accept}] = 1.$$

- **Soundness.** If there is a PPT prover $\mathcal{P}^*$ with

  $$\varepsilon^* = \Pr[\mathtt{out}(\mathrm{View}(_{\mathcal{P}^*}, x_{\mathcal{V}})) \to \mathtt{accept}] > \varepsilon,$$

  then there exists an extractor $\mathcal{E}$ that, given rewindable black-box access to $\mathcal{P}^*$, outputs a witness $w'$ for $x$ in time $\mathrm{poly}(\lambda, \frac{1}{\varepsilon^* - \varepsilon})$.

  This property means that if a cheating prover can convince the verifier with noticeably better probability than $\varepsilon$, then it must know a valid witness, which the extractor can recover.

We now define the zero-knowledge property, which guarantees that the verifier learns nothing beyond the validity of the statement. In the honest-verifier setting, meaning

that the verifier follows the protocol correctly, this is formalized using the existence of a simulator that can generate a view which is indistinguishable from a real interaction.

**Definition 2.7** (Honest-Verifier Zero-Knowledge Proof of Knowledge)**.** A proof of knowledge is called {computationally, statistically, perfectly} *honest verifier zero knowledge* (HVZK) if there exists a PPT algorithm $\mathcal{S}(1^\lambda)$, called *simulator*, that has an output distribution which is {computationally, statistically, perfectly} indistinguishable from the distribution $\text{View}((x,w)_\mathcal{P}, x_\mathcal{V})$ obtained by interacting with an honest verifier $\mathcal{V}$.

### 2.2.5 Multiparty Computation

Throughout the thesis, we use additive sharings of elements. These elements are from additive modular structures like finite fields $\mathbb{F}$ or additive modular rings $\mathcal{R}$ like polynomial rings over finite fields. In an additive sharing scheme, a secret value is split into multiple parts that look random such that the secret can only be reconstructed when all shares are combined. For an element $x$, we denote a *sharing* of $x$ by

$$[\![x]\!] = \left(x^{[\![0]\!]}, \ldots, x^{[\![n-1]\!]}\right),$$

where $x^{[\![i]\!]}$ denotes the $i$-th *share* of $x$ for $i \in [n]$ and we have that

$$x = \sum_{i=0}^{n-1} x^{[\![i]\!]} \text{ (in } \mathbb{F} \text{ or } \mathcal{R}).$$

**Definition 2.8** (Multiparty Computation (MPC))**.** A *multiparty computation (MPC) protocol* is an interactive protocol executed among multiple parties $\mathcal{P}_0, \ldots, \mathcal{P}_{N-1}$. Each party $\mathcal{P}_i$ receives as input a share $x^{[\![i]\!]}$ of a secret value $x$ for $i \in [N]$. Together, the parties communicate in the MPC protocol to jointly compute a function $f(x)$. We denote the output of party $i$ by $f_i(x)$. This function should either output 1 or 0, corresponding to accept or reject, respectively.

We only consider *complete* MPC protocols throughout this dissertation, meaning that if all parties are honest, then the output of each party equals the correct output $f(x)$. The *view* of a party $\mathcal{P}_i$ during an execution includes its input share $x^{[\![i]\!]}$, its internal randomness, and all messages it receives from other parties during the protocol execution.
We recall the definition of $(N-1)$-private MPC protocols that we use in the MPC-in-the-Head setting. Note that this definition can be generalized to $t$-private MPC protocols for any $1 \leq t < N$, but we only work in the $(N-1)$-private setting in this dissertation.

**Definition 2.9** ($(N-1)$-Private)**.** Let $N > 1$ and consider an MPC protocol between $N$ parties $\mathcal{P}_0, \ldots, \mathcal{P}_{N-1}$ that jointly compute a function $f(x)$, where party $i$ gets as input the share $x^{[\![i]\!]}$ of $x$ for $i \in [N]$. We say that the protocol is $(N-1)$-private if for each index set $I \subset [N]$ of cardinality $|I| = N-1$, there exists a PPT algorithm $\mathcal{S}$ such that $\mathcal{S}(I, (x^{[\![i]\!]})_{i\in I}, (f_i(x))_{i\in I})$ is perfectly indistinguishable from the distribution of the views of the parties in $I$.

This definition captures the intuitive notion that the joint views of any subset of $N-1$ parties do not reveal any information about the secret input beyond what can already be derived from their own inputs and outputs. Therefore, as long as one party remains

honest, the secret value is protected. This property is particularly well suited for the MPCitH paradigm, where only $N-1$ simulated views are revealed to the verifier.

### 2.2.6 Digital Signatures

Digital signatures are one of the central primitives in modern public key cryptography. They provide authenticity, integrity, and non-repudiation for digital messages, enabling secure communications across untrusted networks. In essence, a digital signature scheme allows a user holding the secret key to sign a message such that anyone possessing the corresponding public key can verify the validity of the signature. In the context of this dissertation, digital signatures play a particularly important role, since the goal of the constructions in later chapters is to build secure signature schemes. We recall the standard formal definition of a digital signature and of its security properties.

**Definition 2.10** (Signature Scheme)**.** A *(digital) signature scheme* is a tuple of PPT algorithms $(\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$ such that

- $\texttt{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ takes as input the security parameter $1^\lambda$ and outputs a public key $\mathsf{pk}$ and a secret key $\mathsf{sk}$. We assume that the public key implicitly defines the message space $\mathcal{M}$.
- $\texttt{Sign}(\mathsf{sk}, \mathsf{msg}) \to \sigma$ takes as input the secret key $\mathsf{sk}$ and a message $\mathsf{msg} \in \mathcal{M}$ and outputs a signature $\sigma$.
- $\texttt{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma) \to b$ is a deterministic algorithm that takes as input a public key $\mathsf{pk}$, a message $\mathsf{msg} \in \mathcal{M}$ and a signature $\sigma$ and outputs a bit $b \in \{0, 1\}$.

We require that the signature scheme is correct, meaning that honestly generated signatures are declared valid by the $\texttt{Verify}$ algorithm:
**Correctness.** For any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \texttt{KeyGen}(1^\lambda)$ and any $\mathsf{msg} \in \mathcal{M}$ it holds that

$$\texttt{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma) = 1$$

for $\sigma \leftarrow \texttt{Sign}(\mathsf{sk}, \mathsf{msg})$.
To formally reason about the security of a digital signature scheme, we consider the adversary's capabilities in a game based setting. The standard security definition we require for a signature scheme is called *existential unforgeability under chosen message attacks* (or EUF-CMA). This notion models the strongest realistic scenario: an adversary can obtain signatures for arbitrary messages of its choice and should still be unable to forge a valid signature on a new message.

**Definition 2.11** (EUF-CMA Security)**.** We say that a signature scheme of the form $(\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$ is $\mathsf{EUF\text{-}CMA}$-secure, given the security parameter $\lambda$, if any PPT algorithm $\mathcal{A}$ has negligible advantage in the $\mathsf{EUF\text{-}CMA}$ security game, which is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} = \Pr\left[ \begin{array}{c|c} \texttt{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma) = 1 & (\mathsf{sk}, \mathsf{pk}) \leftarrow \texttt{KeyGen}(1^\lambda) \\ \wedge\, \mathsf{msg} \notin \mathcal{M}_q & (\mathsf{msg}, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}(\mathsf{sk},\mathsf{pk})}(\mathsf{pk}) \end{array} \right],$$

where the notation $\mathcal{A}^{\mathcal{O}(\mathsf{sk},\mathsf{pk})}$ means that $\mathcal{A}$ has access to a signing oracle $\mathcal{O}(\mathsf{sk}, \mathsf{pk})$ which has access to the private key $\mathsf{sk}$, and $\mathcal{M}_q$ denotes the set of messages that were queried by $\mathcal{A}$.

We also recall the weaker notion of *existential unforgeability under key-only attacks* (or EUF-KO). In this setting, the adversary only has access to the public key but not to a signing oracle.

**Definition 2.12** (EUF-KO Security)**.** A signature scheme of the form $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is existentially unforgeable under key-only attacks ($\mathsf{EUF\text{-}KO}$-secure) given the security parameter $\lambda$, if any PPT algorithm $\mathcal{A}$ has negligible advantage in the $\mathsf{EUF\text{-}KO}$ security game, which is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}KO}} = \Pr\left[ \mathtt{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma) = 1 \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(1^\lambda) \\ (\mathsf{msg}, \sigma) \leftarrow \mathcal{A}(\mathsf{pk}) \end{array} \right].$$

**The Fiat-Shamir Transformation.** A state-of-the-art method called Fiat-Shamir transformation [FS87] allows us to convert an interactive honest verifier zero knowledge proof of knowledge into a non-interactive signature scheme. This transformation removes the interaction between the prover and the verifier by deriving the challenges of the verifier in a deterministic way using the message $\mathsf{msg}$ and the transcript of the proof. The goal is to prevent a malicious signer from controlling the randomness used to generate the challenges $\mathtt{ch}_1, \ldots, \mathtt{ch}_n$. Following the Fiat-Shamir heuristic, we can compute $\mathtt{ch}_i$ based on a random oracle $\mathcal{H}_i$ (whose output size matches the respective challenge space size of $\mathtt{ch}_i$) as

$$\mathtt{ch}_i \leftarrow \mathcal{H}_i(\mathsf{msg} \parallel \mathtt{com} \parallel \mathtt{res}_1 \parallel \cdots \parallel \mathtt{res}_i).$$

The notation $\mathcal{H}_i$ means that we use domain separation for the random oracle $\mathcal{H}$ depending on the current round $i \in [n]$.

### 2.2.7 Hardness Assumptions

In this dissertation, we follow game-based security notions, which is the state-of-the-art for digital signature schemes. We analyze the asymptotic security of our proposed schemes in the random oracle model. In particular, we say that a scheme is secure if the advantage of any PPT adversary is negligible as a function of the security parameter $\lambda$.
Cryptographic security relies on the assumption that certain mathematical problems cannot be solved efficiently. These *hardness assumptions* are the foundation on which modern cryptographic schemes are built. Intuitively, it should be infeasible for an adversary with polynomially bounded computational resources to solve the underlying computational problem.
In particular, a hardness assumption defines a problem which no PPT algorithm can solve with non-negligible probability. Based on this assumption, we can reduce the security of a cryptographic construction to the hardness of an underlying computational problem. If an adversary could break the scheme, it could therefore also efficiently solve the intrinsic hard problem, contradicting the assumption.
Historically, the first public key cryptosystems did rely on mathematical assumptions from number theory. Examples include the *Integer Factorization Problem* used in RSA [RSA78] and the *Discrete Logarithm Problem* (DLP), which forms the basis from schemes such as the *Diffie-Hellman key exchange* [DH76] and the *Digital Signature Algorithm* (DSA) [SB93]. Later, the *Elliptic Curve Discrete Logarithm Problem* (ECDLP) [Mil86] provided a similar security level with smaller key sizes, enabling the efficiency of *elliptic curve cryptography* (ECC) and introducing schemes like the *Elliptic Curve Digital Signature Algorithm*

(ECDSA).
The emergence of quantum computing fundamentally challenges these classical assumptions. Peter Shor's algorithm [Sho94] can efficiently solve both factoring and discrete logarithms on a quantum computer, rendering RSA, DSA and ECDSA insecure in the post-quantum setting. This has resulted in the development of *post-quantum hardness assumptions*, which are believed to resist quantum attacks. Prominent examples include lattice-based assumptions [SAB+22, BBD+24], code-based assumptions [BBB+24a, BBB+24b], isogeny-based assumptions [AAA+24] and assumptions based on multivariate systems of nonlinear equations [BCC+24a, AFI+24, WCD+24, BCD+24]. In the particular cryptographic use case of digital signatures, another prominent class of schemes can be build from various hardness assumptions based on the MPC-in-the-Head paradigm. The digital signatures which we construct in this dissertation are all based on this general framework, which we introduce in Section 2.2.8. Afterwards, in Chapter 3, we introduce our underlying hardness assumption, called *Subfield Bilinear Collision* (SBC) problem, and demonstrate how we can use it in the MPC-in-the-Head paradigm to construct compact digital signature schemes.

### 2.2.8 MPCitH Paradigm

Our construction of signature schemes relies on the *MPC-in-the-Head* (MPCitH) paradigm introduced in [IKOS07]. This construction allows us to build a Zero-Knowledge proof of knowledge for an NP-relation $\mathcal{R}$. More precisely, assume we want to construct a Zero-Knowledge proof of knowledge of a witness $w$ for a statement $x$ with $(x, w) \in \mathcal{R}$. Consider an MPC protocol in which $N$ parties $\mathcal{P}_0, \ldots, \mathcal{P}_{N-1}$ securely and correctly compute the output of a function $f$ given a secret input $w$. The secret $w$ is given as a sharing $[\![w]\!]$ and party $\mathcal{P}_i$ receives the share $w^{[\![i]\!]}$. Additionally, the function $f$ should output either 1 or 0, corresponding to accept or reject, respectively. For our protocol, we also require that the views of $N-1$ parties do not reveal any information about $w$, i.e. that the protocol is $(N-1)$-private. This type of MPC protocol can be used to construct a Zero-Knowledge proof of a witness $w$ for which $f(w) = 1$. The prover proceeds as follows:

- generate a random sharing $[\![w]\!]$ of $w$,
- simulate privately ("in the head") all $N$ parties of the MPC protocol,
- send commitments to the views of each party to the verifier,
- send the output shares $[\![f(w)]\!]$ of the parties to the verifier.

The verifier then randomly chooses $N-1$ parties for which the prover has to reveal the views. Since the protocol is $(N-1)$-private, this does not reveal any information about the secret. The verifier can then check if these views are consistent with an honest execution of the MPC protocol as well as with the commitments from the prover. Since the choice of the $N-1$ opened parties was random, a malicious prover might be able to cheat with probability $\frac{1}{N}$ by corrupting the computation of one (unopened) party. Therefore, the Zero-Knowledge protocol constructed using this paradigm has a soundness error of at least $\frac{1}{N}$. To amplify this constant soundness error and to obtain our desired security level of $\lambda$ bits, we can use $\tau$ parallel repetitions of this protocol to obtain a soundness error of $\left(\frac{1}{N}\right)^{\tau} < 2^{-\lambda}$.

CHAPTER 3

# The Subfield Bilinear Collision Problem

This chapter is based on the first part of the Crypto 2024 publication [HJ24a] and its full version [HJ23]. I am the main author of this publication, under the supervision of my PhD advisor Antoine Joux. To achieve a consistent notation throughout this dissertation, there are some minor notational changes compared to the original version.

## 3.1 Introduction

In this chapter, we introduce the Subfield Bilinear Collision (SBC) problem, which yields a novel algebraic hardness assumption that serves as the security basis for our signature constructions in Chapter 4 and in Chapter 5. We formalize the problem, analyze its basic properties, discuss parameter choices and provide heuristic arguments supporting its hardness. We also explore potential attacks and its connection to the discrete logarithm problem in small characteristic finite fields.

## 3.2 Definition of the Problem

**Definition 3.1.** The SBC problem depends on three parameters: A prime power $q$ and two positive integers $k, n$.

- *SBC instance:* Two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, which are linearly independent over $\mathbb{F}_q$.
- *SBC solution:* Two linearly independent vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ such that

$$(\vec{u} \cdot \vec{x})\,(\vec{v} \cdot \vec{y}) = (\vec{u} \cdot \vec{y})\,(\vec{v} \cdot \vec{x})\,. \tag{3.1}$$

We denote an instance of the SBC problem given by the two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$ by $\mathrm{SBC}[\vec{u}, \vec{v}]$. If the vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ are a solution of $\mathrm{SBC}[\vec{u}, \vec{v}]$, we use the notation $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$.
Note that we use the canonical embedding from $\mathbb{F}_q$ to $\mathbb{F}_{q^k}$ for the vectors $\vec{x}$ and $\vec{y}$. Therefore, the operations in (3.1) are well-defined.

## 3.3 Basic Considerations about the SBC Problem

Before using the SBC problem as a cryptographic assumption, it is important to understand its structural properties and the intuition behind its hardness. In this section, we analyze basic aspects of the problem, including how collisions arise, how parameter choices influence difficulty, and how instances can be efficiently generated.

**Colinear collisions are easy to find.** For any vector $\vec{x} \in (\mathbb{F}_q)^n$ and for any scalar $\alpha \in \mathbb{F}_q$, we see that

$$(\vec{u} \cdot \vec{x})\,(\vec{v} \cdot (\alpha\vec{x})) = \alpha\,(\vec{u} \cdot \vec{x})\,(\vec{v} \cdot \vec{x}) = (\vec{u} \cdot (\alpha\vec{x}))\,(\vec{v} \cdot \vec{x})\,.$$

Therefore, the pair $(\vec{x}, \alpha\vec{x})$ always yields a collision. This is why we insist on linearly independent vectors $\vec{x}, \vec{y}$ in the definition of the SBC problem.

**Choice of parameters.** We examine the relation between the parameters $k$ and $n$ and the hardness of the corresponding instances of the SBC problem. Let $\mathrm{SBC}[\vec{u}, \vec{v}]$ be an instance of the SBC problem. By rewriting Equation (3.1), we want to find linearly independent vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ such that

$$\frac{\vec{u} \cdot \vec{x}}{\vec{v} \cdot \vec{x}} = \frac{\vec{u} \cdot \vec{y}}{\vec{v} \cdot \vec{y}},$$

assuming that $\vec{v} \cdot \vec{x} \neq 0$ and $\vec{v} \cdot \vec{y} \neq 0$. Consider the map

$$\begin{aligned} \phi\colon \mathbb{P}_{n-1}(\mathbb{F}_q) &\to \mathbb{F}_{q^k} \\ \vec{x} &\mapsto \frac{\vec{u} \cdot \vec{x}}{\vec{v} \cdot \vec{x}}. \end{aligned}$$

We do a heuristic analysis of the expected number of collisions of $\phi$. For simplicity, we assume that the range of $\phi$ is $\mathbb{F}_{q^k}^*$ and that the scalar product $\vec{v} \cdot \vec{x} \neq 0$ for every $\vec{x} \in \mathbb{P}_{n-1}(\mathbb{F}_q)$. We consider $Q \coloneqq \frac{q^n-1}{q-1} \approx q^{n-1}$ vectors in the domain of $\phi$. Therefore, there are

$$\binom{Q}{2} = \frac{Q(Q-1)}{2} \approx \frac{Q^2}{2}$$

possible pairs of vectors which could be part of a collision. Assuming that the range of $\phi$ is $\mathbb{F}_{q^k}^*$, the expected number of collisions depending on $k$ and $n$ can be estimated by

$$\frac{Q(Q-1)}{2(q^k-1)} \approx \frac{Q^2}{2q^k} \approx \frac{q^{2(n-1)}}{2q^k} = \frac{q^{2n-k-2}}{2}.$$

First, we consider the case $n > k+1$. Fix a random vector $\vec{x} \in \mathbb{P}_{n-1}(\mathbb{F}_q)$ and let $c = \phi(\vec{x})$. Finding a collision $(\vec{x}, \vec{y})$ is then equivalent to finding a vector $\vec{y} \in \mathbb{P}_{n-1}(\mathbb{F}_q)$ with $\vec{y} \neq \vec{x}$ such that

$$(c\vec{v} - \vec{u}) \cdot \vec{y} = 0.$$

Each entry of the vectors $\vec{u}$ and $\vec{v}$ is an element of $\mathbb{F}_{q^k}$. Let $(\alpha_0, \ldots, \alpha_{k-1})$ be an $\mathbb{F}_q$ basis of $\mathbb{F}_{q^k}$. We can express each entry of $(c\vec{v} - \vec{u})$ using this basis to obtain a matrix of the form

$$\mathrm{M} = \begin{pmatrix} a_0^{(0)} & \ldots & a_{n-1}^{(0)} \\ \vdots & \ddots & \vdots \\ a_0^{(k-1)} & \ldots & a_{n-1}^{(k-1)} \end{pmatrix} \in \mathbb{F}_q^{k \times n},$$

where $cv_i - u_i = \sum_{j=0}^{k-1} a_i^{(j)} \alpha_j$ for $i = 0, \ldots, n-1$. By construction, $\vec{x}$ is in the right kernel of M. To obtain a collision, the goal is to find another vector $\vec{y}$ in the right kernel of M with $\vec{y} \neq \vec{x}$ in $\mathbb{P}_{n-1}(\mathbb{F}_q)$. If $n > k+1$, such a vector $\vec{y}$ has to exist and can be computed by solving the linear system $\mathrm{M}\vec{y} = 0$.
Next, consider the case $n \leq k+1$. We pick $N$ pairwise non-colinear vectors $\vec{x}_0, \ldots, \vec{x}_{N-1}$ and want to estimate the probability that at least one of these $N$ distinct vectors is part of a collision with the remaining $Q - N \approx Q$ vectors. If one of these $N$ vectors, say $\vec{x}_j$, is part of a collision, we can compute $c_j = \phi(\vec{x}_j)$ and again solve the linear system

$$(c_j \vec{v} - \vec{u}) \cdot \vec{y} = 0$$

to find the vector $\vec{y}$. Following a standard heuristic argument, we expect such a collision to appear if $NQ > q^k$, i.e. if $N > q^{k+1-n}$. By using this technique, we get an attack against the SBC problem in $\mathcal{O}(q^{k+1-n})$, which is quite efficient for $n$ close to $k$.
If we consider $n \ll \frac{k}{2}$, we do not expect any collisions to exist. Therefore, a setup where $n \approx \frac{k}{2}$ seems to be a good parameter choice for the SBC problem: We expect collisions to exist, but the technique for finding collisions described above is not efficient.

### 3.3.1 Normalization of Solutions

Consider a solution $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$ of the SBC problem. We define the matrix

$$N = \begin{pmatrix} x_{n-2} & y_{n-2} \\ x_{n-1} & y_{n-1} \end{pmatrix} \in \mathbb{F}_q^{2\times 2}.$$

First, we consider the case that $N$ is singular. Let $\vec{K} \neq (0,0)$ be an element of the left kernel of $N$, i.e. $\vec{K} \cdot (x_{n-2}, x_{n-1}) = 0$ and $\vec{K} \cdot (y_{n-2}, y_{n-1}) = 0$. We distinguish two cases:

- $\vec{K} \in \langle (1,0) \rangle$: In this case, the entries $x_{n-2}$ and $y_{n-2}$ do not contribute towards the sum in the scalar product computations. Therefore, we could decrease the size of the vectors $\vec{u}$ and $\vec{v}$ from $n$ to $n-1$ by removing the entries $u_{n-2}$ and $v_{n-2}$ from $\vec{u}$ and $\vec{v}$ respectively. Afterwards, we can still obtain a solution $\left(\vec{x'}, \vec{y'}\right)$ to the SBC problem with smaller dimension $n-1$ by removing the entries $x_{n-2}$ and $y_{n-2}$ from $\vec{x}$ and $\vec{y}$ respectively.

- $\vec{K} \in \langle (\alpha, 1) \rangle$ for an $\alpha \in \mathbb{F}_q$: For simplicity, we normalize $\vec{K}$ to be of the form $\vec{K} = (c, 1)$. Then the following equations hold:

$$\begin{aligned}
(u_{n-2}, u_{n-1}) \cdot (x_{n-2}, x_{n-1}) &= (u_{n-2}, u_{n-1}) \cdot (x_{n-2}, x_{n-1}) - u_{n-1}\vec{K} \cdot (x_{n-2}, x_{n-1}), \\
(u_{n-2}, u_{n-1}) \cdot (y_{n-2}, y_{n-1}) &= (u_{n-2}, u_{n-1}) \cdot (y_{n-2}, y_{n-1}) - u_{n-1}\vec{K} \cdot (y_{n-2}, y_{n-1}), \\
(v_{n-2}, v_{n-1}) \cdot (x_{n-2}, x_{n-1}) &= (v_{n-2}, v_{n-1}) \cdot (x_{n-2}, x_{n-1}) - v_{n-1}\vec{K} \cdot (x_{n-2}, x_{n-1}), \\
(v_{n-2}, v_{n-1}) \cdot (y_{n-2}, y_{n-1}) &= (v_{n-2}, v_{n-1}) \cdot (y_{n-2}, y_{n-1}) - v_{n-1}\vec{K} \cdot (y_{n-2}, y_{n-1}).
\end{aligned}$$

  If we replace $u_{n-2}$ by $(u_{n-2} - cu_{n-1})$ and $v_{n-2}$ by $(v_{n-2} - cv_{n-1})$, we can remove the entries $u_{n-1}$ and $v_{n-1}$ from $\vec{u}$ and $\vec{v}$ respectively to obtain a new instance of the SBC problem with smaller dimension $n-1$. This new system still has a solution of the form $\vec{x} = (x_0, \ldots, x_{n-2})$ and $\vec{y} = (y_0, \ldots, y_{n-2})$.

In both cases, we can reduce the dimension $n$ of the vectors but still obtain a solution of the SBC problem if $N$ is singular. Since we do not want the dimension of the problem to be easily reduced, we assume that the matrix $N$ is non-singular when we consider solutions of the SBC problem.
Assume that $N$ is non-singular. Then, we can simplify the solutions in the following way: Compute the inverse of $N$, which we denote by

$$W = \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix} \in \mathbb{F}_q^{2\times 2}.$$

We can use $W$ to normalize the solution vectors $\vec{x}, \vec{y}$ and obtain

$$W \cdot \begin{pmatrix} \vec{x} \\ \vec{y} \end{pmatrix} = \begin{pmatrix} \vec{x'} & 1 & 0 \\ \vec{y'} & 0 & 1 \end{pmatrix}$$

for some $\vec{x'}, \vec{y'} \in (\mathbb{F}_q)^{n-2}$. The normalized vectors $\left(\left(\vec{x'}, 1, 0\right), \left(\vec{y'}, 0, 1\right)\right)$ are also a solution of the same SBC problem SBC$[\vec{u}, \vec{v}]$. To see that, we check the equation

$$\begin{aligned}
\left[\vec{u} \cdot \left(\vec{x'}, 1, 0\right)\right] \left[\vec{v} \cdot \left(\vec{y'}, 0, 1\right)\right] &= \left[\vec{u} \cdot (w_{00}\vec{x} + w_{10}\vec{y})\right] \left[\vec{v} \cdot (w_{10}\vec{x} + w_{11}\vec{y})\right] \\
&= w_{00}w_{10} \left(\vec{u} \cdot \vec{x}\right) \left(\vec{v} \cdot \vec{x}\right) + w_{00}w_{11} \left(\vec{u} \cdot \vec{x}\right) \left(\vec{v} \cdot \vec{y}\right) \\
&\quad + w_{01}w_{10} \left(\vec{u} \cdot \vec{y}\right) \left(\vec{v} \cdot \vec{x}\right) + w_{01}w_{11} \left(\vec{u} \cdot \vec{y}\right) \left(\vec{v} \cdot \vec{y}\right) \\
&= w_{10}w_{00} \left(\vec{u} \cdot \vec{x}\right) \left(\vec{v} \cdot \vec{x}\right) + w_{11}w_{00} \left(\vec{v} \cdot \vec{y}\right) \left(\vec{u} \cdot \vec{x}\right) \\
&\quad + w_{10}w_{01} \left(\vec{v} \cdot \vec{x}\right) \left(\vec{u} \cdot \vec{y}\right) + w_{11}w_{01} \left(\vec{v} \cdot \vec{y}\right) \left(\vec{u} \cdot \vec{y}\right) \\
&= \left[\vec{u} \cdot (w_{10}\vec{x} + w_{11}\vec{y})\right] \left[\vec{v} \cdot (w_{00}\vec{x} + w_{01}\vec{y})\right] \\
&= \left[\vec{u} \cdot \left(\vec{y'}, 0, 1\right)\right] \left[\vec{v} \cdot \left(\vec{x'}, 1, 0\right)\right].
\end{aligned}$$

Therefore, $\left(\left(\vec{x'}, 1, 0\right), \left(\vec{y'}, 0, 1\right)\right) \in$ SBC$[\vec{u}, \vec{v}]$. In the third equation, we used the commutativity of the multiplication in $\mathbb{F}_{q^k}$ and the fact that $(\vec{x}, \vec{y}) \in$ SBC$[\vec{u}, \vec{v}]$. By this argument, we can always normalize a solution of the SBC problem to obtain vectors of the form $\vec{x} = \left(\vec{x'}, 1, 0\right)$ and $\vec{y} = \left(\vec{y'}, 0, 1\right)$. In this case, $\vec{x}$ and $\vec{y}$ are non-colinear. Throughout this dissertation, we therefore assume any solution of the SBC problem to be of this normalized form.

**Definition 3.2** (The Normalized Subfield Bilinear Collision (NSBC) problem.)**.**

- *NSBC instance:* Two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, which are linearly independent over $\mathbb{F}_q$.
- *NSBC solution:* Two vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ of the form $\vec{x} = \left(\vec{x'}, 1, 0\right)$, $\vec{y} = \left(\vec{y'}, 0, 1\right)$ such that

$$\left(\vec{u} \cdot \vec{x}\right) \left(\vec{v} \cdot \vec{y}\right) = \left(\vec{u} \cdot \vec{y}\right) \left(\vec{v} \cdot \vec{x}\right).$$

Similar to before, we denote an instance of the NSBC problem given by the two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$ by NSBC$[\vec{u}, \vec{v}]$. If the vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ are a solution of NSBC$[\vec{u}, \vec{v}]$, we use the notation $(\vec{x}, \vec{y}) \in$ NSBC$[\vec{u}, \vec{v}]$.

### 3.3.2 Generation of Normalized Instances with a Solution

Given $q, k$ and $n$, we can generate an instance of NSBC that has a solution in the following way: Uniformly at random choose two vectors $\vec{x'}, \vec{y'} \in (\mathbb{F}_q)^{n-2}$, set $\vec{x} \coloneqq \left(\vec{x'}, 1, 0\right)$ and $\vec{y} \coloneqq \left(\vec{y'}, 0, 1\right)$. Also, randomly choose the coordinates $u_0, \ldots, u_{n-1} \in \mathbb{F}_{q^k}$ and $v_0, \ldots, v_{n-2} \in \mathbb{F}_{q^k}$ of $\vec{u}$ and $\vec{v}$. Lastly, compute $v_{n-1}$ as

$$v_{n-1} = \frac{(\vec{u} \cdot \vec{y}) \left(\sum_{i=0}^{n-2} v_i x_i\right) - (\vec{u} \cdot \vec{x}) \left(\sum_{i=0}^{n-2} v_i y_i\right)}{y_{n-1} (\vec{u} \cdot \vec{x})}.$$

If the denominator $y_{n-1} (\vec{u} \cdot \vec{x})$ happens to be zero, the computation of $v_{n-1}$ fails. In this case, we can choose new random vectors and start the computation again or we can change the value of $u_{n-2}$ and compute $v_{n-1}$ successfully. If the unlikely case that $\vec{u}, \vec{v}$ are linearly dependent over $\mathbb{F}_q$ occurs, we can easily adapt the vectors accordingly to avoid this case. By this construction, we have that $(\vec{x}, \vec{y}) \in \mathrm{NSBC}[\vec{u}, \vec{v}]$. The one-way function $\tilde{f}$ which we use for the identification protocol for SBC is therefore of the form

$$\tilde{f}(u_0, \ldots, u_{n-1}, v_0, \ldots, v_{n-2}, \vec{x}, \vec{y}) = (u_0, \ldots, u_{n-1}, v_0, \ldots, v_{n-2}, v_{n-1}).$$

### 3.3.3 Origin of the SBC Problem

Consider the discrete logarithm problem (DLP) in the group $G = \mathbb{F}_{q^k}^*$ for some large extension degree $k$. Let $\mathbb{F}_q[X]$ denote the set of polynomials in $X$ with coefficients in $\mathbb{F}_q$. We can represent $\mathbb{F}_{q^k}^*$ as the quotient ring $\mathbb{F}_q[X]/(I_k(X))$, where $I_k$ is an arbitrary irreducible polynomial of degree $k$.
We briefly describe a heuristic algorithm to solve the DLP in this group, which is based on the function field sieve. This family of algorithms, called *Frobenius Representation algorithms*, was for example studied in [Jou14].

In the representation phase of the algorithm from [Jou14], the goal is to find two polynomials $h_0, h_1 \in \mathbb{F}_q[X]$ of degree at most 2 such that there exists an irreducible polynomial $I_k$ of degree $k > 2$ with $I_k(X) | (h_1(X)X^q - h_0(X))$. Let $\theta$ be a root of $I_k$ in the algebraic closure of $\mathbb{F}_q$. Using $I_k$, we can represent $\mathbb{F}_{q^k} \cong \mathbb{F}_q[\theta]$ as $\mathbb{F}_q[X]/(I_k(X))$. In this representation, we see that $\theta^q = \frac{h_0(\theta)}{h_1(\theta)}$. We follow the approach in [Jou14]: In the descent phase of the algorithm, the goal is to find the discrete logarithm of an element $T(\theta) \in \mathbb{F}_{q^k}$. Without loss of generality, we can assume that $T$ is an irreducible polynomial over $\mathbb{F}_q[X]$.[1] Let $\xi \in \mathbb{F}_{q^{\deg(T)}}$ be a root of $T$. We try to find polynomials $A, B \in \mathbb{F}_q[X, Y]$ of degree $d \approx k/2$ such that

$$B(\xi, 1)A(h_0(\xi), h_1(\xi)) - A(\xi, 1)B(h_0(\xi), h_1(\xi)) = 0 \quad \text{in } \mathbb{F}_{q^{\deg(T)}},$$

or equivalently

$$T(x) | [B(x, 1)A(h_0(x), h_1(x)) - A(x, 1)B(h_0(x), h_1(x))],$$

[1] Otherwise, we can factor $T$ into irreducible polynomials and consider each factor individually.

where $A$ and $B$ are the two homogeneous polynomials

$$A(x,y) \coloneqq \sum_{i=0}^{d} a_i x^i y^{d-i} \quad \text{and} \quad B(x,y) \coloneqq \sum_{i=0}^{d} b_i x^i y^{d-i}.$$

This is in fact a special case of the SBC problem: Define

$$\vec{\mathcal{H}} \coloneqq \begin{pmatrix} h_1(\xi)^d \\ h_1(\xi)^{d-1} h_0(\xi) \\ \vdots \\ h_0(\xi)^d \end{pmatrix} \quad \text{and} \quad \vec{\mathcal{G}} \coloneqq \begin{pmatrix} 1 \\ \xi \\ \vdots \\ \xi^d \end{pmatrix}.$$

Then the goal is to find two non-colinear vectors $\vec{a}, \vec{b} \in (\mathbb{F}_q)^d$ with

$$\left(\vec{\mathcal{H}} \cdot \vec{a}\right)\left(\vec{\mathcal{G}} \cdot \vec{b}\right) - \left(\vec{\mathcal{H}} \cdot \vec{b}\right)\left(\vec{\mathcal{G}} \cdot \vec{a}\right) = 0 \quad \text{in } \mathbb{F}_{q^{\deg(T)}},$$

i.e. $\left(\vec{a}, \vec{b}\right) \in \mathrm{SBC}[\vec{\mathcal{H}}, \vec{\mathcal{G}}]$. In this case, the vectors $\vec{a}, \vec{b}$ represent the coefficients of the polynomials $A$ and $B$, respectively. In [JP22], the limiting factor of the discrete logarithm computation comes from this variant of the SBC problem. Improving attacks against the SBC problem would therefore lead to improvements of the discrete logarithm computations using the method described above.

### 3.3.4 Known Attacks

Consider an instance $\mathrm{NSBC}[\vec{u}, \vec{v}]$ of the normalized SBC problem. By rewriting Equation (3.1), the goal is to find vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ of the form $\vec{x} = \left(\vec{x'}, 1, 0\right)$, $\vec{y} = \left(\vec{y'}, 0, 1\right)$ such that $(\vec{u} \cdot \vec{x})(\vec{v} \cdot \vec{y}) - (\vec{u} \cdot \vec{y})(\vec{v} \cdot \vec{x}) = 0$. This is a bilinear equation given by the polynomial

$$\begin{aligned} g(x_0, \ldots, x_{n-3}, y_0, \ldots, y_{n-3}) \coloneqq & \left(\sum_{i=0}^{n-3} u_i x_i + u_{n-2}\right)\left(\sum_{i=0}^{n-3} v_i y_i + v_{n-1}\right) \\ & - \left(\sum_{i=0}^{n-3} u_i y_i + u_{n-1}\right)\left(\sum_{i=0}^{n-3} v_i x_i + v_{n-2}\right). \end{aligned}$$

Since $\vec{u}, \vec{v} \in (\mathbb{F}_{q^k})^n$, the polynomial $g$ is an element of the polynomial ring

$$\mathbb{F}_{q^k}[X_0, \ldots, X_{n-3}, Y_0, \ldots, Y_{n-3}]$$

in $2(n-2)$ variables. We can express this polynomial with $k$ polynomials in the base field $\mathbb{F}_q$ by choosing an $\mathbb{F}_q$ basis $(\alpha_0, \ldots, \alpha_{k-1})$ of $\mathbb{F}_{q^k}$. Therefore, we obtain a system of $k$ bilinear equations of the form

$$\begin{aligned} g_0(x_0, \ldots, x_{n-3}, y_0, \ldots, y_{n-3}) &= 0, \\ &\vdots \\ g_{k-1}(x_0, \ldots, x_{n-3}, y_0, \ldots, y_{n-3}) &= 0, \end{aligned}$$

where $g_i \in \mathbb{F}_q[X_0, \ldots, X_{n-3}, Y_0, \ldots, Y_{n-3}]$ for $i \in [k]$. We can solve this bilinear system using Gröbner basis algorithms, for example the so-called $F_5$ algorithm [Fau02]. For details about Gröbner basis algorithms, see for example [FSS11, Spa12]. We recall the following result from [FSS11]:

**Theorem 3.3** ([FSS11, Corollary 3])**.** The complexity of computing a Gröbner basis of a generic bilinear system $g_0, \ldots, g_{n_x+n_y-1} \in \mathbb{K}[x_0, \ldots, x_{n_x-1}, y_0, \ldots, y_{n_y-1}]$ with the $F_5$ algorithm is upper bounded by

$$\mathcal{O}\left(\binom{n_x - 1 + n_y - 1 + \min(n_x, n_y)}{\min(n_x, n_y)}^{\omega}\right),$$

where $2 \leq \omega \leq 3$ is the linear algebra constant.

In our setting, we have that $n_x = n_y \approx \frac{k}{2}$, which means we can upper bound the complexity of solving the obtained system of bilinear equations by

$$\mathcal{O}\left(\binom{\frac{3k}{2}}{\frac{k}{2}}^{\omega}\right).$$

By using Stirling's formula, we can asymptotically estimate the binomial coefficient by $2^{H\left(\frac{1}{3}\right)\frac{3k}{2}}$, where $H(p) = -p \log p - (1-p)\log(1-p)$ is the *binary entropy function*, which yields an estimate of roughly $2^{1.38k\omega}$ for the upper bound of the complexity. For a security level of $\lambda = n = 128$ and the best case scenario for the linear algebra constant of $\omega = 2$, this would yield a complexity of $2^{706}$.
The upper bound given in Theorem 3.3 is not tight, there are practical examples of Gröbner basis computations which are faster. In [JP22, Section 6], a discrete logarithm computation based on the same bilinear system is feasible to compute up to the extension degree of $k = 36$. However, our suggested parameter choice of $k \geq 256$ seems completely out of reach of these techniques.
**Evidence for post-quantum security.** Concerning the post-quantum security of the SBC problem, we do not see any efficient attack that would outperform a standard attack based on Grover's algorithm [Gro96]. Other post-quantum systems like MinRank [BFS96] rely on the hardness of solving bilinear systems. Indeed, the MinRank problem can be transformed into a bilinear system using the Support-Minor modeling [BBC+20].
Moreover, the resolution of general bilinear systems is NP-complete [DPS20]. Despite the relation of the SBC problem to the discrete logarithm problem shown in Section 3.3.3, the SBC problem is not reducible to discrete logarithms as far as we know. Therefore, SBC cannot be directly attacked using Shor's algorithm [Sho94].

CHAPTER 4

# MPC-in-the-Head Signatures from SBC

This chapter is based on the second part of the Crypto 2024 publication [HJ24a] and its full version [HJ23]. I am the main author of this publication, under the supervision of my PhD advisor Antoine Joux. To achieve a consistent notation throughout this dissertation, there are some minor notational changes compared to the original version.

## 4.1 Introduction

In this chapter, we investigate how the Subfield Bilinear Collision (SBC) problem can be used within the MPCitH framework. Due to its algebraic structure, SBC is particularly well suited for MPCitH protocols, leading to compact proofs that translate directly into digital signatures via the Fiat-Shamir transform.

## 4.2 Contribution

We present the first cryptographic constructions based on the SBC problem. We show that the algebraic structure of SBC fits naturally into the MPCitH paradigm and leads to compact post-quantum signatures. We show how to test for solutions of SBC using a certain class of linear polynomials over finite fields and how to use this property in an interactive Zero-Knowledge proof of knowledge of an SBC solution. We show how to use the Fiat-Shamir transformation to obtain a signature scheme based on the hardness of SBC.

We provide a performance comparison with existing MPCitH based signatures and demonstrate how to optimize the signature computation to improve size and running times. For that, we introduce a fast hypercube folding algorithm with running time $\mathcal{O}(N)$ that accelerates the folding process to significantly improve the practical performance of our scheme. With our improvements, we can achieve signature sizes below 3800 bytes while having signing and verification times around 20 ms for a security level of $\lambda = 128$ bits.

## 4.3 Multi-Party Computations (MPC)

We use additive sharings of finite field elements. Let $N$ be the number of parties. Then an *$N$-sharing* of a finite field element $x \in \mathbb{F}$ is an $N$-tuple

$$[\![x]\!] = \left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right)$$

such that

$$x = \sum_{i=0}^{N-1} x^{[\![i]\!]} \mod |\mathbb{F}|.$$

We call each $x^{[\![i]\!]}$ a *share* of $x$. In the MPC protocol that we use, each party receives one of the $N$ shares. With these shares, the parties can then perform computations independently: Assume each party $i \in [N]$ receives the shares $x^{[\![i]\!]}$ and $y^{[\![i]\!]}$ corresponding to sharings of $x$ and $y$. Let $\alpha$ be a constant. Then the parties can perform the following operations:

- **Addition:** they locally compute $[\![x+y]\!]$ by adding their shares:

  $$(x+y)^{[\![i]\!]} \coloneqq x^{[\![i]\!]} + y^{[\![i]\!]}$$

  for $i \in [N]$. We denote this by $[\![x+y]\!] = [\![x]\!] + [\![y]\!]$.

- **Multiplication by a constant:** they locally compute $[\![\alpha x]\!]$ by multiplying their respective shares by $\alpha$:

  $$(\alpha x)^{[\![i]\!]} \coloneqq \alpha x^{[\![i]\!]}$$

  for $i \in [N]$. We denote this by $[\![\alpha x]\!] = \alpha[\![x]\!]$.

- **Adding a constant:** the constant $\alpha$ can be shared in a trivial way as $[\![\alpha]\!]_T \coloneqq (\alpha, 0, \ldots, 0)$, where the subscript indicates a trivial sharing. The parties then compute $[\![x+\alpha]\!] = [\![x]\!] + [\![\alpha]\!]_T$. We denote this by $[\![x+\alpha]\!] = [\![x]\!] + \alpha$.

In practice, a sharing of $x$ is usually computed by choosing $N-1$ random values $x^{[\![0]\!]}, \ldots, x^{[\![N-2]\!]}$ and by setting

$$x^{[\![N-1]\!]} = x - \sum_{i=0}^{N-2} x^{[\![i]\!]} \mod |\mathbb{F}|$$

afterwards. Then the $N$-tuple $\left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right)$ is actually a sharing of $x$. In this dissertation, we instead mostly use completely random sharings by using $N$ random values $R_x^{[\![0]\!]}, \ldots, R_x^{[\![N-1]\!]}$ in our MPC protocol to obtain a sharing of $x$. If we do that, we need an *auxiliary value* or *offset*

$$\delta_x \coloneqq x - \sum_{i=0}^{N-1} R_x^{[\![i]\!]} \mod |\mathbb{F}|,$$

to reconstruct $x$. The value of $x$ can then be obtained by calculating the sum

$$x = \delta_x + \sum_{i=0}^{N-1} R_x^{[\![i]\!]} \mod |\mathbb{F}|.$$

In the usual setting, this can be viewed as a sharing of the value $x$ between $N+1$ parties. By using auxiliary values of the form $\delta_x$ instead, we can simplify the notation which we need in the identification scheme in Section 4.4 and choose random values for every share.

## 4.4 Main Zero-Knowledge Protocol

In this section, we describe an MPC protocol based on the SBC problem. Let $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, consider the NSBC problem NSBC$[\vec{u}, \vec{v}]$ and consider normalized vectors $\vec{x} = \left(\vec{x'}, 1, 0\right)$, $\vec{y} = \left(\vec{y'}, 0, 1\right) \in (\mathbb{F}_q)^n$. Our protocol is based on the fact that we can test for solutions of SBC using a certain class of polynomials.
Let $X_1, X_2, Y_1, Y_2 \in \mathbb{F}_{q^k}$ be random values. Consider the following polynomial in $t$:

$$\begin{aligned} F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t) &= (X_1 + t\,(\vec{u}\cdot\vec{x}))\,(Y_1 + t\,(\vec{v}\cdot\vec{y})) - (X_2 + t\,(\vec{v}\cdot\vec{x}))\,(Y_2 + t\,(\vec{u}\cdot\vec{y})) \\ &= X_1Y_1 - X_2Y_2 \\ &\quad + [X_1\,(\vec{v}\cdot\vec{y}) + Y_1\,(\vec{u}\cdot\vec{x}) - X_2\,(\vec{u}\cdot\vec{y}) - Y_2\,(\vec{v}\cdot\vec{x})]\,t \\ &\quad + [(\vec{u}\cdot\vec{x})\,(\vec{v}\cdot\vec{y}) - (\vec{u}\cdot\vec{y})\,(\vec{v}\cdot\vec{x})]\,t^2. \end{aligned}$$

For this polynomial, we see that $\deg\left(F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}\right) < 2$ if and only if $(\vec{x}, \vec{y}) \in$ NSBC$[\vec{u}, \vec{v}]$. In that case, we can write

$$F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t) = A + Bt$$

with $A = A_{X_1,X_2,Y_1,Y_2} = X_1Y_1 - X_2Y_2$ and $B = B^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2} = X_1\,(\vec{v}\cdot\vec{y}) + Y_1\,(\vec{u}\cdot\vec{x}) - X_2\,(\vec{u}\cdot\vec{y}) - Y_2\,(\vec{v}\cdot\vec{x})$. This property of $F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t)$ is a crucial part of our MPC protocol.

### 4.4.1 Basic Multiparty Protocol for the SBC Problem

In this protocol, we share the two coefficients $A$ and $B$ and want to prove that the polynomial $F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}$ has indeed the form $F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t) = A + Bt$, which means that $(\vec{x}, \vec{y}) \in$ NSBC$[\vec{u}, \vec{v}]$. Let $\mathcal{P}_0, \ldots, \mathcal{P}_{N-1}$ be a group of $N$ provers that want to verify that $(\vec{x}, \vec{y})$ is a solution of NSBC$[\vec{u}, \vec{v}]$. The basic protocol contains the following steps:

- The parties run an auxiliary protocol to obtain a random value $t_0 \in \mathbb{F}_{q^k}$.
- The parties evaluate $F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}$ at $t_0$.
- The parties evaluate $A + Bt_0$.
- The parties output accept if $F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t_0) = A + Bt_0$ and reject otherwise.

Assume that $\vec{x}$ and $\vec{y}$ are shared as

$$\begin{aligned} \vec{x} &= \vec{\delta}_x + \sum_{i=0}^{N-1} \vec{R}_x^{\,[\![i]\!]}, \\ \vec{y} &= \vec{\delta}_y + \sum_{i=0}^{N-1} \vec{R}_y^{\,[\![i]\!]} \end{aligned}$$

and the coefficients $A$ and $B$ are shared as

$$A = \delta_A + \sum_{i=0}^{N-1} R_A^{[\![i]\!]},$$

$$B = \delta_B + \sum_{i=0}^{N-1} R_B^{[\![i]\!]}.$$

We use (exact) sharings given by $[\![X_1]\!], [\![X_2]\!], [\![Y_1]\!], [\![Y_2]\!]$ for the random values $X_1, X_2, Y_1, Y_2$. Since we only consider normalized solutions of the SBC problem, the last two entries of $\vec{x}$ and $\vec{y}$ can be excluded from the sharing as they are known already. In particular, the protocol runs as follows:

- Each party $i \in [N]$ receives their private input values

  $$\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, X_1^{[\![i]\!]}, X_2^{[\![i]\!]}, Y_1^{[\![i]\!]}, Y_2^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}$$

  as well as the (public) auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$.

- Each party $\mathcal{P}_i$ chooses a random $t_0^{[\![i]\!]} \in \mathbb{F}_{q^k}$ and all parties simultaneously broadcast their shares.

- The parties reconstruct $t_0 \coloneqq \sum_{i=0}^{N-1} t_0^{[\![i]\!]}$.

- Each party $i \in [N]$ computes

  $$\begin{aligned}
  (X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]} &= X_1^{[\![i]\!]} + t_0 \left(\vec{u} \cdot \vec{R_x}^{[\![i]\!]}\right), \\
  (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]} &= X_2^{[\![i]\!]} + t_0 \left(\vec{v} \cdot \vec{R_x}^{[\![i]\!]}\right), \\
  (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]} &= Y_1^{[\![i]\!]} + t_0 \left(\vec{v} \cdot \vec{R_y}^{[\![i]\!]}\right), \\
  (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]} &= Y_2^{[\![i]\!]} + t_0 \left(\vec{u} \cdot \vec{R_y}^{[\![i]\!]}\right), \\
  (A + Bt_0)^{[\![i]\!]} &= R_A^{[\![i]\!]} + R_B^{[\![i]\!]} t_0
  \end{aligned}$$

  and broadcasts their shares.

- The parties reconstruct

  $$\begin{aligned}
  X_1 + t_0 (\vec{u} \cdot \vec{x}) &= \sum_{i=0}^{N-1} (X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]} + t_0 u_{n-2} + t_0 \left(\vec{u} \cdot \vec{\delta}_x\right), \\
  X_2 + t_0 (\vec{v} \cdot \vec{x}) &= \sum_{i=0}^{N-1} (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]} + t_0 v_{n-2} + t_0 \left(\vec{v} \cdot \vec{\delta}_x\right), \\
  Y_1 + t_0 (\vec{v} \cdot \vec{y}) &= \sum_{i=0}^{N-1} (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]} + t_0 v_{n-1} + t_0 \left(\vec{v} \cdot \vec{\delta}_y\right), \\
  Y_2 + t_0 (\vec{u} \cdot \vec{y}) &= \sum_{i=0}^{N-1} (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]} + t_0 u_{n-1} + t_0 \left(\vec{u} \cdot \vec{\delta}_y\right), \\
  A + Bt_0 &= \sum_{i=0}^{N-1} (A + Bt_0)^{[\![i]\!]} + \delta_A + \delta_B t_0.
  \end{aligned}$$

The parties output accept if

$$\begin{aligned} A + Bt_0 &= F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t_0) \\ &= (X_1 + t_0\,(\vec{u}\cdot\vec{x}))\,(Y_1 + t_0\,(\vec{v}\cdot\vec{y})) - (X_2 + t_0\,(\vec{v}\cdot\vec{x}))\,(Y_2 + t_0\,(\vec{u}\cdot\vec{y})) \end{aligned}$$

and reject otherwise.

The protocol is correct: Every party accepts if the shares are correct and if each party is honest. Assume $(\vec{x},\vec{y}) \notin \mathrm{NSBC}[\vec{u},\vec{v}]$. Then the protocol accepts this wrong instance of vectors if $t_0$ is a root of the degree 2 polynomial

$$(A + Bt) - F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t).$$

If at least one of the $N$ parties is honest, the value of $t_0$ obtained in the protocol is random. Therefore, the probability of accepting a wrong pair of vectors $(\vec{x},\vec{y})$ is bounded by $\frac{2}{q^k}$.

We also note that the protocol is $(N-1)$-private with respect to the inputs $\vec{x},\vec{y}$. Indeed, if the sharing of the values is performed uniformly at random, no information about the secret vectors $\vec{x},\vec{y}$ is revealed if at least one of the $N$ parties is honest. After every run of the protocol, the dealer should choose fresh random values for $X_1, X_2, Y_1, Y_2$.

**The view of each party.** During the protocol execution, each party keeps a view of their computation. The view of party $i$ consists of the following six elements:

$$\begin{aligned} \mathrm{View}_i = (&t_0^{[\![i]\!]}, (X_1 + t_0\,(\vec{u}\cdot\vec{x}))^{[\![i]\!]}, (X_2 + t_0\,(\vec{v}\cdot\vec{x}))^{[\![i]\!]}, \\ &(Y_1 + t_0\,(\vec{v}\cdot\vec{y}))^{[\![i]\!]}, (Y_2 + t_0\,(\vec{u}\cdot\vec{y}))^{[\![i]\!]}, (A + Bt_0)^{[\![i]\!]}). \end{aligned}$$

**Preventing adversarial choice of the evaluation point.** We want to adapt the basic MPC protocol to be able to use it in an identification protocol. If the parties do not broadcast their shares simultaneously, a malicious party can wait until it receives all the other shares of $t_0$ and choose its share such that $t_0$ is a root of the polynomial

$$(A + Bt) - F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t).$$

In this section, we adapt the protocol to prevent this. For a first new version, instead of randomly choosing the value of $t_0^{[\![i]\!]}$, party $i$ uses the random oracle $\mathcal{H}$ on their input shares and the public auxiliary values to compute

$$t_0^{[\![i]\!]} = \mathcal{H}_t\left(i \parallel \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B\right). \tag{4.1}$$

Here, the subscript $t$ indicates that we use domain separation for the random oracle $\mathcal{H}$ to derive $t_0$. In particular, for any input string $\mathtt{str}$, define $\mathcal{H}_t(\mathtt{str}) \coloneqq \mathcal{H}(\text{“t commit”}\|\mathtt{str})$.

In a second new version, the dealer distributes purely random shares of each private input to the parties. The dealer achieves this by sending each party a random value $r_i$, which party $i$ can use to derive the shares of

$$\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}.$$

To still have valid sharings, the dealer also publishes the auxiliary values

$$\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B.$$

By using the random oracle to obtain the shares of $t_0$, we can make sure that the value of $t_0$ was not maliciously chosen by a prover to be a root of the polynomial

$$(A + Bt) - F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t),$$

which would make the protocol accept even if $(\vec{x}, \vec{y}) \notin \mathrm{NSBC}[\vec{u}, \vec{v}]$.

### 4.4.2 Identification Protocol Using MPCitH

We can transform the multiparty protocol from Section 4.4.1 into an interactive Zero-Knowledge proof of knowledge of a solution $(\vec{x}, \vec{y}) \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. In a basic version of this protocol, the prover creates sharings of all the inputs of the MPC protocol. In particular, the sharings

$$\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}$$

as well as the auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$. The prover executes the MPC protocol and commits to the views of each party. Then, the prover sends the commitment hash

$$h = \mathcal{H}_V(\mathrm{View}_0, \ldots, \mathrm{View}_{N-1})$$

together with the values

$$X_1 + t_0\,(\vec{u} \cdot \vec{x})\,, X_2 + t_0\,(\vec{v} \cdot \vec{x})\,, Y_1 + t_0\,(\vec{v} \cdot \vec{y})\,, Y_2 + t_0\,(\vec{u} \cdot \vec{y})\,, A + Bt_0, t_0$$

and the auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$ to the verifier. The subscript $V$ for the random oracle indicates domain separation, i.e. $\mathcal{H}_V(\mathtt{str}) \coloneqq \mathcal{H}(\text{“view commit”}\|\mathtt{str})$ for any string $\mathtt{str}$. The verifier randomly chooses an index $i^* \in [N]$ and sends it to the prover. The prover then sends all the shares of the values $\vec{R_x}^{[\![i]\!]}$, $\vec{R_y}^{[\![i]\!]}$, ${X_1}^{[\![i]\!]}$, ${X_2}^{[\![i]\!]}$, ${Y_1}^{[\![i]\!]}$, ${Y_2}^{[\![i]\!]}$, $R_A^{[\![i]\!]}$, $R_B^{[\![i]\!]}$ for $i \neq i^*$ to the verifier. Using these values, the verifier can recompute $\mathrm{View}_i$ of each party for $i \neq i^*$. In particular, the values of $t_0^{[\![i]\!]}$, $(X_1 + t_0\,(\vec{u} \cdot \vec{x}))^{[\![i]\!]}$, $(X_2 + t_0\,(\vec{v} \cdot \vec{x}))^{[\![i]\!]}$, $(Y_1 + t_0\,(\vec{v} \cdot \vec{y}))^{[\![i]\!]}$, $(Y_2 + t_0\,(\vec{u} \cdot \vec{y}))^{[\![i]\!]}$, $(A + Bt_0)^{[\![i]\!]}$ for $i \neq i^*$. Note that for this basic protocol, the prover uses the random oracle to derive $t_0^{[\![i]\!]}$ as described in (4.1). For the missing shares corresponding to party $i^*$, the verifier can recompute them by using the received values from the prover. In particular, the verifier computes

$$\begin{aligned}
(X_1 + t_0\,(\vec{u} \cdot \vec{x}))^{[\![i^*]\!]} &= (X_1 + t_0\,(\vec{u} \cdot \vec{x})) - t_0 u_{n-2} - t_0\left(\vec{u} \cdot \vec{\delta}_x\right) - \sum_{i \neq i^*} (X_1 + t_0\,(\vec{u} \cdot \vec{x}))^{[\![i]\!]}\,,\\
(X_2 + t_0\,(\vec{v} \cdot \vec{x}))^{[\![i^*]\!]} &= (X_2 + t_0\,(\vec{v} \cdot \vec{x})) - t_0 v_{n-2} - t_0\left(\vec{v} \cdot \vec{\delta}_x\right) - \sum_{i \neq i^*} (X_2 + t_0\,(\vec{v} \cdot \vec{x}))^{[\![i]\!]}\,,\\
(Y_1 + t_0\,(\vec{v} \cdot \vec{y}))^{[\![i^*]\!]} &= (Y_1 + t_0\,(\vec{v} \cdot \vec{y})) - t_0 v_{n-1} - t_0\left(\vec{v} \cdot \vec{\delta}_y\right) - \sum_{i \neq i^*} (Y_1 + t_0\,(\vec{v} \cdot \vec{y}))^{[\![i]\!]}\,,\\
(Y_2 + t_0\,(\vec{u} \cdot \vec{y}))^{[\![i^*]\!]} &= (Y_2 + t_0\,(\vec{u} \cdot \vec{y})) - t_0 u_{n-1} - t_0\left(\vec{u} \cdot \vec{\delta}_y\right) - \sum_{i \neq i^*} (Y_2 + t_0\,(\vec{u} \cdot \vec{y}))^{[\![i]\!]}\,,\\
(A + Bt_0)^{[\![i^*]\!]} &= (A + Bt_0) - \delta_A - \delta_B t_0 - \sum_{i \neq i^*} (A + Bt_0)^{[\![i]\!]}\,,
\end{aligned}$$

$$t_0^{[\![i^*]\!]} \quad = \quad t_0 - \sum_{i \neq i^*} t_0^{[\![i]\!]}.$$

With these values, the verifier can compute $h' = \mathcal{H}_V(\text{View}_0, \ldots, \text{View}_{N-1})$, check if it matches the commitment hash $h$ and accept or reject accordingly.

**Reducing the communication.** To reduce the required communication of the basic protocol described above, we can use punctured PRFs. The prover chooses a random key $K$ and sets $r_i = \text{PRF}_K(i)$ using a tree PRF based on the GGM tree construction. Using $r_i$, the prover pseudorandomly generates all shares, namely:

$$\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}$$

and computes the auxiliary values

$$\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B.$$

Again, the prover uses the random oracle to derive $t_0^{[\![i]\!]}$ as described in (4.1). After receiving the value $i^* \in [N]$ from the verifier, the prover returns the punctured PRF key $K_{i^*}$ to the verifier. With this, the verifier can compute every $r_i$ except $r_{i^*}$ and derive every input share of the MPC protocol except the shares of party $i^*$. By using the auxiliary values, the verifier can still compute the shares of the missing party and verify the commitment hash as before. We also note that the prover does not have to send the value of $A + Bt_0$, as it can be computed by the verifier from the other received values by using the formula

$$A + Bt_0 = (X_1 + t_0 (\vec{u} \cdot \vec{x}))(Y_1 + t_0 (\vec{v} \cdot \vec{y})) - (X_2 + t_0 (\vec{v} \cdot \vec{x}))(Y_2 + t_0 (\vec{u} \cdot \vec{y})).$$

A malicious prover $\mathcal{P}'$ who does not have a valid sharing of the secret has to cheat by creating an invalid sharing in at least one position. This only remains undetected by the verifier if this position equals $i^*$. We have seen in Section 4.4.1 that the protocol has a false positive probability of $\frac{2}{q^k}$. Therefore, the soundness error of the Zero-Knowledge protocol is $\frac{1}{N} + \frac{2}{q^k}$. We prove this formally in Theorem 4.1. The pseudocode of the identification protocol from this section is given in Figure 4.1.

**Hypercube technique.** A common idea in MPCitH schemes is to use the hypercube technique introduced in [AGH+23]. Applied to the SBC identification protocol, it can reduce the number of scalar products that need to be computed during a protocol execution. To apply this technique, assume that $N$ is a power of two, i.e. $N = 2^D$. Denote the $j$-th bit of the binary decomposition of an integer $i$ by $B_j(i)$. The idea is to transform one instance of the protocol with $N = 2^D$ parties into $D$ instances of the protocol with 2 parties. Consider the sharing of an element $s$ of the form

$$s = \delta_s + \sum_{i=0}^{N-1} R_s^{[\![i]\!]}.$$

For any fixed value $j \in \{0, \ldots, D-1\}$, we see that

$$s = \delta_s + \sum_{B_j(i)=0} R_s^{[\![i]\!]} + \sum_{B_j(i)=1} R_s^{[\![i]\!]}.$$

**Inputs**:

- Public key $(\vec{u}, \vec{v})$ (Verifier)
- Secret key $(\vec{x}, \vec{y}) \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ (Prover)

**Step 1: Commitment**

1. Sample $K \leftarrow_{\$} \{0,1\}^{\lambda}$
2. For $i \in [N]$:
   - $r_i \leftarrow K_i$
   - $X_1^{[\![i]\!]}, X_2^{[\![i]\!]}, Y_1^{[\![i]\!]}, Y_2^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \xleftarrow{r_i}_{\$} \mathbb{F}_{q^k}$
   - $\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]} \xleftarrow{r_i}_{\$} (\mathbb{F}_q)^n$
3. $X_1 \leftarrow \sum_{i=0}^{N-1} X_1^{[\![i]\!]}, X_2 \leftarrow \sum_{i=0}^{N-1} X_2^{[\![i]\!]}, Y_1 \leftarrow \sum_{i=0}^{N-1} Y_1^{[\![i]\!]}, Y_2 \leftarrow \sum_{i=0}^{N-1} Y_2^{[\![i]\!]}$
4. $A \leftarrow X_1 Y_1 - X_2 Y_2$
   $B \leftarrow X_1 (\vec{v} \cdot \vec{y}) + Y_1 (\vec{u} \cdot \vec{x}) - X_2 (\vec{u} \cdot \vec{y}) - Y_2 (\vec{v} \cdot \vec{x})$
5. $\delta_A \leftarrow A - \sum_{i=0}^{N-1} R_A^{[\![i]\!]}$
   $\delta_B \leftarrow B - \sum_{i=0}^{N-1} R_B^{[\![i]\!]}$
   $\vec{\delta_x} \leftarrow \vec{x} - \sum_{i=0}^{N-1} \vec{R_x}^{[\![i]\!]}$
   $\vec{\delta_y} \leftarrow \vec{y} - \sum_{i=0}^{N-1} \vec{R_y}^{[\![i]\!]}$
6. For $i \in [N]$: $t_0^{[\![i]\!]} \leftarrow \mathcal{H}_t(i \,\|\, \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, X_1^{[\![i]\!]}, X_2^{[\![i]\!]}, Y_1^{[\![i]\!]}, Y_2^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \,\|\, \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B)$
7. $t_0 \leftarrow \sum_{i=0}^{N-1} t_0^{[\![i]\!]}$
8. For $i \in [N]$: $\mathrm{View}_i \leftarrow (t_0^{[\![i]\!]}, (X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]}, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]}, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]}, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]}, (A + Bt_0)^{[\![i]\!]})$
9. $h \leftarrow \mathcal{H}_V(\mathrm{View}_0, \ldots, \mathrm{View}_{N-1})$
10. $\mathrm{coeff} \leftarrow (X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0)$
11. Send $(h, \mathrm{coeff}, \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B)$ to the verifier

**Step 2: Challenge**

1. Sample $i^* \leftarrow_{\$} [N]$ and send $i^*$ to the prover

**Step 3: Response**

1. Compute punctured PRF key $K_{i^*}$ and send $K_{i^*}$ to the verifier

**Step 4: Verification**

1. For $i \in [N] \setminus \{i^*\}$:
   - Compute $r_i$ from $K_{i^*}$
   - Compute $X_1^{[\![i]\!]}, X_2^{[\![i]\!]}, Y_1^{[\![i]\!]}, Y_2^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}, \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, t_0^{[\![i]\!]}$ from $r_i$ and $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$
2. Compute $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i^*]\!]}, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i^*]\!]}, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i^*]\!]}, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i^*]\!]}, (A + Bt_0)^{[\![i^*]\!]}, t_0^{[\![i^*]\!]}$ from coeff and $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$
3. For $i \in [N]$: Compute $\mathrm{View}_i$
4. Output $\mathtt{Boolean}(\mathcal{H}_V(\mathrm{View}_0, \ldots, \mathrm{View}_{N-1})$ is equal to $h)$

**Figure 4.1:** Identification protocol for the SBC problem

After receiving the PPRF key, the verifier only knows one of the two sums for each $j \in [D]$. This reduces the amount of scalar products that are needed in the protocol execution: In the basic version, four scalar products are computed by each of the $N$ parties. In particular, these scalar products are needed to compute the values

$$(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]}, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]}, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]}, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]}.$$

In the commitment step of the hypercube version, these $4N$ scalar product computations are replaced by $8D$ scalar product computations. For the verification, the verifier only has to compute $4D$ scalar products instead of $4(N-1)$. By using this technique, the amount of multiplications between elements of $\mathbb{F}_{q^k}$ and $\mathbb{F}_q$ during the protocol execution decreases. The computations can therefore be performed faster in this version. For a comparison of the running times of the different implementations, see Section 4.7.

## 4.5 Signature Scheme

We can turn the Honest Verifier Zero Knowledge (HVZK) protocol from Section 4.4 into a signature scheme by using the Fiat-Shamir transform [FS87] with $\tau$ repetitions to obtain a security level of $\lambda = \tau \log(N)$. The public key consists of the two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$ and the private key is $(\vec{x}, \vec{y}) \in \text{NSBC}[\vec{u}, \vec{v}]$. The prover executes $\tau$ rounds of the MPC protocol and commits to all of them by sending one commitment hash. After receiving the commitment hash to the $\tau$ round protocol, the auxiliary values and the broadcast shares for each round, the verifier picks a random $i^* \in [N]$ for each round and sends each of these $\tau$ values to the prover. The prover sends the punctured PRF keys for each round to the verifier, which allows the verifier to validate the commitment hash by recomputing the missing shares for each round.

### 4.5.1 Key and Signature Size

**Signature size.** To avoid collisions in the hash function, we need outputs of $2\lambda$ bits. Since we use the GGM construction for the puncturable PRF, we can instead use a random global salt of $2\lambda$ bits and outputs of size $\lambda$ for the internal hash function used in the GGM tree. By doing this, we reduce the PPRF key size to $\lambda \log(N)$ bits. To avoid generic attacks, we also need the bitsize of the elements of $\mathbb{F}_{q^k}$ to be $2\lambda$. Our assumption that $n \approx \frac{k}{2}$ implies that the bitsize of the elements of $(\mathbb{F}_q)^n$ is $\lambda$. The signature in this basic version consists of the following elements:

- One hash value $h$ corresponding to a global commitment to the $\tau$ round protocol of size $2\lambda$ and one salt of size $2\lambda$.
- One punctured PRF key for each round.
- The auxiliary values $\delta_A, \delta_B \in \mathbb{F}_{q^k}$ and $\vec{\delta}_x, \vec{\delta}_y \in (\mathbb{F}_q)^n$ for each round.
- The values $X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0 \in \mathbb{F}_{q^k}$ for each round.

The total communication cost in bits of the protocol is therefore

$$\tau \cdot (\lambda \log(N)) + 6\tau\lambda + 10\tau\lambda + 4\lambda = \lambda^2 + 16\tau\lambda + 4\lambda \text{ bits},$$

where we use the fact that $\lambda = \tau \log(N)$ in the equation.

**Key size.** The public key can be derived from a seed of size $\lambda$ together with one element in $\mathbb{F}_{q^k}$. By using the seed, the elements $u_0, \ldots, u_{n-1}$ and $v_0, \ldots, v_{n-2}$ can be derived using a PRG and the missing element $v_{n-1}$ is sent separately. The key size in bits is therefore $3\lambda$. For $\lambda = 128$, this yields a public key size of 48 bytes.

**Suggested parameter choice.** For the security parameter of $\lambda = 128$ bits, we need $q^k \geq 2^{128}$ by the discussion above, which can be achieved for example by choosing $q = 2, k = 257$ and $n = 130$. Note that we do not suggest $k = 256$ since it might allow for subfield attacks in the field $\mathbb{F}_{2^{256}}$. Indeed, there are cryptographic schemes that suffer from subfield attacks [JV11, BBB+20, BFJT09]. From a cryptanalytical point, it would be very interesting to study whether it is indeed possible to perform subfield attacks against SBC. For the signature, we want to pick the minimum number of rounds $\tau$ such that the cost of the forgery attack on the Fiat-Shamir-based signature is at least $2^{128}$. This cost is given by the following formula (which is based on the attack from Kales and Zaverucha [KZ20]):

$$\text{cost} = \min_{\tau_1, \tau_2 : \tau_1 + \tau_2 = \tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} \mathrm{p}^i (1-\mathrm{p})^{\tau - i}} + N^{\tau_2} \right\},$$

where p is the false positive probability of the underlying identification scheme. For our scheme with $\mathrm{p} = \frac{2}{2^{257}}$, the parameters for $\lambda = 128$ are listed in Table 4.1.

| $D$ | $\tau$ | \|sgn\| |
|---|---|---|
| 8 | 16 | 6 222 B |
| 9 | 15 | 6 078 B |
| 10 | 13 | 5 484 B |
| 11 | 12 | 5 259 B |
| 12 | 11 | 5 002 B |
| 13 | 10 | 4 713 B |
| 15 | 9 | 4 536 B |
| 16 | 8 | 4 167 B |

**Table 4.1:** Signature size for $N = 2^D$ parties and $\tau$ rounds with parameters $q = 2, k = 257$ and $n = 130$ for the basic version of the SBC problem.

As the false positive probability p is very small for our scheme, we can also use the union bound to get an upper bound on the probability that a malicious prover can generate a valid signature, which is given by

$$\frac{1}{N^\tau} + \tau \cdot \mathrm{p}.$$

This yields a security parameter of more than $\lambda = 128$ bits for the values $D = 9$, 10, 11, 12, 13, 15 from Table 4.1. For the edge cases of $D = 8$ and $D = 16$, we obtain a value of $\lambda \geq 127.9999$, which is sufficiently close to the target security level for this parameter choice. In Section 4.7, we introduce some improvements to further decrease the signature sizes from Table 4.1.

## 4.6 Formal Schemes

In this section, we give a formal description of the identification scheme proposed in Section 4.4. We assume that the internal random oracle $\mathcal{H}$ used in the GGM tree construction has an output size of $\lambda$ bits and that the size of the finite field $\mathbb{F}_{q^k}$ is $2^{2\lambda}$ to avoid generic attacks faster than $2^{\lambda}$. We further assume that the random oracle $\mathcal{H}_V$ used to compute the commitment hash has an output size of $2\lambda$ bits.
We first give the description of the puncturable PRF which we use in our construction.The number of leaves of the puncturable PRF is a power of two, i.e. $N = 2^D$. To obtain a finite field element, we expand the leaves of the tree to size $3\lambda$ before applying the map $\gamma_{\text{Field}}$ to convert the leaves to field elements. Here, we assume that there exists an efficient bijective map $\gamma_{\text{Field}}$ from $\mathbb{Z}_{q^k}$ to $\mathbb{F}_{q^k}$. Similarly, we assume there is an efficient bijective map $\gamma_{\text{Vec}}$ from $\mathbb{Z}_{q^n}$ to $(\mathbb{F}_q)^n$. To obtain vectors in $(\mathbb{F}_q)^n$, where $n \approx k/2$, we apply the map $\gamma_{\text{Vec}}$ to a leaf. Before applying these maps, we first expand the bitstring of the respective leaf and then map the extended bitstring to an integer using the function `ToInteger`. By doing this, we can obtain pseudorandom field elements and vectors. We use domain separation to be able to use the same random oracle $\mathcal{H}$ for each hash function.
Algorithm 1 is the implementation of the random oracle and Algorithms 2 to 10 are the implementations of the puncturable PRF. In Algorithms 11 to 14, we provide implementations of the four step identification protocol described in Section 4.4.2, i.e. Commitment, Challenge, Response and Verification.

**Algorithm 1** Random Oracle $\mathcal{H}$

**Input**
A bitstring **Message**

**Output**
A bitstring of size $\lambda$

```
1: Let Dict be a global dictionary initially empty
2: if Message appears in Dict then
3:     return Dict[Message]
4: else
5:     Let Value be a uniformly random λ-bitstring
6:     Define Dict[Message] = Value
7:     return Value
```

**Algorithm 2** Descendant

**Input**
An integer **Level** corresponding to the current level of the node in the tree
A bitstring **Salt** of size $2\lambda$
A bitstring **Node** of size $\lambda$ corresponding to the current node in the tree
A bitstring **SubPath** of the path from the root to a sibling of the node

**Output**
A bitstring of size $\lambda$ corresponding to the sibling node along the subpath

```
1: return H("Descendant"‖Level‖Salt‖Node‖SubPath)
```

**Algorithm 3** RecursiveDescend

**Input**

An integer **Level** corresponding to the current level of the node in the tree
A bitstring **Salt** of size $2\lambda$
A bitstring **Node** of size $\lambda$ corresponding to the current node in the tree
A bitstring **Path** corresponding to the descend path from the root to the leaf

**Output**

A bitstring of size $\lambda$ corresponding to the leaf node along the subpath

1: **if** Level is equal to the length of Path **then**
2: **return** Node
3: **else**
4: Let **SubPath** = Path$[1 \dots \text{Level} + 1]$
5: Let NextNode = **Descendant**(Level, Salt, Node, SubPath)
6: **return** **RecursiveDescend**(Level + 1, Salt, NextNode, Path)

**Algorithm 4** ExpandNode

**Input**

A bitstring **Sep** for domain separation; A bitstring **Salt** of size $2\lambda$
A bitstring **Leaf** of size $\lambda$ with corresponding path **PathLeaf**

**Output**

A bistring of size $3\lambda$

1: Let **ExpandedNode** = $\mathcal{H}$("Exp0"$\|$Sep$\|$Salt$\|$Leaf$\|$PathLeaf)
$\|\mathcal{H}$("Exp1"$\|$Sep$\|$Salt$\|$Leaf$\|$PathLeaf) $\|\mathcal{H}$("Exp2"$\|$Sep$\|$Salt$\|$Leaf$\|$PathLeaf)
2: **return** ExpandedNode

**Algorithm 5** FieldEltFromLeaf

**Input**

A bitstring **Sep** for domain separation; A bitstring **Salt** of size $2\lambda$
A bitstring **Leaf** of size $\lambda$ with corresponding path **PathLeaf**

**Output**

An element of the finite field $\mathbb{F}$

1: **return** $\gamma_{\text{Field}}$ ( `ToInteger`(**ExpandNode**(Sep, Salt, Leaf, PathLeaf)) mod $|\mathbb{F}|$)

**Algorithm 6** VecFromLeaf

**Input**

A bitstring **Sep** for domain separation; A bitstring **Salt** of size $2\lambda$
A bitstring **Leaf** of size $\lambda$ with corresponding path **PathLeaf**

**Output**

A vector in $\mathbb{F}^n$

1: **return** $\gamma_{\text{Vec}}$ (`ToInteger`(**ExpandNode**(Sep, Salt, Leaf, PathLeaf)) mod $|\mathbb{F}^n|$)

**Algorithm 7** PuncturedKey

**Input**
A bitstring **Salt** of size $2\lambda$
A bitstring **Leaf** of size $\lambda$ with corresponding path **PathLeaf**

**Output**
A bitstring corresponding to the punctured key along the path

Init **PuncturedKey** (as an empty array)
Let PuncturedKey[0] = PathLeaf
**for** Level **from** 1 **to** TreeDepth **do**
    Set **Path** = PathLeaf[1 . . . Level]; **Flip Bit** Path[Level]
    Set PuncturedKey[Level] = **RecursiveDescend**(0, Salt, RootKey, Path)
**return** PuncturedKey

**Algorithm 8** LeafFromPuncKey

**Input**
A bitstring **Salt** of size $2\lambda$; A **PuncturedKey**
A bitstring **PathLeaf** corresponding to the path from the root to a leaf

**Output**
A bitstring of size $\lambda$ corresponding to the leaf node along the path

Set **ForbiddenPath** = PuncturedKey[0]
**if** PathLeaf is equal to ForbiddenPath **then**
    **return** $\perp$
Let **Level** be the first bit position where PathLeaf differs from ForbiddenPath
Let **Leaf** = **RecursiveDescend**(Level, Salt, PuncturedKey[Level], PathLeaf)
**return** Leaf

**Algorithm 9** FieldEltFromPuncKey

**Input**
A bitstring **Sep** for domain separation; A bitstring **Salt** of size $2\lambda$
A **PuncturedKey**
A bitstring **PathLeaf** corresponding to the path from the root to a leaf

**Output**
An element of the finite field $\mathbb{F}$ corresponding to the leaf of PathLeaf

Set **ForbiddenPath** = PuncturedKey[0]
**if** PathLeaf is equal to ForbiddenPath **then**
    **return** $\perp$
Let **Level** be the first bit position where PathLeaf differs from ForbiddenPath
Let **Leaf** = **RecursiveDescend**(Level, Salt, PuncturedKey[Level], PathLeaf)
**return FieldEltFromLeaf**(Sep, Salt, Leaf, PathLeaf)

**Algorithm 10** VecFromPuncKey

**Input**
A bitstring **Sep** for domain separation; A bitstring **Salt** of size $2\lambda$
A **PuncturedKey**
A bitstring **PathLeaf** corresponding to the path from the root to a leaf

**Output**
A vector in $\mathbb{F}^n$ corresponding to the leaf of PathLeaf

1: Set **ForbiddenPath** = PuncturedKey[0]
2: **if** PathLeaf is equal to ForbiddenPath **then**
3: **return** $\perp$
4: Let **Level** be the first bit position where PathLeaf differs from ForbiddenPath
5: Let **Leaf** = **RecursiveDescend**(Level, Salt, PuncturedKey[Level], PathLeaf)
6: **return VecFromLeaf**(Sep, Salt, Leaf, PathLeaf)

**Theorem 4.1.** Consider an instance NSBC$[\vec{u}, \vec{v}]$ of the normalized SBC problem for $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$. The identification protocol described above is a statistical honest verifier Zero-Knowledge proof of knowledge of $(\vec{x}, \vec{y}) \in$ NSBC$[\vec{u}, \vec{v}]$ with soundness error $1/N + 2/q^k$.

*Proof.*
**Completeness.** By the discussion of the polynomial $F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t)$ in Section 4.4, we see that an execution of the protocol by an honest prover and an honest verifier is always accepted.
**Statistical honest Verifier Zero-Knowledge.** We construct a simulator $\mathcal{S}$ in the following way:

- Uniformly at random pick a RootKey, a Salt, a Query and auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$.
- Compute a PuncturedKey from RootKey, Salt and Query.
- Use the Verify algorithm to get the expected Commitment of the input (without using the Commitment hash in the algorithm).
- Output a simulated transcript of the protocol using PuncturedKey, Salt, Query, Commitment and the auxiliary values.

We claim that the distribution of this output is statistically indistinguishable from a valid transcript of the protocol. To see that, first note that if a valid prover runs the protocol with the same values for RootKey, Salt and Query, all the values for $X_1^i$, $X_2^i$, $Y_1^i$, $Y_2^i$, $\vec{R}_x^i$, $\vec{R}_Y^i$, $\vec{R}_A^i$, $\vec{R}_B^i$, $t_0^i$, $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^i$, $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^i$, $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i$, $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i, (A + Bt_0)^i$ for $i \neq$ Query are the same as in the run of the simulator $\mathcal{S}$. Only the values for the index Query are missing. There exists a configuration for these values for which the simulated and honest transcripts would be equal. By programming the random oracle at the respective points, the transcript produced by $\mathcal{S}$ could also come from a legitimate prover. These two transcripts only differ because the auxiliary values were chosen uniformly at random in the simulator but are obtained using a summation of outputs of the functions FieldEltFromLeaf and VecFromLeaf on random values for the prover. The outputs of the functions FieldEltFromLeaf and VecFromLeaf are statistically close to a random map. Therefore, the distribution obtained by the simulated transcript is statistically close to the honest one.

**Algorithm 11** Step 1: Commitment

Let **RootKey** be a uniform random bitstring (of the expected size $\lambda$)
Let **Salt** be a uniform random bitstring (of the expected size $2\lambda$)
Let **CommitString** be the empty string
Let $X_1, X_2, Y_1, Y_2, t_0 = 0$
**for** i **from** 0 **to** $N-1$ **do**
    Let **Path** be the $m$-bit binary encoding of $i-1$
    Let $r_i = \textbf{RecursiveDescend}(0, \text{Salt}, \text{RootKey}, \text{Path})$
    Let ${X_1}^i = \textbf{FieldEltFromLeaf}(\text{“X1 derivation”}, \text{Salt}, r_i, \text{Path})$
    Let ${X_2}^i = \textbf{FieldEltFromLeaf}(\text{“X2 derivation”}, \text{Salt}, r_i, \text{Path})$
    Let ${Y_1}^i = \textbf{FieldEltFromLeaf}(\text{“Y1 derivation”}, \text{Salt}, r_i, \text{Path})$
    Let ${Y_2}^i = \textbf{FieldEltFromLeaf}(\text{“Y2 derivation”}, \text{Salt}, r_i, \text{Path})$
    Set $X_1 = X_1 + {X_1}^i$
    Set $X_2 = X_2 + {X_2}^i$
    Set $Y_1 = Y_1 + {Y_1}^i$
    Set $Y_2 = Y_2 + {Y_2}^i$
    Let $R_A^i = \textbf{FieldEltFromLeaf}(\text{“A derivation”}, \text{Salt}, r_i, \text{Path})$
    Let $R_B^i = \textbf{FieldEltFromLeaf}(\text{“B derivation”}, \text{Salt}, r_i, \text{Path})$
    Let $\vec{R_x}^i = \textbf{VecFromLeaf}(\text{“x derivation”}, \text{Salt}, r_i, \text{Path})$
    Let $\vec{R_y}^i = \textbf{VecFromLeaf}(\text{“y derivation”}, \text{Salt}, r_i, \text{Path})$
Let $A = X_1 Y_1 - X_2 Y_2$
Let $B = X_1 (\vec{v} \cdot \vec{y}) + Y_1 (\vec{u} \cdot \vec{x}) - X_2 (\vec{u} \cdot \vec{y}) - Y_2 (\vec{v} \cdot \vec{x})$
Let $\delta_A = A - \sum_{i=0}^{N-1} R_A^i$
Let $\delta_B = B - \sum_{i=0}^{N-1} R_B^i$
Let $\vec{\delta}_x = \vec{x} - \sum_{i=0}^{N-1} \vec{R_x}^i$
Let $\vec{\delta}_y = \vec{y} - \sum_{i=0}^{N-1} \vec{R_y}^i$
**for** i **from** 0 **to** $N-1$ **do**
    $h_{t_0}^i = \mathcal{H}_t\left(i \parallel \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B\right)$
    Let $t_0^i = \gamma_{\text{Field}}(\texttt{ToInteger}(h_{t_0}^i) \mod |\mathbb{F}|)$
    Set $t_0 = t_0 + t_0^i$
**for** i **from** 0 **to** $N-1$ **do**
    Let $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^i = X_1^i + t_0(\vec{u} \cdot \vec{R_x}^i)$
    Let $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^i = X_2^i + t_0(\vec{v} \cdot \vec{R_x}^i)$
    Let $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i = Y_1^i + t_0(\vec{v} \cdot \vec{R_y}^i)$
    Let $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i = Y_2^i + t_0(\vec{u} \cdot \vec{R_y}^i)$
    Let $(A + Bt_0)^i = R_A^i + R_B^i t_0$
    Let $\text{View}^i = ({t_0}^i, (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i, (A + Bt_0)^i)$
    Append encoding of $\text{View}^i$ to CommitString
Let $X_1 + t_0 (\vec{u} \cdot \vec{x}) = \sum_{i=0}^{N-1} (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i + t_0 u_{n-2} + t_0(\vec{u} \cdot \vec{\delta}_x)$
Let $X_2 + t_0 (\vec{v} \cdot \vec{x}) = \sum_{i=0}^{N-1} (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i + t_0 v_{n-2} + t_0(\vec{v} \cdot \vec{\delta}_x)$
Let $Y_1 + t_0 (\vec{v} \cdot \vec{y}) = \sum_{i=0}^{N-1} (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i + t_0 v_{n-1} + t_0(\vec{v} \cdot \vec{\delta}_y)$
Let $Y_2 + t_0 (\vec{u} \cdot \vec{y}) = \sum_{i=0}^{N-1} (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i + t_0 u_{n-1} + t_0(\vec{u} \cdot \vec{\delta}_y)$
Let $\textbf{Commitment} = (\mathcal{H}_V(\text{CommitString}), (X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0), \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B)$
**return** Commitment

**Algorithm 12** Step 2: Challenge

1: Let **Query** be a uniformly random integer in $[N]$
2: **return** Query

**Algorithm 13** Step 3: Response

**Input**
An integer **Query** in $[N]$

**Output**
A punctured key
A salt

1: Import **RootKey** and **Salt** from Step 1
2: Convert Query to a binary **QueryPath**
3: **return** Salt$\|$**PuncturedKey**(Salt, RootKey, QueryPath)

**Proof of Knowledge.** Let $\mathcal{P}^*$ be a prover that convinces a verifier with probability $\geq 1/N + 2/q^k + \varepsilon$ for a non-negligible $\varepsilon$. Then we can build an extractor which, with read access to the random oracle memory, can learn the secret $(\vec{x}, \vec{y})$.
First, note that all random oracle queries made in many executions of the verify function are pairwise distinct with extremely high probability. Since we use a salt and domain separators, identical queries can only occur if identical salt values are selected. After $\mathcal{P}^*$ runs the commitment step of the protocol, the extractor can obtain the input to the random oracle that produced the commitment hash $h$. If this commitment hash was not produced by an honest protocol execution, the verification algorithm can only succeed with exponentially small probability. With the input string, the extractor can construct a candidate tree for the puncturable PRF used by the prover to derive the values for $r_i$. For each of these values, the extractor knows the corresponding value of $t_0^i$ from the input string. It searches through the oracle queries whose input could have come from a call of the Descendant function at the corresponding position by considering all queries to the random oracle that use the domain separator "t commit". It then uses the map $\gamma_{\text{Field}}$ on this value modulo $q^k$ to find the matching $h_{t_0}^i$. By searching though the oracle queries again, it can find the value of $r_i$ that was used to obtain $h_{t_0}^i$.
There are three cases that can occur after using this strategy to reconstruct the tree. The extractor obtains the full GGM tree, one leaf is missing or more than one leaf is missing. We consider each of these cases separately:

- If more than one leaf is missing, the extractor can use the response from $\mathcal{P}^*$ from the third step to obtain a PuncturedKey. Using this key, the extractor can find at least one missing leaf in the tree. If this PuncturedKey did not come from an honest protocol execution, the probability that the partial tree obtained from this key matches the tree of the extractor is negligible.
- If no leaf is missing, the extractor can compute every share of every party in the MPC protocol. Using the auxiliary values, the extractor can compute the values of $\vec{x}$ and $\vec{y}$ used by $\mathcal{P}^*$. If $(\vec{x}, \vec{y}) \notin \text{NSBC}[\vec{u}, \vec{v}]$, the verification algorithm in step 4 outputs accept with probability $2/q^k$, which occurs if $t_0$ happens to be a root of the polynomial $F^{\vec{x},\vec{y},\vec{u},\vec{v}}_{X_1,X_2,Y_1,Y_2}(t) - (A + Bt)$.

**Algorithm 14** Step 4: Verification

**Input**

A bitstring **Salt** of size $2\lambda$ and a **PuncKey** from Step 3

A **Commitment** from Step 1 of the form

$(h, (X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0), \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B))$

**Output**

A boolean corresponding to accept or reject

Let **Query** = `ToInteger`(PuncKey[0])
Let **CommitVerifyString** be the empty string
**for** i **in** $[N] \setminus \{\text{Query}\}$ **do**
    Let **Path** be the $m$-bit binary encoding of $i - 1$
    Let $r_i =$ **LeafFromPuncKey**(Salt, PuncKey, Path)
    Let ${X_1}^i =$ **FieldEltFromPuncKey**("X1 derivation", Salt, PuncKey, Path)
    Let ${X_2}^i =$ **FieldEltFromPuncKey**("X2 derivation", Salt, PuncKey, Path)
    Let ${Y_1}^i =$ **FieldEltFromPuncKey**("Y1 derivation", Salt, PuncKey, Path)
    Let ${Y_2}^i =$ **FieldEltFromPuncKey**("Y2 derivation", Salt, PuncKey, Path)
    Let $R_A^i =$ **FieldEltFromPuncKey**("A derivation", Salt, PuncKey, Path)
    Let $R_B^i =$ **FieldEltFromPuncKey**("B derivation", Salt, PuncKey, Path)
    Let $\vec{R_x}^i =$ **VecFromPuncKey**("x derivation", Salt, PuncKey, Path)
    Let $\vec{R_y}^i =$ **VecFromPuncKey**("y derivation", Salt, PuncKey, Path)
    $h_{t_0}^i = \mathcal{H}_t \left( i \parallel \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B \right)$
    Let $t_0^i = \gamma_{\text{Field}}(\texttt{ToInteger}(h_{t_0}) \mod |\mathbb{F}|)$
    Let $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^i = X_1^i + t_0(\vec{u} \cdot \vec{R_x}^i)$
    Let $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^i = X_2^i + t_0(\vec{v} \cdot \vec{R_x}^i)$
    Let $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i = Y_1^i + t_0(\vec{v} \cdot \vec{R_y}^i)$
    Let $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i = Y_2^i + t_0(\vec{u} \cdot \vec{R_y}^i)$
    Let $(A + Bt_0)^i = R_A^i + R_B^i t_0$
Let $A + Bt_0 = (X_1 + t_0 (\vec{u} \cdot \vec{x}))(Y_1 + t_0 (\vec{v} \cdot \vec{y})) - (X_2 + t_0 (\vec{v} \cdot \vec{x}))(Y_2 + t_0 (\vec{u} \cdot \vec{y}))$
Let $t_0^{\text{Query}} = t_0 - \sum_{i \neq \text{Query}} t_0^i$
Let $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{\text{Query}} = (X_1 + t_0 (\vec{u} \cdot \vec{x})) - t_0 u_{n-2} - t_0(\vec{u} \cdot \vec{\delta}_x) - \sum_{i \neq \text{Query}} (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i$
Let $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^{\text{Query}} = (X_2 + t_0 (\vec{v} \cdot \vec{x})) - t_0 v_{n-2} - t_0(\vec{v} \cdot \vec{\delta}_x) - \sum_{i \neq \text{Query}} (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i$
Let $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{\text{Query}} = (Y_1 + t_0 (\vec{v} \cdot \vec{y})) - t_0 v_{n-1} - t_0(\vec{v} \cdot \vec{\delta}_y) - \sum_{i \neq \text{Query}} (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i$
Let $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{\text{Query}} = (Y_2 + t_0 (\vec{u} \cdot \vec{y})) - t_0 u_{n-1} - t_0(\vec{u} \cdot \vec{\delta}_y) - \sum_{i \neq \text{Query}} (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i$
Let $(A + Bt_0)^{\text{Query}} = (A + Bt_0) - \delta_A - \delta_B t_0 - \sum_{i \neq \text{Query}} (A + Bt_0)^i$
**for** i **from** 0 **to** $N - 1$ **do**
    Let $\text{View}^i = (t_0{}^i, (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i, (A + Bt_0)^i)$
    Append encoding of $\text{View}^i$ to CommitVerifyString
**return** `Boolean`($\mathcal{H}_V$(CommitVerifyString) is equal to $h$)

- If one leaf is missing, then $\mathcal{P}^*$ might still be able to provide a punctured key which is consistent with the tree obtained by the extractor. However, this only accounts for a $1/N$ success probability. In this case, the extractor queries at another position to obtain the missing path from the root to the leaf if it gets a consistent response from $\mathcal{P}^*$ in step 3. This occurs with probability at least $\varepsilon$. If the extractor learns the full tree, it can compute the secret key $(\vec{x}, \vec{y})$.

Therefore, the combination of the extractor and $\mathcal{P}^*$ can break the SBC problem in time $\mathcal{O}(1/\varepsilon)$. □

## 4.7 Implementation and Optimizations

In this section, we provide the signature sizes and running times of our scheme based on an implementation in C. To significantly improve the running time of the scheme, we make use of a puncturable PRF based on the AES block cipher instead of a salted hash function, which is a technique introduced in [BCC+24c]. We compare different versions of the signature based on SBC. In the basic version, we implemented the scheme as described in Figure 4.1. In a second implementation, we use a different method to derive the value of $t_0 \in \mathbb{F}_{q^k}$ used in the protocol: Here, we use the initial commitment of the signer to derive the value of $t_0$ for round $j = 0, \ldots, \tau - 1$ using the random oracle:

$$t_{0,j} \coloneqq \mathcal{H}\left(\text{"t derivation"} \parallel j \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B\right),$$

which reduces the signature size by $\log\left(q^k\right)$ bits for each round. We denote this variant of the SBC problem by SBC$^{\mathcal{H}(t_0)}$. For both variants, we introduce another version of the scheme which further reduces the signature size. We can use the correlated GGM (cGGM) tree introduced in [GYW+23] to remove $\vec{\delta}_x$ from the communication cost, as we use exact sharings for $\vec{x}$ with an XOR-preserving GGM tree. We compare the signature sizes of the resulting variants of the scheme for the 128-bit security level in Table 4.2.

| $D$ | $\tau$ | SBC GGM | SBC cGGM | SBC$^{\mathcal{H}(t_0)}$ GGM | SBC$^{\mathcal{H}(t_0)}$ cGGM |
|---|---|---|---|---|---|
| 8 | 16 | 6 222 B | 5 966 B | 5 708 B | 5 452 B |
| 9 | 15 | 6 078 B | 5 838 B | 5 596 B | 5 356 B |
| 10 | 13 | 5 484 B | 5 276 B | 5 066 B | 4 858 B |
| 11 | 12 | 5 259 B | 5 067 B | 4 873 B | 4 681 B |
| 12 | 11 | 5 002 B | 4 826 B | 4 649 B | 4 473 B |
| 13 | 10 | 4 713 B | 4 553 B | 4 392 B | 4 232 B |
| 15 | 9 | 4 536 B | 4 392 B | 4 247 B | 4 103 B |
| 16 | 8 | 4 167 B | 4 039 B | 3 910 B | 3 782 B |

**Table 4.2:** Signature size for $\lambda = 128$ using $N = 2^D$ parties and $\tau$ rounds with parameters $q = 2$, $k = 257$ and $n = 130$ for the basic SBC GGM, SBC cGGM, SBC$^{\mathcal{H}(t_0)}$ GGM and SBC$^{\mathcal{H}(t_0)}$ cGGM versions.

### 4.7.1 Comparison with other MPCitH Signature Schemes

We compare our scheme with the current state-of-the-art MPCitH-based signatures for the 128-bit security level and list the corresponding public key sizes |pk| and signature sizes |sgn|. The SBC signature scheme yields smaller signature sizes and smaller public key sizes compared to previously known signature schemes based on the MPCitH paradigm.

| Name | Year | \|pk\| | \|sgn\| |
|---|---|---|---|
| Syndrome Decoding in the Head [FJR22] | 2022 | 144 B | 8 481 B |
| MinRank in the Head [ARZV22] | 2022 | 129 B | 6 695 B |
| MQ on my Mind [FR23] | 2023 | 218 B | 6 348 B |
| Biscuit [BKPV23] | 2023 | 68 B | 5 748 B |
| MIRA [ABB+23] | 2023 | 84 B | 5 640 B |
| SBC | 2024 | 48 B | 5 452 B |

**Table 4.3:** Comparison of the SBC scheme with signatures from the literature for 128-bit security and $N = 2^8$ parties.

| Name | Year | \|sgn\| |
|---|---|---|
| Syndrome Decoding in the Head [FJR22] | 2022 | 5 689 B |
| MinRank in the Head [ARZV22] | 2022 | 4 542 B |
| SBC | 2024 | 3 782 B |

**Table 4.4:** Comparison of the hypercube version of the SBC scheme with hypercube signatures from the literature for 128-bit security and $N = 2^{16}$ parties.

### 4.7.2 About the Speed-ups and the PPRF Implementation

While the signature size of our proposal is inherently due to the choice of the SBC problem, the speed of the implementation is mainly the consequence of the implementation of the puncturable PRF. This is especially relevant when using a large number of parties such as $N = 2^{16}$. A large part of the speed-up comes from the use of the AES-based tree suggested in [BCC+24c].

However, this does not completely account for the speed of our implementation. The other time limiting step in the implementation is the hypercube step, where a sharing between a large number of parties is transformed into many sharings between two parties. This is usually done using a quasi-linear folding algorithm which computes $D$ binary sharings from a single sharing between $N = 2^D$ parties in time $O(D \cdot N)$. We now recall the standard algorithm for this.

Let $(T^{[\![i]\!]})_{i\in[0\cdots 2^D-1]}$ be an additive sharing of $\mathcal{T}$. For simplicity, we ignore offsets here and assume that:

$$\mathcal{T} = \bigoplus_{i=0}^{2^D-1} T^{[\![i]\!]}.$$

Following the standard hypercube technique, we obtain one sharing for every bit position

$j$ from 0 to $D-1$ by setting:

$$\mathcal{T} = \mathcal{T}^{[\![0]\!]_j} \oplus \mathcal{T}^{[\![1]\!]_j} \quad \text{where} \quad \mathcal{T}^{[\![b]\!]_j} = \bigoplus_{\substack{i \text{ s.t.} \\ B_j(i)=b}} T^{[\![i]\!]}.$$

For efficiency, a standard optimization is that it is sufficient for the signer to compute $\mathcal{T}^{[\![0]\!]_j}$, since $\mathcal{T}^{[\![1]\!]_j}$ can be obtained with a single additional XOR operation with $\mathcal{T}$. Of course, the verifier can only compute one of the two values and does not need this standard optimization.
Profiling our code, we realized this *folding* step was dominating the computation for $N = 2^{16}$ parties. For this reason, we replaced it by a linear time algorithm, i.e., with time $O(N)$ based on the following technique. We first create a partial folding of $T$ into a table $T_h$ of $2^{D-1}$ values defined as:

$$T_h^{[\![i]\!]} = T^{[\![i]\!]} \oplus T^{[\![i+2^{D-1}]\!]}.$$

For any bit position $j < D-1$, if we apply the standard folding to $T_h$ to obtain shares $\mathcal{T}_h^{[\![b]\!]_j}$, we see that:

$$\mathcal{T}^{[\![b]\!]_j} = \mathcal{T}_h^{[\![b]\!]_j}.$$

Indeed, the two shares are sums of the exact same values (in a different order) since each $T_h^{[\![i]\!]}$ regroups two values from $T$ whose positions only differ by their high-order bit.
This allows us to derive a recursive algorithm that, given $T$, computes all the sharings $\mathcal{T}^{[\![0]\!]_j}$ plus the shared value $\mathcal{T}$ using $2^{D+1}$ XOR operations. The pseudocode for this fast recursive folding is given in Algorithm 15.

**Algorithm 15** FastRecursiveFolding

**Input**
A table $T$ of size $N = 2^D$
**Output**
Concatenation of the 0-shares of the $D$-foldings, plus the global shared value

1: **if** $N$ is 2 **then**
2: **return** $T_0 \| (T_0 \oplus T_1)$
3: **else**
4: Create $T_h$ of size $N/2$
5: **for** $i$ **from** 0 **to** $N/2-1$ **do**
6: Set $T_h^{[\![i]\!]} = T^{[\![i]\!]} \oplus T^{[\![i+2^{D-1}]\!]}$
7: **return** $\left(\bigoplus_{i=0}^{2^{D-1}} T^{[\![i]\!]}\right) \|$ **FastRecursiveFolding**$(T_h, N/2)$

### 4.7.3 Running Times of the Different Versions

We compare the running times of the SBC problem using different instances of the GGM tree in the protocol which we described throughout this chapter. For this comparison, we consider the SBC$^{\mathcal{H}(t_0)}$ variant where we use a hash function to derive the values of $t_0$ for each round, as described in the beginning of Section 4.7. In a first version, we use SHA-256 in the construction of the cGGM tree, denoted by cGGM$_{\text{SHA}}$. In a second version,

we improve the running times by using the AES-based tree from [BCC+24c], which we call $\text{cGGM}_{\text{AES}}$. To further improve the running times, we use the fast folding algorithm from Section 4.7.2. We indicate this by the subscript $\text{SBC}_{\text{FF}}^{\mathcal{H}(t_0)}$. The improvement of the running time of the scheme based on these different versions is illustrated in Table 4.5, Table 4.6, Table 4.7 and Table 4.8. All computations were performed using an AMD EPYC 9374F processor running at 3.85 GHz.
We see that using AES as the tree derivation function yields a speedup of a factor of $25-30\times$ compared to using SHA-256 for large values of $D$. Using the fast folding algorithm again yields a speedup by a factor of two.

| | | | SBC $\text{GGM}_{\text{SHA}}$ | | SBC $\text{GGM}_{\text{AES}}$ | | $\text{SBC}_{\text{FF}}$ $\text{GGM}_{\text{AES}}$ | |
|---|---|---|---|---|---|---|---|---|
| $D$ | $\tau$ | \|sgn\| | Sign | Verify | Sign | Verify | Sign | Verify |
| 8 | 16 | 6 222 B | 10.26 ms | 10.16 ms | 0.96 ms | 0.86 ms | 0.92 ms | 0.82 ms |
| 9 | 15 | 6 078 B | 18.86 ms | 18.52 ms | 1.16 ms | 1.06 ms | 1.07 ms | 0.98 ms |
| 10 | 13 | 5 484 B | 31.68 ms | 32.31 ms | 1.40 ms | 1.32 ms | 1.24 ms | 1.15 ms |
| 11 | 12 | 5 259 B | 57.84 ms | 57.74 ms | 1.99 ms | 1.91 ms | 1.64 ms | 1.56 ms |
| 12 | 11 | 5 002 B | 105.62 ms | 105.51 ms | 3.10 ms | 3.01 ms | 2.38 ms | 2.29 ms |
| 13 | 10 | 4 713 B | 192.19 ms | 191.27 ms | 5.39 ms | 5.30 ms | 3.75 ms | 3.64 ms |
| 15 | 9 | 4 536 B | 693.99 ms | 693.83 ms | 23.24 ms | 22.99 ms | 12.07 ms | 11.83 ms |
| 16 | 8 | 4 167 B | 1234.67 ms | 1233.38 ms | 41.18 ms | 40.82 ms | 21.71 ms | 21.40 ms |

**Table 4.5:** Running times for $\lambda = 128$ using $N = 2^D$ parties and $\tau$ rounds with $q = 2$, $k = 257$ and $n = 130$ for SBC comparing GGM tree versions.

| | | | SBC $\text{cGGM}_{\text{SHA}}$ | | SBC $\text{cGGM}_{\text{AES}}$ | | $\text{SBC}_{\text{FF}}$ $\text{cGGM}_{\text{AES}}$ | |
|---|---|---|---|---|---|---|---|---|
| $D$ | $\tau$ | \|sgn\| | Sign | Verify | Sign | Verify | Sign | Verify |
| 8 | 16 | 5 966 B | 9.34 ms | 9.23 ms | 0.95 ms | 0.85 ms | 0.91 ms | 0.81 ms |
| 9 | 15 | 5 838 B | 16.88 ms | 16.75 ms | 1.15 ms | 1.04 ms | 1.06 ms | 0.96 ms |
| 10 | 13 | 5 276 B | 28.62 ms | 28.58 ms | 1.38 ms | 1.29 ms | 1.22 ms | 1.13 ms |
| 11 | 12 | 5 067 B | 52.33 ms | 52.25 ms | 1.95 ms | 1.86 ms | 1.61 ms | 1.51 ms |
| 12 | 11 | 4 826 B | 95.44 ms | 95.15 ms | 3.03 ms | 2.94 ms | 2.31 ms | 2.22 ms |
| 13 | 10 | 4 553 B | 173.44 ms | 173.51 ms | 5.27 ms | 5.18 ms | 3.64 ms | 3.52 ms |
| 15 | 9 | 4 392 B | 628.41 ms | 627.91 ms | 22.77 ms | 22.54 ms | 11.71 ms | 11.46 ms |
| 16 | 8 | 4 039 B | 1116.88 ms | 1117.77 ms | 40.39 ms | 40.09 ms | 20.94 ms | 20.59 ms |

**Table 4.6:** Running times for $\lambda = 128$ using $N = 2^D$ parties and $\tau$ rounds with $q = 2$, $k = 257$ and $n = 130$ for SBC comparing cGGM tree versions.

| | | | $\text{SBC}^{\mathcal{H}(t_0)}$ $\text{GGM}_{\text{SHA}}$ | | $\text{SBC}^{\mathcal{H}(t_0)}$ $\text{GGM}_{\text{AES}}$ | | $\text{SBC}_{\text{FF}}^{\mathcal{H}(t_0)}$ $\text{GGM}_{\text{AES}}$ | |
|---|---|---|---|---|---|---|---|---|
| $D$ | $\tau$ | \|sgn\| | Sign | Verify | Sign | Verify | Sign | Verify |
| 8 | 16 | 5 708 B | 9.29 ms | 9.20 ms | 0.92 ms | 0.83 ms | 0.77 ms | 0.68 ms |
| 9 | 15 | 5 596 B | 16.83 ms | 16.74 ms | 1.10 ms | 1.01 ms | 0.91 ms | 0.83 ms |
| 10 | 13 | 5 066 B | 28.85 ms | 28.51 ms | 1.33 ms | 1.25 ms | 1.07 ms | 1.00 ms |
| 11 | 12 | 4 873 B | 52.17 ms | 52.31 ms | 1.90 ms | 1.82 ms | 1.46 ms | 1.39 ms |
| 12 | 11 | 4 649 B | 95.30 ms | 95.21 ms | 2.95 ms | 2.87 ms | 2.17 ms | 2.08 ms |
| 13 | 10 | 4 392 B | 173.14 ms | 172.91 ms | 5.15 ms | 5.08 ms | 3.46 ms | 3.36 ms |
| 15 | 9 | 4 247 B | 627.42 ms | 625.49 ms | 22.70 ms | 22.46 ms | 11.48 ms | 11.22 ms |
| 16 | 8 | 3 910 B | 1115.23 ms | 1113.22 ms | 40.60 ms | 40.23 ms | 20.12 ms | 19.78 ms |

**Table 4.7:** Running times for $\lambda = 128$ using $N = 2^D$ parties and $\tau$ rounds with $q = 2$, $k = 257$ and $n = 130$ for $\text{SBC}^{\mathcal{H}(t_0)}$ comparing GGM tree versions.

| | | | $\text{SBC}^{\mathcal{H}(t_0)}$ $\text{cGGM}_{\text{SHA}}$ | | $\text{SBC}^{\mathcal{H}(t_0)}$ $\text{cGGM}_{\text{AES}}$ | | $\text{SBC}_{\text{FF}}^{\mathcal{H}(t_0)}$ $\text{cGGM}_{\text{AES}}$ | |
|---|---|---|---|---|---|---|---|---|
| $D$ | $\tau$ | \|sgn\| | Sign | Verify | Sign | Verify | Sign | Verify |
| 8 | 16 | 5 452 B | 9.29 ms | 9.17 ms | 0.80 ms | 0.71 ms | 0.76 ms | 0.67 ms |
| 9 | 15 | 5 356 B | 16.78 ms | 16.64 ms | 0.97 ms | 0.89 ms | 0.90 ms | 0.81 ms |
| 10 | 13 | 4 858 B | 28.42 ms | 28.44 ms | 1.20 ms | 1.12 ms | 1.05 ms | 0.97 ms |
| 11 | 12 | 4 681 B | 51.96 ms | 52.06 ms | 1.75 ms | 1.67 ms | 1.42 ms | 1.34 ms |
| 12 | 11 | 4 473 B | 94.78 ms | 94.93 ms | 2.75 ms | 2.66 ms | 2.10 ms | 2.01 ms |
| 13 | 10 | 4 232 B | 177.58 ms | 172.76 ms | 4.77 ms | 4.67 ms | 3.33 ms | 3.23 ms |
| 15 | 9 | 4 103 B | 625.84 ms | 624.88 ms | 22.12 ms | 21.95 ms | 11.85 ms | 10.81 ms |
| 16 | 8 | 3 782 B | 1110.60 ms | 1112.82 ms | 39.33 ms | 39.02 ms | 19.32 ms | 19.00 ms |

**Table 4.8:** Running times for $\lambda = 128$ using $N = 2^D$ parties and $\tau$ rounds with $q = 2$, $k = 257$ and $n = 130$ for $\text{SBC}^{\mathcal{H}(t_0)}$ comparing cGGM tree versions.

CHAPTER 5

# VOLE-in-the-Head Signatures from SBC

This chapter is based on the Asiacrypt 2025 publication [HJ25] and its full version [HJ24b]. I am the main author of this publication, under the supervision of my PhD advisor Antoine Joux. To achieve a consistent notation throughout this dissertation, there are some minor notational changes compared to the original version.

## 5.1 Introduction

In this chapter, we consider the *Subfield Bilinear Collision* (SBC) problem and develop a *VOLE-in-the-Head* (VOLEitH) signature scheme based on it. In Chapter 4, it already appeared that the structure of the problem was highly suited to the MPCitH paradigm, leading to a simple and quite compact signature scheme. We show that the structure of the SBC problem also works very well in the context of *VOLE-in-the-Head* [BBD+23]. In fact, because of this, we do not need to use the full extent of the functionalities presented in [BBD+23]. Instead, we reconstruct a specific version of VOLEitH from scratch dedicated to the SBC problem in order to obtain a simplified scheme.

## 5.2 Contribution

In MPCitH based schemes, the use of puncturable PRFs (PPRFs) based on GGM trees [GGM86] is common to reduce the communication cost of the scheme. We introduce a new method to correlate multiple such GGM trees, which we call a *correlated GGM forest*, by using a layered construction of two state-of-the-art techniques from MPCitH protocols: the correlated GGM tree derivation [GYW+23] and the hypercube folding technique [AGH+23]. This method allows us to correlate the values obtained by the leaves of GGM trees between multiple rounds of the identification protocol inside the signature scheme, while having the same PPRF key size as state-of-the-art protocols. This is a general construction and is not tied to the use of the SBC problem, and can therefore be used for any multi-round MPCitH signature scheme.
The correlated forest can also be used in VOLEitH protocols. However, if we want to correlate values with a size larger than the internal nodes used in the GGM trees, we can derive them from the leaves of the tree and compute offsets to correct them to the desired value. In that case, we need to include a consistency check, which convinces the verifier that the values obtained between multiple rounds of the protocol coincide. A common method to perform this (probabilistic) consistency check is known as the SoftSpokenOT technique [Roy22] and is used in many VOLEitH schemes based on the transformation from [BBD+23]. In the context of the SBC problem, we simplify the consistency check in Section 5.4.5 and include it in the signature scheme.
Using the correlated GGM forest and this simplified consistency check, we introduce the

identification scheme for the SBC problem based on VOLEitH and use the Fiat-Shamir transformation to obtain a signature scheme. To further reduce the signature size, we introduce an alternative commitment to the cGGM forest called *MinMax commitment* in Section 5.7.1.

Compared to the SBC signature scheme based on MPCitH from Chapter 4, this approach leads to improvements in both signature size and running time. For a security level of $\lambda = 128$ bits, our smallest signature has a size of only 2 658 B, compared to 3 782 B for the smallest signature from Chapter 4.

## 5.3 Preliminaries

We introduce a slightly adjusted notation for sharings to use them in the GGM forest construction. We introduce the adapted notation that we use for the MPCitH setting and hypercube folding in Sections 5.3.1 and 5.3.2 before we present the VOLEitH protocol for SBC.

### 5.3.1 MPC-in-the-Head Setting

The construction of a ZK proof in Chapter 4 relies on the MPCitH paradigm, which was introduced in [IKOS07]. As in Chapter 4, we use additive sharings of finite field elements and finite field vectors. Let $N$ be the number of parties. An *$N$-sharing* of a finite field element $x \in \mathbb{F}$ is an $N$-tuple

$$[\![x]\!] = \left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right)$$

such that

$$x = \sum_{i=0}^{N-1} x^{[\![i]\!]} \quad (\text{in } \mathbb{F}).$$

We refer to $x^{[\![i]\!]}$ as the $i$-th *share* of $x$. Similarly, we use the notation

$$[\![\vec{x}]\!] = \left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![N-1]\!]}\right)$$

for a sharing of a finite field vector $\vec{x} \in \mathbb{F}^n$. In our setting, each (virtual) party receives one of the $N$ shares.

A sharing of a given value $x$ is usually obtained by computing $N-1$ random values $x^{[\![0]\!]}, \ldots, x^{[\![N-2]\!]}$, and then setting $x^{[\![N-1]\!]} = x - \sum_{i=0}^{N-2} x^{[\![i]\!]} \mod |\mathbb{F}|$. For our scheme, we often use a random oracle to derive the shares appearing in the protocol. In this situation, each share can be viewed as a random value, and summed together, they would share a random value. Thus, to obtain a sharing of the desired value, we need an *offset* value $\delta_x$, computed as

$$\delta_x = x - \sum_{i=0}^{N-1} x^{[\![i]\!]}.$$

We can then reconstruct $x$ using the formula

$$x = \delta_x + \sum_{i=0}^{N-1} x^{[\![i]\!]}.$$

This can also be viewed as an (exact) sharing of $x$ between $N+1$ parties. By introducing these offset values, we can use a random oracle to derive sharings if needed and simplify the notation used in our protocol description.

### 5.3.2 Hypercube technique

A prevalent tool to improve the efficiency of MPCitH-based schemes is the hypercube technique introduced in [AGH+23]. Let $N = 2^D$ be the number of parties. This approach transforms one instance of the protocol with $N = 2^D$ parties into $D$ instances of the protocol, each involving only 2 parties, while maintaining the same total soundness error as the original protocol. Let the $j$-th bit of the binary decomposition of an integer $i$ be $B_j(i)$. The process of converting one protocol instance with $N = 2^D$ parties into $D$ instances with 2 parties is referred to as *folding* of the shares. Consider the sharing of an element $x$ between $N$ parties of the form

$$x = \sum_{i=0}^{N-1} x^{[\![i]\!]}.$$

For any fixed value $j \in [D]$, we see that

$$\begin{aligned} x &= \sum_{B_j(i)=0} x^{[\![i]\!]} + \sum_{B_j(i)=1} x^{[\![i]\!]} \\ &= {x_j}^{[\![0]\!]} + {x_j}^{[\![1]\!]} \end{aligned}$$

is a sharing of $x$ between two parties. In the hypercube technique, $D$ such protocols between two parties are run in parallel, each with a soundness error of $\frac{1}{2}$. We use the notation $\left({x_j}^{[\![0]\!]}, {x_j}^{[\![1]\!]}\right)_{j\in[D]} \leftarrow \mathsf{Folding}\left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right)$ to denote this construction. For efficiency, we use the fast folding algorithm with a running time of $\mathcal{O}(N)$ described in Section 4.7.2 in our implementation.
The key point of the hypercube technique is that the choice of one missing share among $N$ is equivalent to the (independent) choice of one missing share in each derived binary sharing. Note that the correspondence between the missing positions simply comes from the binary decomposition of the missing index among $N$. Furthermore, the derived binary sharings are mutually independent.

### 5.3.3 Puncturable PRFs

To reduce communication cost, the $N$-sharings appearing in the construction are based on the use of puncturable pseudorandom functions (PPRFs). More specifically, we use the classical PPRF based on GGM trees [GGM86], which is mentioned for example in [BW13, KKW18]. Starting from a root seed, the idea is to build a binary tree with $N = 2^D$ leaves by recursively applying a length-doubling *pseudorandom generator* (PRG) to obtain the left and right children of each node. To reveal all $N$ leaves of such a binary tree except

one leaf $i^* \in [N]$, the idea is to reveal all the nodes of the siblings of the nodes on the path from the root seed to the leaf $i^*$. Using this method, all leaves except $i^*$ can be reconstructed by communicating $\log(N) = D$ nodes instead of $N - 1$ leaves.
In our scheme, we use *correlated* GGM (or cGGM) trees, as introduced in [GYW+23]. We extend the cGGM trees to construct the correlated GGM forest in Section 5.4.1. The idea behind regular cGGM trees is to build a binary tree where the sum of all nodes on each level is preserved as the tree is being constructed. In order to do that, it suffices to modify the length-doubling PRG so that the sum of the labels of the left and right descendants of any node equals the label of the node itself. Given the (salted) hash function $\mathcal{H}$ and any node $T$, this is easily achieved by setting the left child of $T$ to $\mathcal{H}(T)$ and the right child of $T$ to $T - \mathcal{H}(T)$. In the binary case $q = 2$, the right child is rewritten as $T \oplus \mathcal{H}(T)$, and we speak of an *XOR-preserving* GGM tree. Note that correlated GGM trees have a limitation. Namely, the size of the correlated value is limited by the size of the inner nodes of the tree, usually equal to $\lambda$ bits. As a consequence, if the tree is used to share more than $\lambda$ bits between the parties, it is necessary to compute and transmit offsets for the values beyond the first $\lambda$ bits. Of course, this constraint of having offsets does not apply to purely random values shared between the parties.
In order to improve the running time of the scheme for $\lambda = 128$, we use a cGGM tree construction based on the AES block cipher instead of a salted hash function. This technique was introduced in [BCC+24b] and takes full advantage of the AES instruction set available on many recent CPU architectures. In the standard GGM, we need two keys $K_0$ and $K_1$, the left child is set to $T \oplus \mathrm{AES}_{K_0}(T)$ and the right child is set to $T \oplus \mathrm{AES}_{K_1}(T)$. It is important to note that a direct application of AES would not lead to a secure PPRF construction because decryption would allow one to climb back to the root node. In order to achieve an XOR-preserving tree, we still use two keys and define the left child to be $\mathrm{AES}_{K_0}(T) \oplus \mathrm{AES}_{K_1}(T)$, which leads to a right child of $T \oplus \mathrm{AES}_{K_0}(T) \oplus \mathrm{AES}_{K_1}(T)$.

## 5.4 VOLE Identification Protocol Based on SBC

Consider an NSBC instance $\mathrm{NSBC}[\vec{u}, \vec{v}]$ for $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$. Based on the cGGM tree construction described in Section 5.3.3, we build an interactive Zero-Knowledge proof of knowledge of a solution $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. As previously explained, the normalized vectors $\vec{x}'$ and $\vec{y}'$ are written in the form $\vec{x}' = (\vec{x}, 1, 0)$ and $\vec{y}' = (\vec{y}, 0, 1)$ for $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$. For simplicity, we only consider the case $q = 2$ for this construction. In other words, the secret key of our scheme consists of the $2n - 4$ bits forming $\vec{x}$ and $\vec{y}$.

### 5.4.1 Correlating Multiple cGGM Trees

To design our scheme, we need multiple (different) sharings of the vectors $\vec{x}$ and $\vec{y}$ between 2 parties in the hypercube setting. More precisely, the number of binary sharings can never be lower than the desired security level $\lambda$. Furthermore, depending on the details of the scheme and of its security proof, a few extra sharings might be needed.
In particular, we need to be able to construct a large number of (independent) sharings of $\vec{x}$ between 2 parties. When doing that, it is essential to ensure that the same value of $\vec{x}$ is used everywhere. With a standard cGGM tree with $N = 2^D$ parties, we obtain $\log(N) = D$ such sharings from the hypercube folding. And, by construction, both the

prover and verifier have the guarantee that all sharings correspond to the same vector $\vec{x}$. Unfortunately, $N$ cannot be arbitrarily large, since it would make the computation costs too high. As a consequence, repetition is needed to obtain a large enough number of sharings. Using $\tau$ repetitions, we obtain a total of $\tau \cdot D$ binary sharings, and to ensure the desired security level, we require $\tau \cdot D \geq \lambda$.
In this section, we introduce a technique that allows us to guarantee the equality of the vector $\vec{x}$ across several distinct cGGM trees. This is achieved by a layered construction: first, an initial cGGM tree is built, followed by the hypercube folding, which is used to initialize a second level consisting of $\tau$ distinct cGGM trees.
First, we construct the initial cGGM tree, which we also call the pre-tree, that contains $2^\tau$ leaves, where every level sums to $\vec{x}$, using a bitwise XOR on $\lambda$ bits. To initialize this tree, we generate a random initial value $R_{\vec{x}}$ and set the two sibling elements on the second level of the tree as $R_{\vec{x}}$ and $R_{\vec{x}} \oplus \vec{x}$. The root node is unused and can remain unassigned. The XOR of these two nodes is $\vec{x}$ by design. From them, we compute the next levels of a XOR-preserving cGGM tree $T_{\vec{x}_{\text{pre}}}$ with $2^\tau$ leaves. We call this functionality the generation of the pre-tree and denote it by $T_{\vec{x}_{\text{pre}}} \leftarrow \mathsf{cGGM}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$. If we use a salted hash function denoted by $\mathcal{H}^{\mathsf{salt}}(\mathsf{msg}) \coloneqq \mathcal{H}(\mathsf{salt} \parallel \mathsf{msg})$ for a random value $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$ to derive the internal nodes of the tree, we use the subscript $\mathsf{cGGM}_{\mathsf{salt}}$ instead. Let $(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]})$ be the leaves of $T_{\vec{x}_{\text{pre}}}$.
Using the hypercube folding for these leaves, we obtain $\tau$ (pre) sharings of $\vec{x}$ between two parties of the form $[\![\vec{x}]\!]_j = \left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}\right)$ for $j \in [\tau]$ using $\mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right)$. By construction, we have that $\vec{x}^{[\![0]\!]_j} \oplus \vec{x}^{[\![1]\!]_j} = \vec{x}$ for $j \in [\tau]$.
We can use each of these sharings to initialize a second level containing $\tau$ new correlated trees $T_{\vec{x}_j}$ for $j \in [\tau]$. Each of these trees can then be fully computed with the cGGM technique. We thus obtain $\tau$ trees $T_{\vec{x}_0}, \ldots, T_{\vec{x}_{\tau-1}}$ with $2^D$ leaves each using the function $T_{\vec{x}_j} \leftarrow \mathsf{cGGM}(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}, 2^D)$ for each $j \in [\tau]$. The leaves of tree $T_{\vec{x}_j}$ are denoted by $\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right)$. By using the hypercube folding on the leaves of each of these trees again, we obtain $D$ sharings of $\vec{x}$ between 2 parties for each of the $\tau$ trees. We denote this by $\left(\vec{x}_j^{[\![0]\!]_i}, \vec{x}_j^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right)$ for $j \in [\tau]$. We call the family of cGGM trees containing $T_{\vec{x}_{\text{pre}}}$ and $T_{\vec{x}_0}, \ldots, T_{\vec{x}_{\tau-1}}$ a *correlated GGM forest*.
Since we use XOR-preserving cGGM trees on both layers of our construction, this is guaranteed to yield a total of $\tau \cdot D$ hypercube sharings of the same value $\vec{x}$.
To simplify the notation, we renumber the $\tau \cdot D$ sharings obtained in this way and simply write them as:

$$\vec{x}_m^{[\![0]\!]} \oplus \vec{x}_m^{[\![1]\!]} = \vec{x} \quad \text{for each } m \in [\tau \cdot D].$$

In addition, by using a pseudorandom generator to expand the leaves of each tree $T_{\vec{x}_j}$ into more than $\lambda$ bits, it is easy to produce shares for additional values in each tree. Note, however, that these extra values are not correlated. As a consequence, they might require offsets and extra precautions to prove equality across trees.
To illustrate the idea of the construction for the correlated GGM forest, we include a diagram of the resulting structure for a small depth tree example in Figure 5.1.

### 5.4.2 Punctured Keys for the Correlated GGM Forest

When giving out the punctured key for a correlated GGM forest, we need to do it in a way that guarantees to the verifier that the construction was properly used. Thus, it is

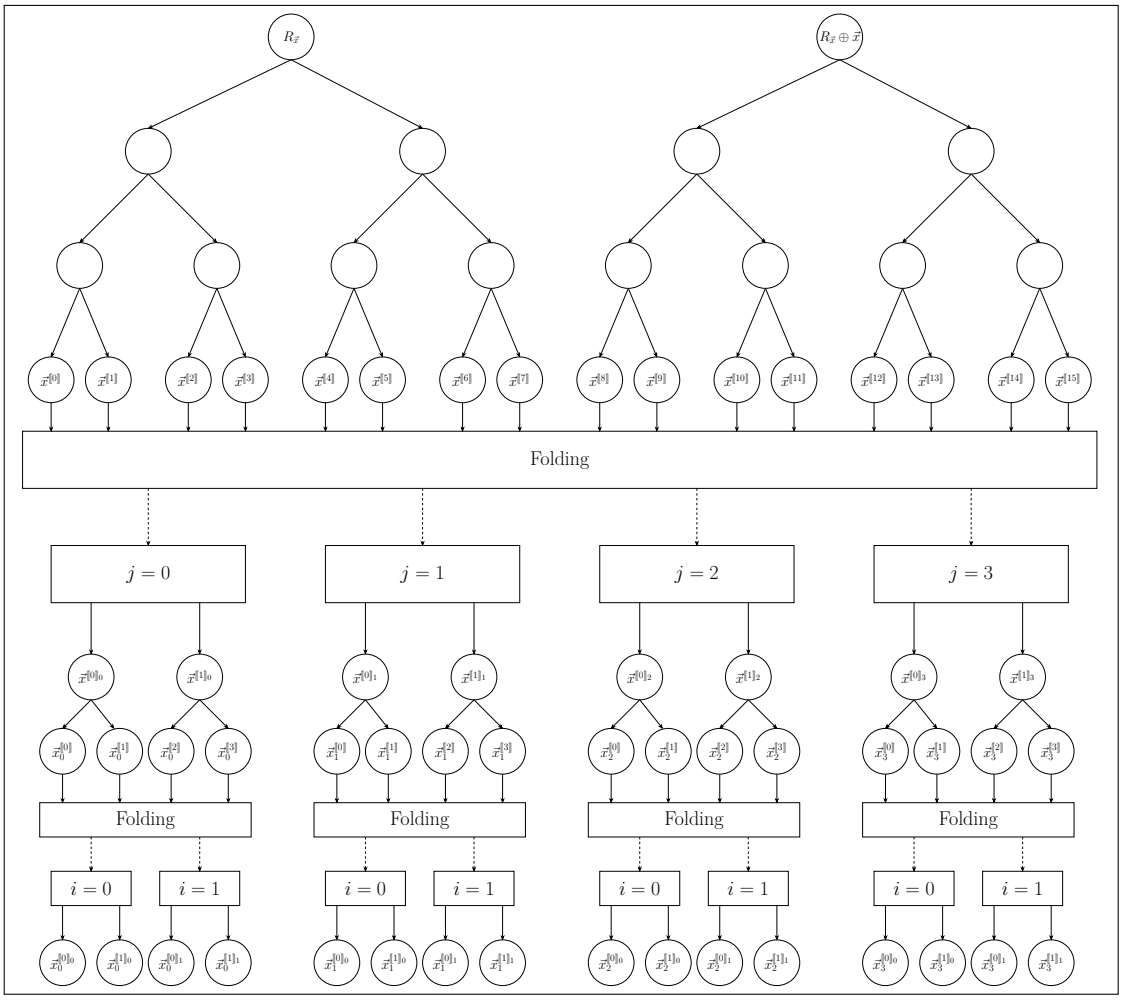


**Figure 5.1:** Illustration of the correlated GGM forest for $\tau = 4$ and $D = 2$ resulting in 8 hypercube sharings of $\vec{x}$.

not sufficient to independently give a punctured key for each tree from the second level. Indeed, such an opening would be possible even with completely independent trees. Thus, equality across trees would not be guaranteed. Instead, we first give a punctured key for the pre-tree. From this punctured key, the verifier can recover all the leaves of the pre-tree but one. Therefore, after hypercube folding, the verifier learns exactly one node on the second level from each of the $\tau$ trees. Note that these nodes would normally be part of the punctured keys of the second level trees, so this technique allows us to remove one element from each of the second level punctured keys. Therefore, proceeding in this manner does not modify the overall size of the global punctured key. Indeed, we remove $\tau$ elements and replace them by the punctured key of the pre-tree, which contains $\tau$ nodes. To locate the missing position that is needed in the pre-tree, it suffices to assemble the high-order bits of the missing nodes of each second-level tree into a $\tau$-bit number.
In Figure 5.2, we give an example of the punctured key for a correlated GGM forest which has $\tau = 4$ second layer trees with $N = 2^2$ leaves each. For this example, we assume that we receive the challenges $i_0^* = 0, i_1^* = 1, i_2^* = 3, i_3^* = 1$, indicated by the red leaves. First, we compute the corresponding challenge in the pre-tree as $i_p^* = \sum_{j=0}^{\tau-1} \lfloor i_j^* / 2^{D-1} \rfloor \cdot 2^j =$

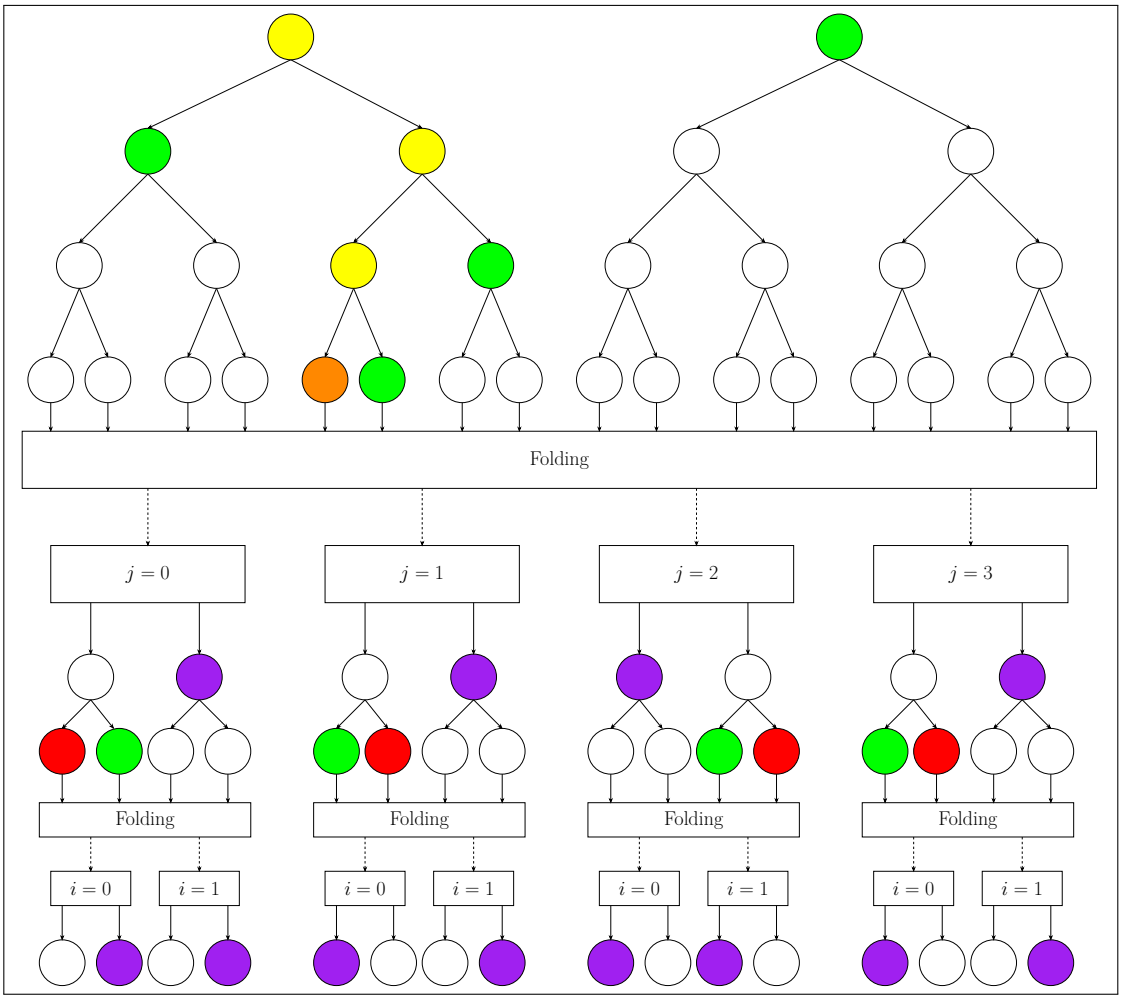


**Figure 5.2:** Example of the punctured key for correlated GGM forest for $\tau = 4$ and $D = 2$. The challenges are indicated by red leaves, while the corresponding challenge in the pre-tree is colored orange. The yellow nodes correspond to the path from the root to the pre-tree challenge. The PPRF key contains all of the green nodes. The purple nodes can be recomputed with the Folding algorithm.

$0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 = 4$. Therefore, we need to compute the PPRF key for the pre-tree at position $i_p^* = 4$, which is colored orange. To do that, we first consider the path from the root of the left half of the pre-tree to $i_p^*$ (indicated by yellow nodes) and include the siblings of these nodes in the PPRF key $K_{i_p^*}$. We indicate them in green. With $K_{i_p^*}$, we can recompute all leaves of the pre-tree except at position $i_p^* = 4$. Using the hypercube folding, we can recompute one of the two hypercube shares for each index $j \in \{0, 1, 2, 3\}$. These nodes, colored purple, therefore do not have to be included in the PPRF key. Note that, since the leaf of the pre-tree at position $i_p^* = 4$ is missing, we cannot compute the values of the siblings of these purple nodes. Lastly, we have to include the siblings of the remaining nodes on the paths to the red challenge leaves for each second layer tree to the PPRF key, which are also colored in green. Using the Folding algorithm again, we can recompute one hypercube share for each index $i \in [D]$, which are also colored purple. In

particular, for our example in Figure 5.2, the hypercube sharings for $i = 0$ after folding have the form $\left(\vec{x}_j^{[\![0]\!]_0}, \vec{x}_j^{[\![1]\!]_0}\right) = \left((\vec{x}_j^{[\![0]\!]} + \vec{x}_j^{[\![2]\!]}), (\vec{x}_j^{[\![1]\!]} + \vec{x}_j^{[\![3]\!]})\right)$, while for $i = 1$ they have the form $\left(\vec{x}_j^{[\![0]\!]_1}, \vec{x}_j^{[\![1]\!]_1}\right) = \left((\vec{x}_j^{[\![0]\!]} + \vec{x}_j^{[\![1]\!]}), (\vec{x}_j^{[\![2]\!]} + \vec{x}_j^{[\![3]\!]})\right)$. Depending on which leaf is missing in tree $T_{\vec{x}_j}$, we can only compute one out of the two sums in each such sharing. As claimed, this PPRF key can be used to recompute a total of $8 = \tau \cdot D$ hypercube shares.

### 5.4.3 From Sharings to Vector Oblivious Linear Evaluation (VOLE)

By using the correlated GGM forest described in the previous section, the prover always knows the complete trees, while the verifier only lacks one leaf from each tree (including the pre-tree). At the level of the sharings, this means that for every $m \in [\tau \cdot D]$, the verifier knows only one value $\vec{x}_m^{[\![b_m]\!]}$ (out of two) for a bit $b_m$ of its choice, while the prover knows both.
It is equivalent and convenient to say that the prover creates an affine vectorial function:

$$f_m(b_m) = \vec{x}_m^{[\![0]\!]} \oplus b_m \cdot \vec{x}$$

and that the verifier chooses $b_m$ and learns $f_m(b_m) = \vec{x}_m^{[\![b_m]\!]}$.
With this view in mind, we can now pick a series of random coefficients $\alpha_m$ in the large field $\mathbb{F}_{2^k}$ and consider the function:

$$\vec{F} = \sum_{m=0}^{\tau D-1} \alpha_m f_m(b_m) = \sum_{m=0}^{\tau D-1} \alpha_m \vec{x}_m^{[\![0]\!]} + \left[\sum_{m=0}^{\tau D-1} \alpha_m b_m\right] \cdot \vec{x}.$$

Note that $\vec{F}$ is an affine function known to the prover, and that the verifier chooses the evaluation point $\Delta = \sum_{m=0}^{\tau D-1} \alpha_m b_m$ and learns the value $\vec{F}(\Delta)$ obtained by evaluating $\vec{F}$ at this point.
In principle, the coefficients $\alpha_m$ could be pre-specified as part of the scheme. However, to avoid casting suspicion on the choice of coefficients, we feel that it is preferable to have them chosen at random by the honest verifier.

### 5.4.4 A Simple but Inefficient Construction

As already mentioned, cGGM trees with internal nodes of size $\lambda$ bits are limited and can only produce correlated values up to $\lambda$ bits. Unfortunately, the secret key of the SBC problem contains a total of $2\lambda$ bits, namely $\vec{x}$ and $\vec{y}$. To illustrate our construction in a simple but inefficient way, let us use the construction of Section 5.4.1 twice in order to independently get $\tau \cdot D$ sharings for both $\vec{x}$ and $\vec{y}$. We write them as:

$$\vec{x}_m^{[\![0]\!]} \oplus \vec{x}_m^{[\![1]\!]} = \vec{x} \quad \text{and} \quad \vec{y}_m^{[\![0]\!]} \oplus \vec{y}_m^{[\![1]\!]} = \vec{y} \quad \text{for each } m \in [\tau \cdot D]. \tag{5.1}$$

Proceeding as in Section 5.4.3, we use these two cGGM forests to define two vectorial functions $\vec{F}_x, \vec{F}_y : \mathbb{F}_{2^k} \to (\mathbb{F}_{2^k})^{n-2}$:

$$\begin{aligned} \vec{F}_x(\Delta) &= \vec{A}_{F_x} + \Delta \cdot \vec{x}, \\ \vec{F}_y(\Delta) &= \vec{A}_{F_y} + \Delta \cdot \vec{y}, \end{aligned}$$

whose constant coefficients are given by

$$\vec{A}_{F_x} \;=\; \vec{F}_x(0) = \sum_{m=0}^{\tau\cdot D-1} \alpha_m \, \vec{x}_m^{[\![0]\!]} \quad \text{and} \quad \vec{A}_{F_y} = \vec{F}_y(0) = \sum_{m=0}^{\tau\cdot D-1} \alpha_m \, \vec{y}_m^{[\![0]\!]}.$$

Note that, since we want to evaluate both functions at the same $\Delta$, the missing tree leaves in both forests are located in the same manner.
Before learning the evaluation point $\Delta$, the prover computes the polynomial $\Phi$ defined as:

$$\begin{aligned}
\Phi(\Delta) \;:=\; & \left(\vec{u}\cdot\begin{pmatrix}\vec{F}_x(\Delta)\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_y(\Delta)\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{F}_y(\Delta)\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_x(\Delta)\\1\\0\end{pmatrix}\right) \\
= \; & \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right) \\
& + \Bigg[\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right) + \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right) \\
& - \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right) - \left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)\Bigg]\Delta \\
& + \left[\left(\vec{u}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)\right]\Delta^2.
\end{aligned}$$

**Remark 5.1.** Note that $\Phi$ is of the form

$$\Phi(\Delta) = A + B\Delta$$

if and only if $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ for $\vec{x}' = (\vec{x}, 1, 0)$, $\vec{y}' = (\vec{y}, 0, 1)$. The coefficients $A, B \in \mathbb{F}_{2^k}$ are given by

$$\begin{aligned}
A \;=\; & \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right), \\
B \;=\; & \left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right) + \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right) \\
& - \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right) - \left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right).
\end{aligned}$$

We use this crucial property of $\Phi$ in our protocol.

The prover sends the coefficients $A, B \in \mathbb{F}_{2^k}$ to the verifier. The verifier responds by sending random challenges $i_0^*, \ldots, i_{\tau-1}^* \in [2^D]$ back, one for each tree. The prover returns the PPRF keys $K_{i_j^*}$ for the trees $T_{\vec{x}_j}$ and $T_{\vec{y}_j}$ (and the corresponding pre-trees) for each

$j \in [\tau]$ to the verifier.
As explained in Section 5.4.3, this corresponds to the verifier choosing an evaluation point

$$\Delta^* \coloneqq \sum_{m=0}^{\tau \cdot D-1} b_m \alpha_m,$$

and obtaining the values $\vec{F}_x(\Delta^*)$ and $\vec{F}_y(\Delta^*)$.
Equipped with this knowledge, the verifier can compute $\Phi(\Delta^*)$ as

$$\Phi(\Delta^*) = \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right).$$

The verifier then checks whether $\Phi(\Delta^*) = A + B\Delta^*$ and accepts or rejects accordingly.

**Remark 5.2.** The protocol accepts a wrong instance of vectors $(\vec{x}', \vec{y}') \notin \mathrm{NSBC}[\vec{u}, \vec{v}]$ if $\Delta^*$ is a root of the polynomial

$$\Phi(\Delta) - B\Delta - A.$$

For $(\vec{x}', \vec{y}') \notin \mathrm{NSBC}[\vec{u}, \vec{v}]$, $\Phi(\Delta)$ is a quadratic polynomial in $\Delta$. Therefore, $\Phi(\Delta) - B\Delta - A$ has at most two roots. Since the evaluation point is chosen from a set of cardinality $2^{\tau D}$, the probability of getting a false positive is bounded by $2^{1-\tau D}$.

### 5.4.5 Consistency Checks

In Section 5.4.4, we use two independent cGGM forests. One to derive $\tau\, D$ sharings of $\vec{x}$ and one to derive $\tau\, D$ sharings of $\vec{y}$. Despite its simplicity, this approach has a very high cost, since the need for opening two forests using a puncturable PRF doubles the amount of data. In this section, we address this problem by implementing the same basic idea while using a single cGGM forest, based on $\vec{x}$. This leads to a significant decrease in the communication cost for the scheme but requires additional techniques.
As mentioned in Section 5.4.1, we can use the cGGM forest provided to share $\vec{x}$ to create sharings of extra values, simply by applying a PRG to extend the size of individual leaves. However, this creates two problems. First, every tree induces sharings of random values, thus offsets are required to set the sharings to the desired value. As a consequence, each of the $\tau$ cGGM trees created to share $\vec{x}$ can be made to share $\vec{y}$ by providing a corresponding offset value. Since the communication costs for these offsets are much lower than for an extra cGGM forest, this is worthwhile in terms of signature size.
The second problem is more subtle and harder to fix. By providing an offset for $\vec{y}$, we can make sure that each of the $\tau$ cGGM trees provides $D$ sharings of a value $\vec{y}_i$. However, nothing guarantees, from the verifier's viewpoint, that all these values are identical. This could be exploited by a cheating prover to boost the cheating probability. To remedy this, we need to add a consistency check that lets the verifier check that all the $\vec{y}_i$ for $i \in [\tau]$ are really equal. This idea of performing a (probabilistic) consistency check is known as the SoftSpokenOT technique [Roy22]. In our case, it can be somewhat simplified, and we now describe our version in detail.
For the purpose of realizing the equality testing, we extend each leaf in the cGGM forest created to share $\vec{x}$ in order to produce additional sharings of two additional elements: the

second secret value $\vec{y}$ and an extra element $z$ from $\mathbb{F}_{2^k}$. Since $\vec{y}$ is fixed, one offset is required for each of the $\tau$ trees. For $z$, since a random element is requested, it is tempting to assume that no offsets are needed. However, to prove equality between sharings of $\vec{y}$, we also need the sharings of $z$ to be equal across the trees. As a consequence, we can only skip the offset corresponding to $z$ for the first of the cGGM trees in the family.
More precisely, from each leaf $\vec{x}_j^{[\![i]\!]} \in (\mathbb{F}_2)^{n-2}$, the prover derives the shares $\vec{y}_j^{[\![i]\!]} \in (\mathbb{F}_2)^{n-2}$ and ${z_j}^{[\![i]\!]} \in \mathbb{F}_{2^k}$ for each $(i,j) \in [2^D] \times [\tau]$, using the random oracle $\mathcal{H}$. To commit to the family of $\tau$ subtrees, the prover sends the offset values $\delta_{\vec{y}_j} \in (\mathbb{F}_2)^{n-2}$ and $\delta_{z_j} \in \mathbb{F}_{2^k}$ such that

$$\vec{y}_j = \sum_{i=0}^{2^D-1} \vec{y}_j^{[\![i]\!]} + \delta_{\vec{y}_j} \text{ and } z_j = \sum_{i=0}^{2^D-1} {z_j}^{[\![i]\!]} + \delta_{z_j} \text{ for } j \in [\tau], \tag{5.2}$$

to the verifier. Note that we set $\delta_{z_0}$ to 0 and do not send this value.
After the prover has committed to the cGGM trees, the verifier computes a random vector $\vec{\mu} \in (\mathbb{F}_{2^k})^{n-2}$ and sends it to the prover. The goal of $\vec{\mu}$ is to provide an efficient check that the pairs $(\vec{y}_j, z_j)$ are equal among the rounds. Since $\vec{\mu}$ is selected randomly and not under the control of the prover, it suffices to check that $z_j + \vec{\mu} \cdot \vec{y}_j$ is constant across the rounds.

**Remark 5.3.** Note that the goal of $z$ is to act as a one-time-pad to hide the value $\vec{\mu} \cdot \vec{y}$ which would, if revealed, leak information about the secret $\vec{y}$.

The equality check we use relies on the following lemma.

**Lemma 5.4.** Let $\vec{y}, \vec{y'} \in (\mathbb{F}_2)^{n-2}$ and $z, z' \in \mathbb{F}_{2^k}$ with $(\vec{y}, z) \neq (\vec{y'}, z')$ and let $\vec{\mu} \leftarrow_{\$} (\mathbb{F}_{2^k})^{n-2}$. Then $\Pr\left[z + \vec{\mu} \cdot \vec{y} = z' + \vec{\mu} \cdot \vec{y'}\right] \leq 2^{-k}$, where the probability is on the choice of $\vec{\mu}$.

*Proof.* Since $(\vec{y}, z) \neq (\vec{y'}, z')$, we have:

$$(\vec{y} \oplus \vec{y'}, z \oplus z') \neq (\vec{0}, 0).$$

To analyze the probability $\Pr\left[z + \vec{\mu} \cdot \vec{y} = z' + \vec{\mu} \cdot \vec{y'}\right]$, we consider two cases:

1. Assume $\vec{y} \oplus \vec{y'} = \delta_{\vec{y}} \neq \vec{0}$. In this situation, the probabilistic event occurs when $\delta_{\vec{y}} \cdot \vec{\mu} = z \oplus z'$. Equivalently, $\vec{\mu}$ belongs to a given affine hyperplane orthogonal to $\delta_{\vec{y}}$, which occurs with (conditional) probability $2^{-k}$.

2. Assume $\vec{y} = \vec{y'}$ and $z \neq z'$, thus $z + \vec{\mu} \cdot \vec{y} \neq z' + \vec{\mu} \cdot \vec{y'}$. In this case, the (conditional) probability is 0.

The statement follows by combining both cases. □

We now go back to the construction of the VOLE for $\vec{y}$ and $z$. A priori, the verifier is not sure that $\vec{y}$ and $z$ are equal in all trees, and the construction from Section 5.4.3 cannot be used without precaution. However, all the binary sharings derived from an individual tree correspond to equal values $(\vec{y}_j, z_j)$, where $j \in [\tau]$ denotes the index of the tree in its family. Thus, the prover can compute a VOLE instance (with smaller evaluation domain) for $(\vec{y}_j, z_j)$ for each $j \in [\tau]$. More precisely, after folding, each tree provides $D$ binary sharings:

$$\left(\vec{y}_j^{[\![0]\!]_i}, {z_j}^{[\![0]\!]_i}\right) \oplus \left(\vec{y}_j^{[\![1]\!]_i}, {z_j}^{[\![1]\!]_i}\right) \oplus (\delta_{\vec{y}_j}, \delta_{z_j}) = (\vec{y}_j, z_j).$$

As in Section 5.4.4, for each value $(i,j) \in [D] \times [\tau]$ the prover defines a linear polynomial:

$$f_{i,j}(b) = \left(\vec{y}_j^{[\![0]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + b \cdot (\vec{y}_j, z_j)\,.$$

We have grouped $\vec{y}_j$ and $z_j$ in the above polynomial for compactness. However, by abuse of notation and for the sake of convenience, we might refer to it as a polynomial in $\vec{y}_j$ or $z_j$ alone. We continue as in Section 5.4.4, and combine the $D$ polynomials in $\vec{y}_j$ and $z_j$ together. This defines a polynomial:

$$\vec{F}^{(j)}_{y,z}(\Delta_j) = \sum_{i=0}^{D-1} \alpha_{D\,j+i}\, f_{i,j}(b_{i,j}) = \sum_{i=0}^{D-1} \alpha_{D\,j+i} \left(\vec{y}_j^{[\![0]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + \Delta_j \cdot (\vec{y}_j, z_j)\,,$$

where $\Delta_j = \sum_{i=0}^{D-1} \alpha_{D\,j+i}\, b_{i,j}$. As before, we can write this as

$$\vec{F}^{(j)}_{y,z}(\Delta_j) = \vec{A}^{(j)}_{F_{y,z}} + \Delta_j \cdot (\vec{y}_j, z_j), \tag{5.3}$$

where the coefficient $\vec{A}^{(j)}_{F_{y,z}} = \vec{F}^{(j)}_{y,z}(0)$ is given by the evaluation at 0.
To implement the test provided by Lemma 5.4, we combine $\vec{\mu}$ with Equation (5.3) and find:

$$\begin{aligned} \vec{F}^{(j)}_{y,z}(\Delta_j) \cdot (\vec{\mu}, 1) &= \vec{A}^{(j)}_{F_{y,z}} \cdot (\vec{\mu}, 1) + \Delta_j (\vec{y}_j, z_j) \cdot (\vec{\mu}, 1) \\ &= \vec{A}^{(j)}_{F_{y,z}} \cdot (\vec{\mu}, 1) + \Delta_j [z_j + \vec{y}_j \cdot \vec{\mu}]. \end{aligned}$$

To understand the goal of the protocol, let us first **assume** that $\vec{y}_j$ and $z_j$ are **equal across trees**. Then the coefficient in front of $\Delta_j$ is the same for all values of $j \in [\tau]$. As a consequence, we ask that the prover computes and transmits this coefficient $b = z + \vec{y} \cdot \vec{\mu} \in \mathbb{F}_{2^k}$, where $\vec{y}$ and $z$ denote the (putative) common values across trees. The prover also computes and transmits the coefficients $a^{(j)} = \vec{A}^{(j)}_{F_{y,z}} \cdot (\vec{\mu}, 1) \in \mathbb{F}_{2^k}$ for all $j \in [\tau]$.
As usual, the verifier sends the challenges $i^*_j \in [2^D]$ for each subtree $T_j$ and obtains the corresponding punctured key from the prover, thus learning the evaluations $\vec{F}^{(j)}_{y,z}(\Delta^*_j)$ for each of the $\tau$ trees. Let us remark that if $\vec{y}_j$ and $z_j$ are equal across trees, then we can add these evaluations and obtain:

$$\vec{F}_{y,z}(\Delta^*) = \sum_{j=0}^{\tau-1} \vec{F}^{(j)}_{y,z}(\Delta^*_j) = \vec{F}_{y,z}(0) + \Delta^* (\vec{y}, z), \text{ where } \Delta^* = \sum_{j=0}^{\tau-1} \Delta^*_j.$$

In particular, the verifier learns $\vec{F}_y(\Delta^*)$ and also $\vec{F}_x(\Delta^*)$ (from the correlated part of the family of trees).
Thus, as in the simplified version of Section 5.4.4, the verifier computes

$$\Phi(\Delta^*) = \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right),$$

checks whether it is equal to $A + B\Delta^*$ and accepts or rejects accordingly.
If we **no longer assume equality across trees**, the verifier has to perform a little extra work. Namely, after obtaining each partial evaluation $\vec{F}^{(j)}_{y,z}(\Delta^*_j)$, the verifier computes

$\vec{F}^{(j)}_{y,z}(\Delta^*_j) \cdot (\vec{\mu}, 1)$ and checks that the value is equal to $a^{(j)} + b\Delta^*_j$. If any of these checks fail, the verifier rejects. If all checks succeed, the verifier continues with the computation described in the case of equality, computing and checking $\Phi(\Delta^*)$. The following theorem states that this provides a sound protocol.

**Theorem 5.5.** The soundness error of the NSBC identification scheme described above using $N = 2^D$ parties and $\tau$ rounds is bounded by $\tau^2\, 2^{-k} + 2^{1-\tau \cdot D}$.

*Proof.* Consider a prover who does not share the same value for $(\vec{y}, z)$ in every round. The values $(\vec{y}_i, z_i)$ can be partitioned into at most $\tau$ distinct subsets. In particular, we have that

$$\{(\vec{y}_0, z_0), \ldots, (\vec{y}_{\tau-1}, z_{\tau-1})\} = S^{(0)} \dot{\cup} \ldots \dot{\cup} S^{(s-1)}, \tag{5.4}$$

where $s \leq \tau$. For any fixed index $j \in [s]$, we have that $(\vec{y}_{i_0}, z_{i_0}) = (\vec{y}_{i_1}, z_{i_1})$ for all $(\vec{y}_{i_0}, z_{i_0}), (\vec{y}_{i_1}, z_{i_1}) \in S^{(j)}$ by definition. Moreover, for any $j_0, j_1 \in [s]$ with $j_0 \neq j_1$, we have that $S^{(j_0)} \cap S^{(j_1)} = \emptyset$.
From Lemma 5.4 and the union bound, we know that

$$\Pr\left[\exists i \neq j | (\vec{y}_i, z_i) \in S^{(i)}, (\vec{y}_j, z_j) \in S^{(j)} \text{ and } z_i + \vec{y}_i \cdot \vec{\mu} = z_j + \vec{y}_j \cdot \vec{\mu}\right] \leq \tau^2\, 2^{-k}.$$

This *bad event* provides the term $\tau^2\, 2^{-k}$ in the soundness error.
We now assume that the bad event has not occurred and without loss of generality, we consider that the reference value is $(\vec{y}_0, z_0) \in S^{(0)}$ and that during the protocol the prover sends the value $b = z_0 + \vec{y}_0 \cdot \vec{\mu}$. Since we excluded the above bad event, we know that $z_j + \vec{y}_j \cdot \vec{\mu} \neq b$ for all $(\vec{y}_j, z_j) \in S^{(i)}$, where $i \in \{1, \ldots, s-1\}$.
Consider a subtree $T_j$ for $j \in [\tau]$ with $(a^{(j)}, b^{(j)}) \neq (a^{(j)}, b)$. For this index, we see that $a^{(j)} + b\Delta_j = z_j + \vec{\mu} \cdot \vec{F}^{(j)}_y(\Delta_j)$ for at most one value $\Delta_j$. Indeed, the affine function $a^{(j)} + b\Delta_j - z_j - \vec{\mu} \cdot \vec{F}^{(j)}_y(\Delta_j)$ is non zero and thus has at most one root.
In order for the verifier to validate the equality testing for this index $j$, a cheating prover must guess the evaluation point $\Delta^*_j$ in advance and provide a corresponding $a^{(j)}$. We denote the subset of $[\tau]$ containing the indices $j$ with $z_j + \vec{y}_j \cdot \vec{\mu} \neq b$ as $I_{\text{bad}}$. In particular,

$$I_{\text{bad}} \coloneqq \{j \in [\tau] | z_j + \vec{y}_j \cdot \vec{\mu} \neq b\}.$$

Since each set of possible evaluation points for the partial VOLE evaluation of any single tree contains $2^D$ points, the probability for the prover to correctly guess all evaluation points for $I_{\text{bad}}$ is:

$$\Pr[\text{Prover guesses all } \Delta_j \text{ for } j \in I_{\text{bad}}] = 2^{-D \cdot |I_{\text{bad}}|}.$$

If the prover guesses correctly, we can treat all the $\Delta_j$ for $j \in I_{\text{bad}}$ as constant values. As a consequence, the linear polynomials that are constructed in this situation only depend on the remaining indices $I_{\text{good}} \coloneqq [\tau] \setminus I_{\text{bad}}$. More precisely, everything can be expressed in terms of the variable:

$$\Delta_g = \sum_{j \in I_{\text{good}}} \Delta_j.$$

In particular, we can view $\Phi$ as a polynomial in $\Delta_g$. Furthermore, we can see that $\Phi(\Delta_g)$ is an affine polynomial if and only if $(\vec{x}', \vec{y}'_g) \in \text{NSBC}[\vec{u}, \vec{v}]$ for $\vec{x}' = (\vec{x}, 1, 0), \vec{y}'_g = (\vec{y}_g, 0, 1)$.

As a consequence, for a cheating prover that does not know a solution to the NSBC instance, we know that $\Phi(\Delta_g)$ is a non-zero polynomial of degree 2. Thus, the linearity check on $\Phi$ can only work at two exceptional points (coming from a quadratic equation). Thus, the probability that this test succeeds for a cheating prover is upper bounded by $\frac{2}{2^{D\cdot|I_{\text{good}}|}} = 2^{1-D\cdot|I_{\text{good}}|}$. Since $\tau = |I_{\text{good}}|+|I_{\text{bad}}|$ and since the equality test for $I_{\text{bad}}$ and the linearity test for $I_{\text{good}}$ are independent, the overall success probability of both is upper bounded by:

$$2^{-D\cdot|I_{\text{bad}}|} \cdot 2^{1-D\cdot|I_{\text{good}}|} = 2^{1-\tau\cdot D}.$$

Adding this to the probability of the bad event described earlier, we conclude that the soundness error of the identification scheme is bounded by:

$$\tau^2\, 2^{-k} + 2^{1-\tau\cdot D}.$$

□

## 5.5 Detailed Description of the Identification Scheme

For completeness, we now provide a detailed description of our five-round identification protocol between the prover $\mathcal{P}$ and the honest verifier $\mathcal{V}$. This protocol uses six algorithmic sub-routines, executed in alternation by the prover and the verifier. These algorithms are: Commitment, First Challenge, First Response, Second Challenge, Second Response, and Verification.

The main cryptographic tool that we use is the correlated GGM forest from Section 5.4.1. In this section, we base the construction on a random oracle $\mathcal{H}_2$ with output size $\lambda$. Following the state of the art, we use a $\mathsf{salt}$ of size $2\lambda$ bits and also do full domain separation, providing a tree number and a node position to derive the left child of each node using $\mathcal{H}_2$. In particular, the left child of a node in tree $j$ (where $j = \mathsf{pre}$ or $j \in [\tau]$) at position $\mathsf{pos} = (\mathsf{Level}, \mathsf{index})$ is derived as

$$\mathcal{H}_2\left(\mathsf{salt} \parallel j \parallel \mathsf{pos} \parallel \text{node value}\right).$$

The right child is obtained by XORing the left child and the parent. At the level of the leaf, we also use a random oracle (with proper domain separation) to derive additional shares for $\vec{y}$ and $z$ as well as a commitment value. To prevent the attack from [KLS25], we use explicit commitments to the tree of size $2\lambda$ bits. However, we propose a new MinMax technique to commit to the cGGM forest in Section 5.7.1 to reduce the signature size compared to the basic version. With our choice of security parameters, we need to generate $5\lambda + 1$ additional bits for each leaf. The commitment algorithm is shown in Figure 5.3. Figures 5.4 and 5.5 present the first challenge and response, while Figures 5.6 and 5.7 detail the second challenge and response. The verification algorithm is given in Figure 5.8.

## 5.6 Signature Scheme

We can now proceed to transform the identification protocol from Section 5.5 into a signature scheme by using the Fiat-Shamir transform [FS87]. As is Chapter 4, the public

**Standard commitment**

1. Sample $R_{\vec{x}} \leftarrow_{\$} (\mathbb{F}_2)^{n-2}$ and a random $\mathsf{salt} \leftarrow_{\$} \{0,1\}^{2\lambda}$ for the cGGM tree derivation
2. Compute pre-tree: $\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$
3. Fold pre-tree: $\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}\right)_{j\in[\tau]} \leftarrow \mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right)$
4. For $j \in [\tau]$:
   - Get Tree $T_j$: $\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}, 2^D\right)$
   - Expand $T_j$ and get $\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D-1]\!]}\right)$, $\left(z_j{}^{[\![0]\!]}, \ldots, z_j{}^{[\![2^D-1]\!]}\right)$ and $\left(\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right)$
   - Compute $X_j \leftarrow \bigoplus_{i=0}^{2^D-1} \mathrm{com}_j{}^{[\![i]\!]}$
   - Fold Tree $T_j$:
     - $\left(\vec{x}_j^{[\![0]\!]_i}, \vec{x}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right)$
     - $\left(\vec{y}_j^{[\![0]\!]_i}, \vec{y}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D-1]\!]}\right)$
     - $\left(z_j{}^{[\![0]\!]_i}, z_j{}^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(z_j{}^{[\![0]\!]}, \ldots, z_j{}^{[\![2^D-1]\!]}\right)$
     - $\left(\mathrm{com}_j{}^{[\![0]\!]_i}, \mathrm{com}_j{}^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right)$
   - Let $z \leftarrow z_0{}^{[\![0]\!]_0} \oplus z_0{}^{[\![1]\!]_0}$
   - Compute offsets:
     - $\delta_{\vec{y}_j} \leftarrow \vec{y} \oplus \vec{y}_j^{[\![0]\!]_0} \oplus \vec{y}_j^{[\![1]\!]_0}$
     - $\delta_{z_j} \leftarrow z \oplus z_j{}^{[\![0]\!]_0} \oplus z_j{}^{[\![1]\!]_0}$
5. Compute $h_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, \left(\mathrm{com}_j{}^{[\![0]\!]_i}, \mathrm{com}_j{}^{[\![1]\!]_i}\right)_{(i,j)\in[D]\times[\tau]}, (X_j)_{j\in[\tau]}\right)$
6. Send $h_{\mathrm{com}}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}, (X_j)_{j\in[\tau]}$ and $\mathsf{salt}$ to the verifier

**Figure 5.3:** Commitment algorithm of the VOLE identification protocol for NSBC.

**First challenge**

1. Sample $\vec{\mu} \leftarrow_{\$} (\mathbb{F}_{2^k})^{n-2}$
2. For $m \in [\tau \cdot D]$: Sample $\alpha_m \leftarrow_{\$} \mathbb{F}_{2^k}$
3. Send $\vec{\mu}, (\alpha_i)_{i\in[\tau\cdot D]}$ to the prover

**Figure 5.4:** First Challenge algorithm of the VOLE identification protocol for NSBC.

key is given by the two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, while the secret key consists of $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$ with $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ for $\vec{x}' \coloneqq (\vec{x}, 1, 0)$ and $\vec{y}' \coloneqq (\vec{y}, 0, 1)$.

### 5.6.1 Removing the Interactivity of the Protocol

In the signature scheme, all algorithms of the prover and verifier from the identification scheme except the final verification are executed by the signer. Only the final verification

**First Response**

1. $\vec{A}_{F_x} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{x}_j^{[\![0]\!]_i} \alpha_i$
2. $\vec{A}_{F_y} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$
3. $A \leftarrow \left(\vec{u} \cdot \begin{pmatrix} \vec{A}_{F_x} \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{A}_{F_y} \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{A}_{F_y} \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{A}_{F_x} \\ 1 \\ 0 \end{pmatrix}\right)$
4. $B \leftarrow \left(\vec{v} \cdot \begin{pmatrix} \vec{A}_{F_y} \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{u} \cdot \begin{pmatrix} \vec{x} \\ 1 \\ 0 \end{pmatrix}\right) + \left(\vec{u} \cdot \begin{pmatrix} \vec{A}_{F_x} \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{y} \\ 0 \\ 1 \end{pmatrix}\right)$
   $- \left(\vec{u} \cdot \begin{pmatrix} \vec{A}_{F_y} \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{x} \\ 1 \\ 0 \end{pmatrix}\right) - \left(\vec{v} \cdot \begin{pmatrix} \vec{A}_{F_x} \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{u} \cdot \begin{pmatrix} \vec{y} \\ 0 \\ 1 \end{pmatrix}\right)$
5. Compute $b \leftarrow z + \vec{y} \cdot \vec{\mu}$
6. For $j \in [\tau]$:
   - Compute $\vec{A}_{F_y}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$ and $\vec{A}_{F_z}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} z_j{}^{[\![0]\!]_i} \alpha_i$
   - Compute $a^{(j)} \leftarrow \vec{A}_{F_z}^{(j)} + \vec{A}_{F_y}^{(j)} \cdot \vec{\mu}$
7. Send $A, B, \left(a^{(j)}\right)_{j\in[\tau]}$ and $b$ to the verifier

**Figure 5.5:** First Response algorithm of the VOLE identification protocol for NSBC.

**Second Challenge**

1. Sample $\left(i_j^*\right)_{j\in[\tau]} \leftarrow_{\$} [2^D]^\tau$ and send it to the prover

**Figure 5.6:** Second Challenge algorithm of the VOLE identification protocol for NSBC.

**Second Response**

1. Compute the pre-challenge $i_p^* \leftarrow \sum_{j=0}^{\tau-1} \lfloor i_j^*/2^{D-1} \rfloor 2^j$
2. Compute the pre-tree PPRF key $K_{i_p^*}$
3. Compute the (truncated) PPRF key $K_{i_0^*,\ldots,i_{\tau-1}^*}$ of the second layer trees
4. Send $\left(K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}\right)$ to the verifier

**Figure 5.7:** Second Response algorithm of the VOLE identification protocol for NSBC.

is performed by the verifier.

Using the five-round Fiat-Shamir transform, we simply replace the first and second challenge by using hash values based on two random oracles $\mathcal{H}_0, \mathcal{H}_1$ with output size $2\lambda$ bits each. These values are expanded using pseudorandom generators $\mathrm{PRG}_0$ and $\mathrm{PRG}_1$ to obtain values in the respective challenge spaces. Given a message $\mathsf{msg} \in \{0,1\}^*$, we derive

**Verification**

1. Recompute, expand and fold all trees (with one missing leaf each) using $\left(K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}\right)$ and $\mathsf{salt}$
2. For $(i,j) \in [D] \times [\tau]$, let $b_{i,j}$ denote the known position of the respective hypercube binary sharing
3. For $(i,j) \in [D] \times [\tau]$: Compute $\mathrm{com}_j{}^{[\![1-b_{i,j}]\!]_i} \leftarrow X_j \oplus \mathrm{com}_j{}^{[\![b_{i,j}]\!]_i}$
4. $\delta_{z_0} \leftarrow 0$
5. For $j \in [\tau]$:
   - Compute $\Delta_j^* \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} b_{i,j}\alpha_i$
   - Compute $\vec{F}_y^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i \left(\vec{y}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{\vec{y}_j}\right)$
   - Compute $Z^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i \left(z_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{z_j}\right)$
   - $a^{(j)'} \leftarrow Z^{(j)}\left(\Delta_j^*\right) \oplus \vec{F}_y^{(j)}(\Delta_j^*) \cdot \vec{\mu} \oplus b\Delta_j^*$
6. $\Delta^* \leftarrow \sum_{j=0}^{\tau-1} \Delta_j^*$
7. $\vec{F}_x(\Delta^*) \leftarrow \sum_{m=0}^{\tau\cdot D-1} \vec{x}_m^{[\![b_m]\!]}\alpha_m$
8. $\vec{F}_y(\Delta^*) \leftarrow \sum_{j=0}^{\tau-1} \vec{F}_y^{(j)}(\Delta_j^*)$
9. Compute $\Phi(\Delta^*)$ as
$$\left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)$$
10. $A' \leftarrow \Phi(\Delta^*) - B\Delta^*$
11. Compute $h'_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, \left(\mathrm{com}_j{}^{[\![0]\!]_i}, \mathrm{com}_j{}^{[\![1]\!]_i}\right)_{(i,j)\in[D]\times[\tau]}, (X_j)_{j\in[\tau]}\right)$
12. If $h'_{\mathrm{com}} = h_{\mathrm{com}}$, $A = A'$ and $(a^{(j)})_{j\in[\tau]} = (a^{(j)'})_{j\in[\tau]}$ output `ACCEPT`, otherwise output `REJECT`

**Figure 5.8:** Verification algorithm of the VOLE identification protocol for NSBC.

the first hash as

$$h_0 \leftarrow \mathcal{H}_0\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, (X_j)_{j\in[\tau]}\right)$$

and expand it into $\vec{\mu}$ and $(\alpha_i)_{i\in[\tau\cdot D]}$ using $\mathrm{PRG}_0$. We also assume that $\mathcal{H}_0$ takes additional context, for example the public key $\mathsf{pk}$, as input. By doing that, any forgery attempt is tied to a specific user. To simplify the notation, we do not specifically write this in every input.

Note that neither $h_0$ nor the derived $\vec{\mu}$ and $(\alpha_i)_{i\in[\tau\cdot D]}$ need to be sent as part of the signature since they can be recomputed from values that are later sent as part of the signature anyway by using $\mathcal{H}_0$ and $\mathrm{PRG}_0$.

To derive the second challenge, we continue hashing, including now $A$, $B$, $\left(a^{(j)}\right)_{j\in[\tau]}$ and $b$ into a hash $h_1$ from which we obtain the second challenge $(i_j^*)_{j\in[\tau]}$ using $\mathrm{PRG}_1$. This also permits additional reduction of the signature size. Indeed, since $A$ and $\left(a^{(j)}\right)_{j\in[\tau]}$ are recomputed during the verification phase, they do not need to be sent. Instead, sending

(and verifying) the hash $h_1$ is sufficient. We provide pseudocode of the algorithms for the VOLEitH SBC signature scheme below. The signing algorithm is given in Figure 5.9, while the verification algorithm is given in Figure 5.10.

### 5.6.2 Signature Size

As mentioned in Section 4.5, we need the bitsize of the elements of $\mathbb{F}_{q^k}$ to be at least $2\lambda$ to avoid generic attacks based on collision finding. Therefore, since $n \approx \frac{k}{2}$, the bitsize of the elements of $(\mathbb{F}_q)^{n-2}$ is $\lambda$. The signature consists of the following elements:

- One hash value corresponding to the global commitment of size $2\lambda$.
- One salt of size $2\lambda$.
- The punctured key of the correlated GGM forest of size $\lambda\tau\log(N)$.
- The coefficient $B \in \mathbb{F}_{q^k}$.
- The coefficient $b = z_0 + \vec{y}_0 \cdot \vec{\mu} \in \mathbb{F}_{q^k}$.
- The offset values $\delta_{\vec{y}_0}, \ldots, \delta_{\vec{y}_{\tau-1}} \in (\mathbb{F}_q)^{n-2}$ and $\delta_{z_1}, \ldots, \delta_{z_{\tau-1}} \in \mathbb{F}_{q^k}$.
- **Explicit tree commitment**[1]: the XORs $(X_j)_{j\in[\tau]}$ of the commitment values of each tree of size $\tau \cdot 2\lambda$.

The total communication cost in bits of the protocol is therefore

$$\begin{aligned} \mathsf{size} &\geq \lambda\tau\log(N) + \tau\lambda + 2\tau\lambda + (\tau-1)2\lambda + 2\lambda + 2\lambda + 2\lambda + 2\lambda \\ &= \lambda\tau\log(N) + 5\tau\lambda + 6\lambda \\ &\approx \lambda^2 + 5\tau\lambda + 7\lambda. \end{aligned}$$

The final approximate size depends on the exact value of $\tau\log(N)$. We need $\tau\log(N) > \lambda$ and take $\tau\log(N) \approx \lambda + 1$, as far as factorization of numbers close to $\lambda + 1$ permits.
From Theorem 5.5, we know that the total soundness error of the identification protocol we use is bounded by

$$\tau^2\, 2^{-k} + 2^{1-\tau\cdot D}. \tag{5.5}$$

In general, for a signature scheme based on the Fiat-Shamir transformation of a 5 round protocol, the cost of forgery is given by the Kales and Zaverucha formula, based on their attack from [KZ20]:

$$\text{cost} = \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} \mathrm{p}^i (1-\mathrm{p})^{\tau-i}} + N^{\tau_2} \right\},$$

where p denotes the false positive probability of the underlying identification scheme.
In our case, the false positive probability, in the sense of Kales and Zaverucha, is $\mathrm{p} = \tau\, 2^{-k}$ from Lemma 5.4. Since this is extremely small compared to the desired security level, the Kales and Zaverucha attack does not apply in our case. In fact, our proof of Theorem 5.5 uses a union bound argument to prevent any false positive from happening, and the same proof directly shows the security level of our signature scheme in the random oracle model.
Consider the security level of $\lambda = 128$ bits and the parameters $q = 2, k = 257$ and $n = 130$. For these parameters, the values for $D$ and $\tau$, including the respective signature sizes, are

[1] In Section 5.7.1, we introduce a technique to remove this explicit commitment to shorten the signature.

**Inputs:** Secret key $\mathsf{sk} = (\vec{x}, \vec{y})$, public key $\mathsf{pk} = (\vec{u}, \vec{v})$ and a message $\mathsf{msg} \in \{0,1\}^*$

1. Sample $R_{\vec{x}} \leftarrow_\$ (\mathbb{F}_2)^{n-2}$ and $\mathsf{salt} \leftarrow_\$ \{0,1\}^{2\lambda}$ for the cGGM tree derivation
2. Compute pre-tree: $\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$
3. Fold pre-tree: $\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}\right)_{j\in[\tau]} \leftarrow \mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right)$
4. For $j \in [\tau]$:
   - Get Tree $T_j$: $\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}, 2^D\right)$
   - Expand $T_j$ and get $\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D-1]\!]}\right)$, $\left(z_j{}^{[\![0]\!]}, \ldots, z_j{}^{[\![2^D-1]\!]}\right)$ and $\left(\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right)$
   - Compute $X_j \leftarrow \bigoplus_{i=0}^{2^D-1} \mathrm{com}_j{}^{[\![i]\!]}$
   - Fold Tree $T_j$:
     - $\left(\vec{x}_j^{[\![0]\!]_i}, \vec{x}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right)$
     - $\left(\vec{y}_j^{[\![0]\!]_i}, \vec{y}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D-1]\!]}\right)$
     - $\left(z_j{}^{[\![0]\!]_i}, z_j{}^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(z_j{}^{[\![0]\!]}, \ldots, z_j{}^{[\![2^D-1]\!]}\right)$
     - $\left(\mathrm{com}_j{}^{[\![0]\!]_i}, \mathrm{com}_j{}^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right)$
   - Let $z \leftarrow z_0{}^{[\![0]\!]_0} \oplus z_0{}^{[\![1]\!]_0}$
   - Compute offsets $\delta_{\vec{y}_j} \leftarrow \vec{y} \oplus \vec{y}_j^{[\![0]\!]_0} \oplus \vec{y}_j^{[\![1]\!]_0}$ and $\delta_{z_j} \leftarrow z \oplus z_j{}^{[\![0]\!]_0} \oplus z_j{}^{[\![1]\!]_0}$
5. Compute $h_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, \left(\mathrm{com}_j{}^{[\![0]\!]_i}, \mathrm{com}_j{}^{[\![1]\!]_i}\right)_{(i,j)\in[D]\times[\tau]}, (X_j)_{j\in[\tau]}\right)$
6. $h_0 \leftarrow \mathcal{H}_0\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, (X_j)_{j\in[\tau]}\right)$
7. $(\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]}) \leftarrow \mathrm{PRG}_0(h_0)$
8. Compute $\vec{A}_{F_x} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{x}_j^{[\![0]\!]_i} \alpha_i$
9. Compute $\vec{A}_{F_y} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$
10. Compute the coefficients $A$ and $B$ of $\Phi$ as in the First Response algorithm
11. Compute $b \leftarrow z + \vec{y} \cdot \vec{\mu}$
12. For $j \in [\tau]$:
    - Compute $\vec{A}_{F_y}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$ and $\vec{A}_{F_z}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} z_j{}^{[\![0]\!]_i} \alpha_i$
    - Compute $a^{(j)} \leftarrow \vec{A}_{F_z}^{(j)} + \vec{A}_{F_y}^{(j)} \cdot \vec{\mu}$
13. $h_1 \leftarrow \mathcal{H}_1\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, h_0, A, B, \left(a^{(j)}\right)_{j\in[\tau]}, b\right)$
14. $(i_j^*)_{j\in[\tau]} \leftarrow \mathrm{PRG}_1(h_1)$
15. Compute the pre-challenge $i_p^* = \sum_{j=0}^{\tau-1} \lfloor i_j^* / 2^{D-1} \rfloor \cdot 2^j$
16. Compute the pre-tree PPRF key $K_{i_p^*}$ and the PPRF key $K_{i_0^*,\ldots,i_{\tau-1}^*}$
17. Output $\sigma \leftarrow \left(h_1, \mathsf{salt}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}, (X_j)_{j\in[\tau]}\right)$

**Figure 5.9:** Signing algorithm of the $\mathrm{SBC}_{\mathrm{VOLE}}$ signature scheme.

**Inputs:** Public key $\mathsf{pk} = (\vec{u}, \vec{v})$, a message $\mathsf{msg} \in \{0,1\}^*$ and a signature $\sigma = \left(h_1, \mathsf{salt}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}, (X_j)_{j\in[\tau]}\right)$

1. $(i_j^*)_{j\in[\tau]} \leftarrow \mathrm{PRG}_1(h_1)$
2. Recompute, expand and fold all trees (with one missing leaf each) using $K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}$ and $\mathsf{salt}$
3. For $(i,j) \in [D]\times[\tau]$, let $b_{i,j}$ denote the known position of the respective hypercube binary sharings
4. For $(i,j) \in [D]\times[\tau]$: Compute $\mathrm{com}_j{}^{[\![1-b_{i,j}]\!]_i} \leftarrow X_j \oplus \mathrm{com}_j{}^{[\![b_{i,j}]\!]_i}$
5. Compute $h'_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, \left(\mathrm{com}_j{}^{[\![0]\!]_i}, \mathrm{com}_j{}^{[\![1]\!]_i}\right)_{(i,j)\in[D]\times[\tau]}, (X_j)_{j\in[\tau]}\right)$
6. $\delta_{z_0} \leftarrow 0$
7. $h'_0 \leftarrow \mathcal{H}_0\left(\mathsf{msg}, h'_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, (X_j)_{j\in[\tau]}\right)$
8. $(\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]}) \leftarrow \mathrm{PRG}_0(h'_0)$
9. For $j \in [\tau]$:
   - Compute $\Delta_j^* \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} b_{i,j}\alpha_i$
   - Compute $\vec{F}_y^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i\left(\vec{y}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{\vec{y}_j}\right)$
   - Compute $Z^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i\left(z_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{z_j}\right)$
   - $a^{(j)'} \leftarrow Z^{(j)}\left(\Delta_j^*\right) \oplus \vec{F}_y^{(j)}(\Delta_j^*)\cdot\vec{\mu} \oplus b\Delta_j^*$
10. $\Delta^* \leftarrow \sum_{j=0}^{\tau-1}\Delta_j^*$
11. $\vec{F}_x(\Delta^*) \leftarrow \sum_{m=0}^{\tau\cdot D-1} \vec{x}_m^{[\![b_m]\!]}\alpha_m$
12. $\vec{F}_y(\Delta^*) \leftarrow \sum_{j=0}^{\tau-1}\vec{F}_y^{(j)}(\Delta_j^*)$
13. Compute $\Phi(\Delta^*)$ as
$$\left(\vec{u}\cdot\begin{pmatrix}\vec{F}_x(\Delta^*)\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_y(\Delta^*)\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{F}_y(\Delta^*)\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_x(\Delta^*)\\1\\0\end{pmatrix}\right)$$
14. $A' \leftarrow \Phi(\Delta^*) - B\Delta^*$
15. $h'_1 \leftarrow \mathcal{H}_1\left(\mathsf{msg}, h'_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, h'_0, A', B, \left(a^{(j)'}\right)_{j\in[\tau]}, b\right)$
16. If $h'_1 = h_1$ output `ACCEPT`, otherwise output `REJECT`

**Figure 5.10:** Verification algorithm of the $\mathrm{SBC}_{\mathrm{VOLE}}$ signature scheme.

displayed in Table 5.1. We note that, unlike in the standard MPCitH SBC scheme from Chapter 4, using $(D,\tau) = (16,8)$ and $(D,\tau) = (8,16)$ does not yield a signature with 128 bits security level, since Equation (5.5) yields a security parameter smaller than 127 bits for these cases. This one bit gap comes from the fact that the evaluation point of the quadratic polynomial is selected from a smaller set of values. For this reason, the potential existence of two roots boosts the probability of a cheating prover by a factor of two.

**Public key size.** The public key of the SBC scheme consists of a seed $\mathsf{seed}_{\mathsf{pk}}$ of size $\lambda$ and a finite field element in $\mathbb{F}_{q^k}$, using the key generation described in Section 3.3.2. From the seed, we can derive the values of $\vec{u}$ and $v_0,\ldots,v_{n-2}$ using a PRG, while the element $v_{n-1} \in \mathbb{F}_{q^k}$ is sent separately. Therefore, the public key size is $3\lambda$ bits. For $\lambda = 128$, this

| $D$ | $\tau$ | $\lambda^2 + 5\tau\lambda + 7\lambda$ | \|sgn\| |
|---|---|---|---|
| 9 | 15 | 3 360 B | 3 458 B |
| 10 | 13 | 3 200 B | 3 218 B |
| 11 | 12 | 3 120 B | 3 170 B |
| 12 | 11 | 3 040 B | 3 090 B |
| 13 | 10 | 2 960 B | 2 978 B |
| 15 | 9 | 2 880 B | 2 978 B |

**Table 5.1:** $\text{SBC}_{\text{VOLE}}$ signature size for $N = 2^D$ parties and $\tau$ rounds with parameters $q = 2, k = 257$ and $n = 130$ for the security parameter $\lambda = 128$ bits.

yields a public key size of 48 bytes.

### 5.6.3 Proof of Existential Unforgeability

In this section, we provide a proof for existential unforgeability against chosen message attacks (EUF-CMA) of the VOLEitH SBC signature scheme from Section 5.6. The adversary, given a public key $\text{pk} = (\vec{u}, \vec{v})$, is allowed to query a signing oracle to sign messages of its choice. The goal of the adversary is to generate a valid message-signature pair $(\mathsf{msg}, \sigma)$, with the restriction that $\mathsf{msg}$ was not queried before. In the security analysis, we also use the notion of existential unforgeability against key-only attacks (EUF-KO), where the adversary only has access to the public key. The formal definitions of these standard cryptographic notions are given in Section 2.2.6.
The security analysis for our scheme is based on the following assumptions:

- **Hardness of SBC.** We assume that solving the given SBC instance is hard. In particular, any adversary $\mathcal{A}$, given as input a random SBC instance $(\vec{u}, \vec{v})$ and running in time $t$, outputs a valid solution $(\vec{x}, \vec{y}) \in \text{SBC}[\vec{u}, \vec{v}]$ with probability at most $\varepsilon_{\text{SBC}}$. Here, $\varepsilon_{\text{SBC}}$ and $t$ implicitly depend on the security parameter $\lambda$.
- **The Random Oracle Model.** The hash functions used in the protocol are assumed to be random oracles.
- **Secure pseudorandomness.** We assume that the PRGs used in the protocol are $(t, \varepsilon_{\text{PRG}})$-secure.

**Theorem 5.6.** Consider the NSBC signature scheme described in Section 5.6.1. Assume that the PRGs used are $(t, \varepsilon_{\text{PRG}})$-secure and that any adversary running in time $t$ has an advantage of at most $\varepsilon_{\text{SBC}}$ against the SBC problem. Assume that $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ are random oracles. Let $\mathcal{A}$ be an adversary against the EUF-CMA security of the signature scheme, which has a running time $t$ and makes $q_i$ queries to $\mathcal{H}_i$ for $i \in \{0, 1, 2, 3\}$ and $q_s$ queries to the signing oracle. Then, the advantage of $\mathcal{A}$ in the EUF-CMA game is upper bounded by

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq \frac{4Q^2}{2^{2\lambda}} + \varepsilon_{\text{PRG}} + \varepsilon_{\text{SBC}} + \varepsilon(t),$$

where $Q = q_0 + q_1 + q_2 + q_3 + N_{\mathcal{H}} \cdot q_s$ is the total number of random oracle queries. Here, $N_{\mathcal{H}} = 3\tau \cdot 2^D + 2^{\tau+1} - 2\tau$ is the number of random oracle queries per signature computation and $\varepsilon := \tau^2\, 2^{-k} + 2^{1-\tau \cdot D}$ is the soundness error of the SBC protocol from Theorem 5.5.

*Proof.* This proof is an adaptation of the EUF-CMA proof of the Banquet signature scheme [BDK+21] and follows the standard hybrid argument proof technique used in MPCitH signature schemes like [BFR24, FJR22, Bui24].

First, we consider the EUF-KO game and upper-bound the advantage of the adversary in this game. For simplicity, we assume that the public key and the output from the PRGs are truly random values, instead of being pseudorandomly generated. Later in the proof, we account for the pseudorandomness in the real signature scheme by relaxing this assumption.

**EUF-KO security.**($\mathsf{Adv}^{\mathsf{EUF\text{-}KO}}_{\mathcal{A}}$) Assume an adversary $\mathcal{A}$ with running time $t$ wins the EUF-KO security game with probability $\varepsilon_{\mathsf{EUF\text{-}KO}}$. We use $\mathcal{A}$ to construct a reduction $\mathcal{R}$ that solves the SBC problem in time $t$ with probability $\varepsilon_{\mathrm{SBC}}$. This reduction $\mathcal{R}$, given the public key $(\vec{u}, \vec{v})$, interacts with $\mathcal{A}$ by simulating the EUF-KO game. For any hash query by $\mathcal{A}$, the reduction samples a random value and returns it as a response. For the whole interaction, $\mathcal{R}$ keeps a list $\mathcal{Q}$ of all random oracle queries $q$ and a list $L$ with the corresponding answers $h$. Given a fresh query $q \notin \mathcal{Q}$, $\mathcal{R}$ samples a random value $h$. If $h \in L$, $\mathcal{R}$ aborts, otherwise, $h$ is added to $L$. Moreover, for each query to $\mathcal{H}_1$ of the form

$$h_1 \leftarrow \mathcal{H}_1\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, h_0, A, B, \left(a^{(j)}\right)_{j\in[\tau]}, b\right),$$

$\mathcal{R}$ also adds $h_0$ and $h_{\mathrm{com}}$ to $L$. It follows that

$$\Pr[\mathcal{R} \text{ aborts}] \leq q_{\mathcal{H}} \cdot \frac{\max|L|}{2^{2\lambda}} = q_{\mathcal{H}} \cdot \frac{(q_0 + 3q_1 + q_2 + q_3)}{2^{2\lambda}} \leq \frac{3q_{\mathcal{H}}^2}{2^{2\lambda}}, \tag{5.6}$$

where $q_{\mathcal{H}} = q_0 + q_1 + q_2 + q_3$.

During the interaction, $\mathcal{R}$ also keeps track of a list $\mathcal{L}_{\mathsf{cand}}$ of the candidate witnesses that can be derived from the queries of $\mathcal{A}$ to $\mathcal{H}_2$ and $\mathcal{H}_3$.

In particular, from the internal nodes of the cGGM tree derivation of the form

$$\mathcal{H}_2\left(\mathsf{salt} \parallel j \parallel \mathsf{pos} \parallel \text{node value}\right),$$

we can derive the leaf values $\vec{x}_j^{[\![i]\!]}$. From these shares, $\mathcal{R}$ can also derive $\vec{y}_j^{[\![i]\!]}$ and add $\vec{x} = \sum_{i=0}^{N-1} \vec{x}_j^{[\![i]\!]}$ to $\mathcal{L}_{\mathsf{cand}}$. After receiving a valid $(\mathsf{msg}, \sigma)$-pair from $\mathcal{A}$, the reduction checks if there is a candidate witness in $\mathcal{L}_{\mathsf{cand}}$ such that $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$, where $\vec{y} = \sum_{i=0}^{N-1} \vec{y}_j^{[\![i]\!]} + \delta_{\vec{y}_j}$. If this is the case, then $\mathcal{R}$ returns $(\vec{x}, \vec{y})$, otherwise $\mathcal{R}$ returns $\perp$.

By the union bound, we have that

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{R} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{R} \text{ returns } \perp] \\
&\quad + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{R} \text{ returns } (\vec{x}, \vec{y})] \\
&\leq \Pr[\mathcal{R} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \mid \mathcal{R} \text{ returns } \perp] + \Pr[\mathcal{R} \text{ returns } (\vec{x}, \vec{y})].
\end{aligned}$$

By (5.6) and the assumed hardness of SBC, this yields

$$\mathsf{Adv}^{\mathsf{EUF\text{-}KO}}_{\mathcal{A}} \leq \frac{3q_{\mathcal{H}}^2}{2^{2\lambda}} + \Pr[\mathcal{A} \text{ wins} \mid \mathcal{R} \text{ returns } \perp] + \varepsilon_{\mathrm{SBC}}.$$

Next, we analyze the conditional probability $\Pr[\mathcal{A} \text{ wins} \mid \mathcal{R} \text{ returns } \perp]$. If $\mathcal{R}$ returns $\perp$, this means that there is no solution to the SBC problem that is obtained from the internal

queries to $\mathcal{H}_2$ in the cGGM construction by the adversary. If no such queries exist, the adversary can only output a valid signature if all consistency checks verify and the evaluation point $\Delta^*$ is a root of the degree 2 polynomial $\Phi(\Delta) - B\Delta - A$. As seen in the proof of Theorem 5.5, this probability is at most $\varepsilon = \tau^2\, 2^{-k} + 2^{1-\tau \cdot D}$. In total, we can therefore bound the advantage of $\mathcal{A}$ in the existential unforgeability game against key-only attacks by

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}KO}} \leq \frac{3q_{\mathcal{H}}^2}{2^{2\lambda}} + \varepsilon(t) + \varepsilon_{\mathrm{SBC}}.$$

We use a sequence of modified games, based on the reduction $\mathcal{R}$ above, to upper-bound the advantage of the adversary in the $\mathsf{EUF\text{-}CMA}$ game and use $\varepsilon_{\mathsf{Game}_i}$ to denote the probability that $\mathcal{R}$ outputs a solution $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$ in game $i$.

**Game 1.** In this game, the reduction also answers signature queries from $\mathcal{A}$ by using the signing oracle $\mathcal{O}(\mathsf{sk}, \mathsf{pk})$. For each signing query, the reduction uses the same method as before to answer the hash queries, while the signature is computed as in the signing algorithm described in Figure 5.9. The total number of random oracle queries during the interaction is therefore $Q = q_0 + q_1 + q_2 + q_3 + N_{\mathcal{H}} \cdot q_s$, where the total number of hash queries per signing query is $N_{\mathcal{H}} = 3\tau \cdot 2^D + 2^{\tau+1} - 2\tau$ (one query to $\mathcal{H}_0$ and $\mathcal{H}_1$ each, $2^{\tau+1} - 2$ queries to $\mathcal{H}_2$ in the pre-tree computation, $2^{D+1} - 2$ queries to $\mathcal{H}_2$ for each of the $\tau$ second layer trees and $\tau \cdot 2^D$ for the commitment values). Assume that an adversary $\mathcal{A}$ with running time $t$ wins the $\mathsf{EUF\text{-}CMA}$ security game with probability $\varepsilon_{\mathsf{EUF\text{-}CMA}}$. By doing the same union bound argument as in the key-only game, we obtain the upper bound

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq \frac{3Q^2}{2^{2\lambda}} + \varepsilon(t) + \varepsilon_{\mathsf{Game}_1}. \tag{5.7}$$

**Game 2.** We modify the first game. Upon answering signature queries, the signing oracle $\mathcal{O}(\mathsf{sk}, \mathsf{pk})$ is replaced by a simulator $\mathcal{S}(\mathsf{pk})$, which does not have the secret key $\mathsf{sk}$ as input. This simulator behaves similar to the one used for the MPCitH protocol from Theorem 4.1, with some adaptations for the new VOLEitH based protocol. In particular, during the interaction, $\mathcal{R}$ saves a list $L_{\mathsf{salt}}$ of all $\mathsf{salt}$ values used in random oracle queries of the form

$$\begin{aligned} h_1 &\leftarrow \mathcal{H}_1\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, h_0, A, B, \left(a^{(j)}\right)_{j\in[\tau]}, b\right) \text{ and} \\ h_0 &\leftarrow \mathcal{H}_0\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, (X_j)_{j\in[\tau]}\right). \end{aligned}$$

First, the simulator picks random values for $\mathsf{salt}$ and $R_{\vec{x}}$ and aborts if the salt was used in a previous cGGM tree construction for a signature. This way, if $\mathcal{S}$ does not abort, we can view the shares of $\vec{x}$ generated by the tree as random values, since they come from a new cGGM forest. Since the simulator $\mathcal{S}$ knows the challenges in advance, it can generate a signature with perfect distribution without knowing $\mathsf{sk}$.

In particular, $\mathcal{S}$ picks random values $h_0, h_1$ for the random oracle outputs of $\mathcal{H}_0$ and $\mathcal{H}_1$ in advance, instead of computing them during the signing process. In the signing algorithm from Figure 5.9, the output of $\mathcal{H}_0$ is set to $h_0$ and the output of $\mathcal{H}_1$ is set to $h_1$. Using the PRGs, these values are expanded to obtain the challenges $(\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]})$ and $(i_j^*)_{j\in[\tau]}$. The simulator then proceeds to derive the shares of $\vec{x}$ and $\vec{y}$ as in the original signing algorithm, with the difference that $\vec{x}$ and the offset values $\left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}$ are

picked at random. Therefore, $\vec{x}$ and the derived $\vec{y} = \sum_{i=0}^{N-1} \vec{y}_j^{[\![i]\!]} + \delta_{\vec{y}_j}$ solve the SBC problem only with negligible probability. The simulator can then use the Verification algorithm to compute the expected values of $A, B, b$ and $\left(\delta_{z_j}\right)_{j\in[\tau]^*}$. Using this simulator, the shares revealed in the signature are uniformly distributed under the condition that $\Phi(\Delta^*) = A + B\Delta^*$, i.e., they have the same distribution as the shares obtained from an honest signature execution. If the algorithms do not abort, the obtained signatures from $\mathcal{S}(\mathsf{pk})$ and $\mathcal{O}(\mathsf{sk}, \mathsf{pk})$ are therefore identically distributed. Moreover, the probability that the algorithms abort because a hash collision (in list $L$) was found is equal for both cases. The simulator might also abort if there is a collision for $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$, which occurs with probability at most $\frac{(q_0+q_1+q_2+q_3+q_s)}{2^{2\lambda}}$ per signing query. Therefore, by comparing Game 1 and Game 2, we see that

$$|\varepsilon_{\mathsf{Game}_2} - \varepsilon_{\mathsf{Game}_1}| \leq \frac{q_s(q_0+q_1+q_2+q_3+q_s)}{2^{2\lambda}} \leq \frac{Q^2}{2^{2\lambda}}. \tag{5.8}$$

From (5.7) and (5.8), we obtain

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq \varepsilon_{\mathsf{Game}_2} + \frac{4Q^2}{2^{2\lambda}} + \varepsilon(t). \tag{5.9}$$

**Game 3.** In this game, we replace the idealized true randomness generator from the previous games by the real PRGs used in the signature scheme. By assumption, each PRG is $(t, \varepsilon_{\mathrm{PRG}})$-indistinguishable from true randomness. Also, we pseudorandomly generate the public key $(\vec{u}, \vec{v})$ based on a seed $\mathsf{seed}_{\mathsf{pk}}$, as described in Section 5.6.2. Therefore, this game represents the interaction of $\mathcal{R}$ with the real SBC signature scheme from Figure 5.9. The probability that $\mathcal{R}$ recovers the secret key $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$ in Game 3 is therefore $\varepsilon_{\mathsf{Game}_3} = \varepsilon_{\mathrm{SBC}}$. Since we only changed the true randomness from Game 2 into pseudorandomness, we have

$$|\varepsilon_{\mathsf{Game}_3} - \varepsilon_{\mathsf{Game}_2}| \leq \varepsilon_{\mathrm{PRG}}. \tag{5.10}$$

Finally, by combining (5.9) and (5.10), we obtain the claimed upper bounds as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq \frac{4Q^2}{2^{2\lambda}} + \varepsilon_{\mathrm{PRG}} + \varepsilon_{\mathrm{SBC}} + \varepsilon(t).$$

□

## 5.7 Implementation and Performance

In this section, we provide the signature sizes and running times of the scheme based on our C implementation, which is an adaptation of the artifact from [HJ24a]. The total communication cost of the standard MPCitH SBC signature is $\lambda^2 + 16\tau\lambda + 4\lambda$ bits, while the communication cost of the VOLEitH version is $\lambda^2 + 5\tau\lambda + 7\lambda$ bits. As mentioned in Section 5.3.3, we use AES as the internal cGGM forest derivation function for $\lambda = 128$ in both implementations to improve the running time, which is a technique introduced in [BCC+24b].

We compare our VOLEitH based scheme with the standard MPCitH signature scheme

| | | standard MPCitH SBC [HJ24a] | | | | | $\text{SBC}_{\text{VOLE}}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D$ | $\tau$ | \|sgn\| | Sign | | Verify | | \|sgn\| | Sign | | Verify | |
| | | | time | cycles | time | cycles | | time | cycles | time | cycles |
| 8 | 16 | 5 452 B | 0.77 ms | 3.42 | 0.68 ms | 3.00 | — | — | — | — | — |
| 9 | 15 | 5 356 B | 0.90 ms | 3.97 | 0.81 ms | 3.55 | 3 458 B | 2.41 ms | 9.29 | 2.40 ms | 9.26 |
| 10 | 13 | 4 858 B | 1.29 ms | 4.54 | 1.21 ms | 4.14 | 3 218 B | 1.58 ms | 6.08 | 1.57 ms | 6.05 |
| 11 | 12 | 4 681 B | 1.60 ms | 6.00 | 1.51 ms | 5.59 | 3 170 B | 1.60 ms | 6.17 | 1.59 ms | 6.12 |
| 12 | 11 | 4 473 B | 2.25 ms | 8.61 | 2.15 ms | 8.17 | 3 090 B | 1.79 ms | 6.88 | 1.77 ms | 6.82 |
| 13 | 10 | 4 232 B | 3.47 ms | 13.5 | 3.34 ms | 12.9 | **2 978 B** | **2.21ms** | **8.53** | **2.19ms** | **8.43** |
| 15 | 9 | 4 103 B | 11.3 ms | 43.9 | 10.9 ms | 42.6 | 2 978 B | 5.13 ms | 19.8 | 5.0 ms | 19.3 |
| 16 | 8 | 3 782 B | 19.7 ms | 77.6 | 19.2 ms | 75.1 | — | — | — | — | — |

**Table 5.2:** Signature sizes and running times for $\lambda = 128$ using $N = 2^D$ parties and $\tau$ rounds with the parameters $q = 2$, $k = 257$ and $n = 130$ for standard MPCitH based SBC and $\text{SBC}_{\text{VOLE}}$ using an AMD EPYC 9374F processor running at 3.85 GHz. The CPU clock cycles are given in millions of cycles (Mc).

for SBC from Chapter 4 in Table 5.2. We see that the case $(D, \tau) = (13, 10)$ yields the smallest signature size of 2 978 bytes, which results in a decrease of the signature size of $\approx 21\%$ compared to the smallest signature size obtained by the standard MPCitH based SBC signature scheme from Chapter 4, i.e., for $(D, \tau) = (16, 8)$. The running time of the smallest $\text{SBC}_{\text{VOLE}}$ instance $(D, \tau) = (13, 10)$ is reduced by 90% compared to the smallest instance of the MPCitH version for $(D, \tau) = (16, 8)$. We also note that the $\text{SBC}_{\text{VOLE}}$ scheme for $(D, \tau) = (9, 15)$ and $(D, \tau) = (15, 9)$ is redundant since it is slower and larger compared to the $(13, 10)$ version.

**Remark 5.7** (Improving signature size with variable-sized trees)**.** The main restriction on finding good parameter choices is to find multiples of the desired number of trees, i.e. of the value $\tau$, slightly above 128. As seen in Table 5.2, this leads to very limited good choices. We can improve the accessible range of parameters by allowing the second level trees in the correlated forest to have variable sizes. For example, with $\tau = 9$, we could use four trees of size $2^{15}$ and five of size $2^{14}$. In this situation, the number of binary sharings would be 130, i.e., the same as in the $\tau = 10$ case. This approach can potentially reduce the signature sizes to the estimated value $\lambda^2 + 5\tau\lambda + 7\lambda$ given in Table 5.1.

### 5.7.1 Using MinMax Commitments to Improve the Signature Size

To reduce the communication cost of the signature scheme, we introduce an alternative way to commit to the cGGM forest. In the basic version from Section 5.6, we expand each leaf into a total of $5\lambda$ bits: $\lambda$ bits for shares of $\vec{y}$, $2\lambda$ bits for shares of $z$ and $2\lambda$ bits for the commitment values ${\text{com}_j}^{[\![i]\!]}$.
In this section, we introduce an implicit commitment based on the hash values ${\text{com}_j}^{[\![i]\!]}$, which are pseudorandom values derived using a random oracle. This reduces the communication cost of the signature scheme by $\tau \cdot 2\lambda$ bits, corresponding to the XOR values $(X_j)_{j \in [\tau]}$. However, this method has an abort probability, which leads to a small (amortized) increase of the expected running time of the signing algorithm. Note that this technique is generically applicable to other MPCitH or VOLEitH signature schemes that use GGM trees and is not tied to our SBC protocol.
For each subtree $T_j$, we compute the maximum value among the hash values ${\text{com}_j}^{[\![i]\!]}$ with

respect to some fixed ordering (e.g. lexicographical ordering). In particular, let

$$M_{i_j^{\max}} \leftarrow \max\left\{\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right\}$$

for $j$ in $[\tau]$. We denote the indices of these values by $i_j^{\max} \in [2^D]$ for $j \in [\tau]$. Since each commitment value is a pseudorandom value of $2\lambda$ bits, the maximum value in each subtree $T_j$ is unique for each $j \in [\tau]$ with overwhelming probability. Indeed, if there are two distinct positions in a subtree with the same value, these correspond to a collision of the hash function used to expand the leaves. This happens with negligible probability. If the maximum value of a given subtree is not unique, we abort and restart the signing process. In this new version, we replace the computation of the commitment hash with

$$h_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, (j, M_{i_j^{\max}}, i_j^{\max})_{j\in[\tau]}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}\right).$$

After computing the challenges based on this hash as before, the signer obtains the indices $(i_j^*)_{j\in[\tau]}$ using $\mathrm{PRG}_1$. If one of these challenge indices coincides with the index $i_j^{\max}$ of a maximum value $M_{i_j^{\max}}$ in the corresponding subtree $T_j$, the signer aborts and restarts the signing process. By the union bound, the probability of this event is at most $\tau \cdot \frac{1}{2^D}$. Otherwise, the signer proceeds as before and sends the signature

$$\sigma \leftarrow \left(h_1, \mathsf{salt}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}\right)$$

to the verifier. Note that, compared to the standard version, we do not need to send the values $(X_j)_{j\in[\tau]}$ in this version of the signature.
The verifier proceeds as before and also computes the maximum values among the hashes in each subtree:

$$M'_{i_j^{\max}} \leftarrow \max\left\{\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![i_j^*-1]\!]}, \mathrm{com}_j{}^{[\![i_j^*+1]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right\}.$$

Since $i_j^* \neq i_j^{\max}$ for $j \in [\tau]$, the maximum values and their respective indices match the ones computed by the signer. Therefore, the verifier can recompute

$$h'_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, (j, M'_{i_j^{\max}}, i_j^{\max})_{j\in[\tau]}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}\right)$$

and proceed with the protocol as before.
This version of the signature uses a string of $2\tau\lambda$ bits to commit to the cGGM forest. In the random oracle model, finding two trees with the same commitment string is virtually impossible. In some sense, $2\tau\lambda$ bits is even overkill for our purpose. Therefore, to improve the signing speed, we can instead use the global maximum across all $\tau$ subtrees as an implicit commitment to the cGGM forest. In particular, we can compute

$$M^{\max} \leftarrow \max \bigcup_{j\in[\tau]} \left\{\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right\}$$

and adapt the computation of the commitment hash accordingly:

$$h_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, M^{\max}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}\right).$$

This version has an abort probability of $\frac{1}{2^D}$. However, we only use a string of $2\lambda$ bits to commit to the cGGM forest if we do that. To find a middle ground between these two versions, we also include the global minimum into the commitment hash computation. In particular, we compute

$$M^{\min} \leftarrow \min \bigcup_{j\in[\tau]} \left\{ \mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]} \right\}$$

and compute the commitment hash as

$$h_{\mathrm{com}} \leftarrow \mathcal{H}_3 \left( \mathsf{salt}, M^{\min}, M^{\max}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*} \right).$$

Similar to before, if one of the challenge indices $(i_j^*)_{j\in[\tau]}$ coincides with the index of the minimum or with the index of the maximum or if the extrema are not unique, we abort. In this version, the verifier recomputes the minimum and maximum values among the commitment hashes as

$$\begin{aligned} M^{\min'} &\leftarrow \min \bigcup_{j\in[\tau]} \left( \bigcup_{i\in[2^D]\setminus\{i_j^*\}} \{\mathrm{com}_j{}^{[\![i]\!]}\} \right), \\ M^{\max'} &\leftarrow \max \bigcup_{j\in[\tau]} \left( \bigcup_{i\in[2^D]\setminus\{i_j^*\}} \{\mathrm{com}_j{}^{[\![i]\!]}\} \right). \end{aligned}$$

Since $i_j^*$ does not equal to the position of the minimum or maximum for $j \in [\tau]$, the recomputed minimum and maximum value match the ones computed by the signer and the verifier can proceed as before.
The abort probability of this version is upper bounded by $\frac{2}{2^D}$ and we use a string of size $4\lambda$ bits to commit to the cGGM forest. The signature size of this new version is $\lambda^2 + 3\tau\lambda + 7\lambda$ bits, compared to $\lambda^2 + 5\tau\lambda + 7\lambda$ bits for the basic version. We denote the signature scheme with this MinMax commitment using the superscript $\mathrm{SBC}^{M}_{\mathrm{VOLE}}$. We provide a comparison of signature sizes and running times of this new version compared to the basic version in Table 5.3. We provide the adapted pseudocode of our signature scheme using the MinMax commitment. The signing algorithm is given in Figure 5.11, while the verification algorithm is given in Figure 5.12.

### 5.7.2 Comparison with other VOLEitH Signature Schemes

In this section, we compare our SBC signature scheme with other post-quantum VOLEitH based signatures from the literature. These include the NIST proposal using the security of AES and the Even-Mansour AES variant (EM-AES) [BBB+24c]. The ReSolveD signature scheme uses the Regular Syndrome Decoding (RSD) problem over $\mathbb{F}_2$. The signature scheme in [Bui24] is based on the Multivariate Quadratic (MQ) problem, while [BBGK24] is based on the Permuted Kernel Problem (PKP). In [BFG+24a], the authors construct signature schemes based on the $\mathrm{RSD}_s$ problem (which is a variant of RSD) and MinRank.

| Parameters | | $\text{SBC}_{\text{VOLE}}$ | | | $\text{SBC}^{M}_{\text{VOLE}}$ | | | |
|---|---|---|---|---|---|---|---|---|
| $D$ | $\tau$ | \|sgn\| | Sign | Verify | \|sgn\| | Pr[abort] | Sign | Verify |
| 9 | 15 | 3 458 B | 2.41 ms | 2.40 ms | 2 978 B | 0.39 % | 2.49 ms | 2.40 ms |
| 10 | 13 | 3 218 B | 1.58 ms | 1.57 ms | 2 802 B | 0.20 % | 1.61 ms | 1.57 ms |
| 11 | 12 | 3 170 B | 1.60 ms | 1.59 ms | 2 786 B | 0.10 % | 1.61 ms | 1.59 ms |
| 12 | 11 | 3 090 B | 1.79 ms | 1.77 ms | 2 738 B | 0.05 % | 1.80 ms | 1.77 ms |
| 13 | 10 | 2 978 B | 2.21 ms | 2.19 ms | 2 658 B | 0.02 % | 2.22 ms | 2.19 ms |
| 15 | 9 | 2 978 B | 5.13 ms | 5.02 ms | 2 690 B | 0.01 % | 5.14 ms | 5.02 ms |

**Table 5.3:** Signature sizes and running times for $\lambda = 128$ using $N = 2^D$ parties and $\tau$ rounds with the parameters $q = 2$, $k = 257$ and $n = 130$ for $\text{SBC}_{\text{VOLE}}$ and $\text{SBC}^{M}_{\text{VOLE}}$ using an AMD EPYC 9374F processor running at 3.85 GHz. We provide the average of the signing times for the $\text{SBC}^{M}_{\text{VOLE}}$ version.

| Scheme | Assumption | \|pk\| | \|sgn\| |
|---|---|---|---|
| FAEST [BBB+24c] | AES | 32 B | 5 006 B |
| FAEST-EM [BBB+24c] | EM-AES | 32 B | 4 566 B |
| ReSolveD [CLY+24b] | RSD over $\mathbb{F}_2$ | 96 B | 3 916 B |
| VOLEitH from MQ [Bui24] | MQ | 38 B | 3 792 B |
| VOLEitH from PKP [BBGK24] | PKP | 102 B | 3 686 B |
| Dual Support Decomposition in the Head [BFG+24a] | $\text{RSD}_s$ | 75 B | 2 851 B |
| Dual Support Decomposition in the Head [BFG+24a] | MinRank | 73 B | 2 813 B |
| Standard MPCitH SBC [HJ24a] | SBC | 48 B | 3 782 B |
| $\text{SBC}_{\text{VOLE}}$ | SBC | 48 B | 2 978 B |
| $\text{SBC}^{M}_{\text{VOLE}}$ | SBC | 48 B | 2 658 B |

**Table 5.4:** Size comparison of the $\text{SBC}_{\text{VOLE}}$ signature with other schemes from the literature for the security level of $\lambda = 128$ bits. For each scheme, we use the smallest signature size for this security level that is specified in the respective documentation.

**Inputs:** Secret key $\mathsf{sk} = (\vec{x}, \vec{y})$, public key $\mathsf{pk} = (\vec{u}, \vec{v})$ and a message $\mathsf{msg} \in \{0,1\}^*$

1. Sample $R_{\vec{x}} \leftarrow_\$ (\mathbb{F}_2)^{n-2}$ and $\mathsf{salt} \leftarrow_\$ \{0,1\}^{2\lambda}$ for the cGGM tree derivation
2. Compute pre-tree: $\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$
3. Fold pre-tree: $\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}\right)_{j\in[\tau]} \leftarrow \mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau-1]\!]}\right)$
4. For $j \in [\tau]$:
   - Get Tree $T_j$: $\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right) \leftarrow \mathsf{cGGM}_{\mathsf{salt}}\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}, 2^D\right)$
   - Expand $T_j$ and get $\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D-1]\!]}\right)$, $\left(z_j{}^{[\![0]\!]}, \ldots, z_j{}^{[\![2^D-1]\!]}\right)$ and $\left(\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\right)$
   - Fold Tree $T_j$:
     - $\left(\vec{x}_j^{[\![0]\!]_i}, \vec{x}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D-1]\!]}\right)$
     - $\left(\vec{y}_j^{[\![0]\!]_i}, \vec{y}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D-1]\!]}\right)$
     - $\left(z_j{}^{[\![0]\!]_i}, z_j{}^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(z_j{}^{[\![0]\!]}, \ldots, z_j{}^{[\![2^D-1]\!]}\right)$
   - Let $z \leftarrow z_0{}^{[\![0]\!]_0} \oplus z_0{}^{[\![1]\!]_0}$
   - Compute offsets $\delta_{\vec{y}_j} \leftarrow \vec{y} \oplus \vec{y}_j^{[\![0]\!]_0} \oplus \vec{y}_j^{[\![1]\!]_0}$ and $\delta_{z_j} \leftarrow z \oplus z_j{}^{[\![0]\!]_0} \oplus z_j{}^{[\![1]\!]_0}$
5. $M^{\min} \leftarrow \min \bigcup_{j\in[\tau]} \{\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\}$
6. $M^{\max} \leftarrow \max \bigcup_{j\in[\tau]} \{\mathrm{com}_j{}^{[\![0]\!]}, \ldots, \mathrm{com}_j{}^{[\![2^D-1]\!]}\}$
7. If $M^{\min}$ or $M^{\max}$ are not unique: `ABORT`
8. $h_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, M^{\min}, M^{\max}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}\right)$
9. $h_0 \leftarrow \mathcal{H}_0\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}\right)$
10. $(\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]}) \leftarrow \mathrm{PRG}_0(h_0)$
11. Compute $\vec{A}_{F_x} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{x}_j^{[\![0]\!]_i} \alpha_i$
12. Compute $\vec{A}_{F_y} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$
13. Compute the coefficients $A$ and $B$ of $\Phi$ as in the First Response algorithm
14. Compute $b \leftarrow z + \vec{y} \cdot \vec{\mu}$
15. For $j \in [\tau]$:
    - Compute $\vec{A}_{F_y}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$ and $\vec{A}_{F_z}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} z_j{}^{[\![0]\!]_i} \alpha_i$
    - Compute $a^{(j)} \leftarrow \vec{A}_{F_z}^{(j)} + \vec{A}_{F_y}^{(j)} \cdot \vec{\mu}$
16. $h_1 \leftarrow \mathcal{H}_1\left(\mathsf{msg}, h_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, h_0, A, B, \left(a^{(j)}\right)_{j\in[\tau]}, b\right)$
17. $(i_j^*)_{j\in[\tau]} \leftarrow \mathrm{PRG}_1(h_1)$
18. If there is an index $j \in [\tau]$ with $i_j^* = i^{\min}$ or $i_j^* = i^{\max}$: `ABORT`
19. Compute the pre-challenge $i_p^* = \sum_{j=0}^{\tau-1} \lfloor i_j^* / 2^{D-1} \rfloor \cdot 2^j$
20. Compute the pre-tree PPRF key $K_{i_p^*}$ and the PPRF key $K_{i_0^*,\ldots,i_{\tau-1}^*}$
21. Output $\sigma \leftarrow \left(h_1, \mathsf{salt}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}\right)$

**Figure 5.11:** Signing algorithm of the $\mathrm{SBC}^M_{\mathrm{VOLE}}$ signature scheme.

**Inputs:** Public key $\mathsf{pk} = (\vec{u}, \vec{v})$, a message $\mathsf{msg} \in \{0,1\}^*$ and a signature $\sigma = \left(h_1, \mathsf{salt}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}\right)$

1. $(i_j^*)_{j\in[\tau]} \leftarrow \mathrm{PRG}_1(h_1)$
2. Recompute, expand and fold all trees (with one missing leaf each) using $K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}$ and $\mathsf{salt}$
3. $M^{\min'} \leftarrow \min \bigcup_{j\in[\tau]} \left(\bigcup_{i\in[2^D]\setminus\{i_j^*\}} \{\mathrm{com}_j{}^{[\![i]\!]}\}\right)$
4. $M^{\max'} \leftarrow \max \bigcup_{j\in[\tau]} \left(\bigcup_{i\in[2^D]\setminus\{i_j^*\}} \{\mathrm{com}_j{}^{[\![i]\!]}\}\right)$
5. For $(i,j) \in [D]\times[\tau]$, let $b_{i,j}$ denote the known position of the respective hypercube binary sharings
6. $h'_{\mathrm{com}} \leftarrow \mathcal{H}_3\left(\mathsf{salt}, M^{\min'}, M^{\max'}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}\right)$
7. $\delta_{z_0} \leftarrow 0$
8. $h'_0 \leftarrow \mathcal{H}_0\left(\mathsf{msg}, h'_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}\right)$
9. $(\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]}) \leftarrow \mathrm{PRG}_0(h'_0)$
10. For $j \in [\tau]$:
    - Compute $\Delta_j^* \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} b_{i,j}\alpha_i$
    - Compute $\vec{F}_y^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i \left(\vec{y}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{\vec{y}_j}\right)$
    - Compute $Z^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i \left(z_j{}^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{z_j}\right)$
    - $a^{(j)'} \leftarrow Z^{(j)}\left(\Delta_j^*\right) \oplus \vec{F}_y^{(j)}(\Delta_j^*)\cdot\vec{\mu} \oplus b\Delta_j^*$
11. $\Delta^* \leftarrow \sum_{j=0}^{\tau-1} \Delta_j^*$
12. $\vec{F}_x(\Delta^*) \leftarrow \sum_{m=0}^{\tau\cdot D-1} \vec{x}_m^{[\![b_m]\!]}\alpha_m$
13. $\vec{F}_y(\Delta^*) \leftarrow \sum_{j=0}^{\tau-1} \vec{F}_y^{(j)}(\Delta_j^*)$
14. Compute $\Phi(\Delta^*)$ as
$$\left(\vec{u}\cdot\begin{pmatrix}\vec{F}_x(\Delta^*)\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_y(\Delta^*)\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{F}_y(\Delta^*)\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_x(\Delta^*)\\1\\0\end{pmatrix}\right)$$
15. $A' \leftarrow \Phi(\Delta^*) - B\Delta^*$
16. $h'_1 \leftarrow \mathcal{H}_1\left(\mathsf{msg}, h'_{\mathrm{com}}, \mathsf{salt}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}, h'_0, A', B, \left(a^{(j)'}\right)_{j\in[\tau]}, b\right)$
17. If $h'_1 = h_1$ output `ACCEPT`, otherwise output `REJECT`

**Figure 5.12:** Verification algorithm of the $\mathrm{SBC}^M_{\mathrm{VOLE}}$ signature scheme.

CHAPTER 6

# Fherret: Proof of FHE Correct-and-Honest Evaluation from MPCitH

This chapter is based on the preprint available on the cryptology ePrint archive [HJS25]. Both Giacomo Santato and I are the main authors of this publication, under the supervision of our PhD advisor Antoine Joux. I am mainly responsible for the parts related to MPCitH techniques, while Giacomo Santato focused on the parts related to FHE. For completeness, the whole content of the publication is included in this chapter to illustrate the use case of techniques from the MPCitH paradigm for FHE and to present the Fherret scheme. To obtain a consistent notation throughout this dissertation, there are some minor notational changes compared to the original version.

## 6.1 Introduction

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that enables computation on encrypted data without requiring decryption. Specifically, a client encrypts a message $m$ and sends the ciphertext to a server. The server, having chosen a function $f$, evaluates it homomorphically on the ciphertext and returns a result that decrypts to $f(m)$.

The major FHE schemes [BGV12, FV12, DM15, CGGI20] guarantee the privacy of the encrypted message only in the honest-but-curious setting, when the server follows the protocol without deviating. However, various attacks in the literature have shown that an actively malicious server can recover sensitive information by executing an incorrect function or tampering with ciphertexts.

In this setting, security concerns arise when a dishonest server attempts to extract information by observing the client's reaction during decryption. These attacks can be described by giving a verification oracle to the adversary.

Verification oracle attacks are well-known in the FHE literature: in [LMSV12] and [ZPS12], an adversary can construct a decryption oracle by observing the reactions of the decrypting party. In [CGG16] and [CCCM22], the secret key gets recovered in a similar way. Analogous attacks were studied for the case of client-aided outsourced computations in [AGHV22] and [AV21]. Here, the attacker substitutes the ciphertexts with freshly encrypted ones during the interactions with the client to learn the encrypted message without any decryption happening.

Another attack strategy involves breaking the correctness of the computation. Li and Micciancio [LM21] present a key recovery attack on certain approximate FHE schemes that do not satisfy the IND-CPA$^{\mathsf{D}}$ security definition they introduce. This definition is theoretically equivalent to IND-CPA for exact schemes. While their attack only applies to approximate schemes like CKKS, this attack has recently been extended to exact schemes

[CCP+24, CSBB24] where adversaries compromise correctness by manipulating function evaluations. These new IND-CPA$^{\mathsf{D}}$-style attacks against exact schemes can be viewed as reaction-based attacks, exploiting the client's response when correctness is violated.
A common countermeasure to reaction-based attacks is to require a proof of the correct behavior of the server. This allows the user to refuse to decrypt unverified ciphertexts. However, most existing proofs require that the function $f$ which is evaluated homomorphically by the server is also known by the client.
In many FHE applications, such as private set intersection, privacy-preserving neural network inference, and genomic data analysis, the server must keep the evaluated function secret. While standard FHE ensures the confidentiality of user inputs, it does not inherently protect $f$. In other words, the ciphertext returned by the server may leak information about the function. To prevent this, specialized variants of FHE enforce circuit privacy.
Circuit privacy is a crucial property in FHE, ensuring that the ciphertext produced by the server reveals no information about $f$ beyond the fact that it decrypts to $f(m)$.
Currently, the only known techniques for proving correct FHE evaluation while preserving circuit privacy rely on zk-SNARKs. However, this approach suffers from significant overhead, as the prover must account for all FHE operations that do not directly affect the plaintext but only the ciphertext (e.g., bootstrapping, modulus switching, noise flooding, and relinearization). As a result, the computational cost of generating the proof becomes prohibitively high.

### 6.1.1 Our Contribution

We present Fherret (FHE coRREcT-and-Honest Evaluation Proof), a new scheme designed to protect FHE schemes against reaction-based attacks in settings where the evaluated functions need to remain confidential. Our contributions can be summarized as follows:
**A Novel Approach to FHE Integrity.** We present a new method for ensuring the integrity of homomorphic computations while preserving circuit privacy. Unlike previous methods that relied on zk-SNARKs of the homomorphic evaluation circuit, we leverage the MPC-in-the-Head (MPCitH) paradigm among $N$ parties using $\tau$ rounds. Our approach operates in the random oracle model (ROM) and can be applied to any circuit-private FHE scheme without additional assumptions.
**Operations are performed in the plaintext space.** Rather than proving the evaluation of the FHE circuit itself, Fherret constructs proofs directly within the FHE scheme. This approach allows us to bypass the need for proving computation-heavy tasks that do not affect the encrypted message, such as bootstrapping, modulus switching, and noise flooding. These operations are simply executed as part of the protocol, significantly improving efficiency.
**Scalability with FHE efficiency.** The efficiency of proof generation and verification scales directly with the performance of the underlying FHE scheme. Therefore, our approach benefits from any hardware acceleration designed to optimize FHE computations. Furthermore, Fherret is inherently parallelizable, enabling both the prover and verifier to achieve computational performance comparable to that of homomorphically evaluating a random function from $\mathcal{F}$.
**Scheme Universality.** Fherret is agnostic to the specific FHE scheme used and can be applied to any non-approximate FHE scheme. This also means it does not interfere with packing, SIMD, modified bootstrapping, NTT/RNS optimizations, or scheme switching. The only requirements are correctness and circuit privacy.

**Security against reaction-based attacks.** Fherret provides security against verification oracle attacks, ensuring that reaction-based adversaries cannot exploit the decryption process to recover sensitive information. More than that, it guarantees that correctness is preserved through all the server's homomorphic evaluation, preventing the FHE scheme from becoming approximate and protecting it from IND-CPA$^{\mathsf{D}}$-style attacks.

**Publicly Verifiable.** The verification of the proof can be performed by any user who has access to the public key of the FHE scheme, to the input ciphertext of the client, and the proof itself. Even without the secret key and without learning anything about the function that was evaluated or about the result of the evaluation, any third party can still verify the proof.

**Implementation.** We provide a proof of concept implementation of our scheme in C++ applied to the BGV-RNS scheme, as implemented in the OpenFHE library. We also discuss implementation-related optimizations that allow us to strongly reduce the number of FHE evaluations needed in the scheme.

### 6.1.2 Related Work

**Verifiable computation.** A significant line of research focuses on enhancing FHE schemes with additional guarantees. One prominent direction is the integration of *verifiable computation* (VC) into FHE.

These methods ensure that the decrypted message corresponds to the honest homomorphic evaluation of a function that is *known* by the verifier, making them incompatible with circuit privacy. Examples include Homomorphic Message Authentication Codes [GW13, CF13, FGP14, CKP+24], Zero-Knowledge Proofs [GGW24, CCC+25, ABPS24] and Trusted Execution Environments [NLDD21]. A more comprehensive survey can be found in [VKH23].

Some ZK-based VC schemes allow the function to be partially or fully private. While these could be analyzed within our framework, our focus differs fundamentally from VC. VC aims to minimize verifier costs, often achieving sublinear or even constant verification complexity. In contrast, our approach prioritizes protection against a dishonest server, accepting a linear verification cost.

The majority of the ZK-based VC schemes for FHE that support the evaluation of a private function [FNP20, BCFK21, GNS21, ABPS24] require the prover to prove the correct execution of the FHE evaluation circuit itself. As previously discussed, operating in the ciphertext space significantly increases computational complexity due to the inclusion of FHE maintenance operations (e.g., bootstrapping, modulus switching, and relinearization). Consequently, prover overhead remains high even for simple functions. For instance, in [VKH23], functions with a homomorphic evaluation time of 10–15 milliseconds require proof generation times of 5–7 minutes in the most optimized cases. Moreover, to achieve reasonable efficiency, most VP constructions are tailored to specific FHE schemes.

Finally, some ZK-based VC schemes for FHE support the evaluation of a private function while operating in the plaintext space [ACGS24, GGW24, GBK+24, ZLW+25] using techniques based on Interactive Oracle Proofs (IOPs) or MPCitH and VOLEitH. These techniques are also used in our protocol. Unfortunately, these schemes rely on evaluating a zk-SNARK homomorphically within the FHE scheme. This approach makes the constructions vulnerable to reaction-based attacks, as the proof must be decrypted before verification, which renders these schemes susceptible to attacks from a dishonest server.

A detailed discussion on verification attacks against such constructions is provided in [ZWL+25], highlighting their incompatibility with our security model.

**Achieving stronger security definitions.** While IND-CCA2 security is unattainable for malleable encryption schemes like FHE, researchers have explored several intermediate security notions. Numerous studies have investigated IND-CCA1 security for FHE schemes. Recently, Manulis and Nguyen showed that IND-CCA1 security can be achieved even in the presence of a bootstrapping key [MN24]. Although IND-CCA1 security allows an attacker access to a decryption oracle, this oracle is unavailable during the challenge phase. As a result, reaction-based attacks remain effective against IND-CCA1 schemes (see [DSA13] for further details).

A stronger definition, vCCA, which provides security against verification oracle attacks, was introduced in [MN24]. Their construction embeds the FHE scheme into an IND-CCA2-secure framework for robustness and applies SNARKs to the homomorphic evaluation for integrity. However, this approach is primarily of theoretical interest and remains impractical.

## 6.2 Preliminaries

### 6.2.1 Fully Homomorphic Encryption

We recall the definition of Fully Homomorphic Encryption [Riv87] in the public key setting.

**Definition 6.1** (Fully Homomorphic Encryption)**.** We define a fully homomorphic encryption scheme FHE as a tuple of four algorithms FHE = (KeyGen, Enc, Eval, Dec) with the following syntax.

$\mathtt{KeyGen}(\lambda) \to (\mathsf{sk}, \mathsf{pk})$: Given a security parameter $\lambda$, returns a secret key sk and a public key pk.

$\mathtt{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$: Given a public key pk and a message $m$, returns a ciphertext ct.

$\mathtt{Eval}(\mathsf{pk}, f, \mathsf{ct}) \to \mathsf{ct}_{\mathsf{out}}$: Given a public key pk, a function $f$, and a vector of ciphertexts ct, returns a ciphertext $\mathsf{ct}_{\mathsf{out}}$.

$\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$: Given a secret key sk and a ciphertext ct, returns a message $m$.

An FHE scheme is said to be correct if, for any pair $(\mathsf{sk}, \mathsf{pk})$ output by KeyGen, for any function $f$ and for any vector of plaintexts $m$, where the number of elements in the vector is equal to the number of inputs of $f$, the following holds:

$$\mathtt{Dec}(\mathsf{sk}, (\mathtt{Eval}(\mathsf{pk}, f, \mathsf{ct}))) = f(m),$$

where ct is the vector of the encryptions of elements of $m$, i.e., $\mathsf{ct}_i \leftarrow \mathtt{Enc}(\mathsf{pk}, m_i)$.

We recall the definition of Circuit Privacy for FHE schemes.

**Definition 6.2** (Circuit Privacy [Gen09a])**.** An FHE scheme is called *Circuit Private* on $\mathcal{F}$ if there is a probabilistic polynomial-time simulator $\mathcal{S}_{\mathsf{CP}}$, such that, for any vector of valid ciphertexts ct,

$$\{\mathsf{sk}, \mathcal{S}_{\mathsf{CP}}(\mathsf{pk}, m_{\mathsf{out}})\} \approx_{\mathsf{c}} \{\mathsf{sk}, \mathtt{Eval}(\mathsf{pk}, f, \mathsf{ct})\},$$

where $f \in \mathcal{F}$, $m_{\mathsf{out}} \leftarrow f(\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}))$ and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(\lambda)$.

We recall the definition of IND-CPA security.

**Definition 6.3** (IND-CPA security)**.** Let $\mathsf{FHE} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{Dec})$ be a fully homomorphic encryption scheme. We define the IND-CPA game as the experiment $\mathtt{Exp}^{\mathsf{IND\text{-}CPA}}$, where $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is an adversary. The experiment is defined as follows:

$$\begin{aligned}\mathtt{Exp}^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](\lambda): \quad & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{KeyGen}(\lambda)\\ & (m_0, m_1, s) \leftarrow \mathcal{A}_1(\mathsf{pk})\\ & b \leftarrow_{\$} \{0,1\}\\ & c^* \leftarrow \mathtt{Enc}(\mathsf{pk}, m_b)\\ & b' \leftarrow \mathcal{A}_2(\mathsf{pk}, s, c^*)\\ & \textbf{return } b = b'\end{aligned}$$

We say that an FHE scheme is IND-CPA-secure if any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has a negligible advantage. The advantage here is defined as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](\lambda) = \left|\Pr\left[\mathtt{Exp}^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](\lambda) = 1\right] - \frac{1}{2}\right|.$$

We now describe three oracles $\mathcal{E}, \mathcal{H}$, and $\mathcal{D}$ used in the definition of $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security. These oracles share a common state $S$, which is initially empty. Each entry in $S$ is a triple $(m_0, m_1, \mathsf{ct})$, where $m_0, m_1$ are messages from the message space, and $\mathsf{ct}$ is a ciphertext from the ciphertext space.

- The encryption oracle $\mathcal{E}$ operates as follows: it takes as input a pair of messages $(m_0, m_1)$, randomly selects a challenge bit $b \in \{0,1\}$, and returns the ciphertext $\mathsf{ct} = \mathtt{Enc}(\mathsf{pk}, m_b)$ to the adversary. Additionally, the oracle adds the triple $(m_0, m_1, \mathsf{ct})$ to the global state $S$.
- The homomorphic evaluation oracle $\mathcal{H}$ takes as input a function $g$ and a vector of ciphertexts $\mathsf{ct}$ from the state $S$. It computes a new ciphertext $\mathsf{ct}' = \mathtt{Eval}(\mathsf{pk}, g, \mathsf{ct})$, returns $\mathsf{ct}'$ to the adversary and adds the corresponding plaintext evaluations of the form $(g(m_0), g(m_1), \mathsf{ct}')$ to the global state $S$.
- The decryption oracle $\mathcal{D}$ accepts as input a ciphertext $\mathsf{ct}$ from the state $S$ and returns the decryption only if the associated messages satisfy $m_0 = m_1$. This constraint prevents the adversary from trivially determining the challenge bit $b$ from the decryption result. Any information gained must arise from the ciphertext's internal structure rather than from direct comparison of the underlying messages.

We provide the pseudocode of these three oracles in Algorithm 16. We recall the formal definition of $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security from [LM21, Definition 2].

**Definition 6.4** ($\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security)**.** Let $\mathsf{FHE} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{Dec})$ be a fully homomorphic encryption scheme. We define the $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security experiment $\mathtt{Exp}_b^{\mathsf{IND\text{-}CPA}^{\mathsf{D}}}$ with respect to an adversary $\mathcal{A}$, having access to three stateful oracles $\mathcal{E}, \mathcal{H}, \mathcal{D}$ described in Algorithm 16.
The experiment is defined as follows:

$$\begin{aligned}\mathtt{Exp}_b^{\mathsf{IND\text{-}CPA}^{\mathsf{D}}}[\mathcal{A}](\lambda): \quad & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{KeyGen}(\lambda),\\ & b' \leftarrow \mathcal{A}^{\mathcal{E}^b_{\mathsf{pk}}, \mathcal{H}^b_{\mathsf{pk}}, \mathcal{D}^b_{\mathsf{sk}}}(\lambda),\\ & \textbf{return } b'.\end{aligned}$$

We say that an FHE scheme is IND-CPA$^{\mathsf{D}}$-secure if any PPT adversary $\mathcal{A}$ has a negligible advantage. The advantage here is defined as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA^D}}[\mathcal{A}](\lambda) = \left| \Pr\left[ \mathtt{Exp}_b^{\mathsf{IND\text{-}CPA^D}}[\mathcal{A}](\lambda) = b \right] - \frac{1}{2} \right|.$$

**Algorithm 16** Oracles for the IND-CPA$^{\mathsf{D}}$ game

**Initialization:**
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$

**global state:**
  $S \leftarrow \emptyset$
  $i \leftarrow 0$

**Oracle** $\mathcal{E}^b_{\mathsf{pk}}(m_0, m_1)$ **:**
  $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
  $S[i] \leftarrow (m_0, m_1, \mathsf{ct})$
  $i \leftarrow i + 1$
  **return** ct

**Oracle** $\mathcal{H}^b_{\mathsf{pk}}(g, J = (j_1, \ldots, j_k))$**:**
  $\mathsf{ct} \leftarrow \mathsf{Eval}(\mathsf{pk}, g, (S[j_1].\mathsf{ct}, \ldots, S[j_k].\mathsf{ct}))$
  $gm_0 \leftarrow g(S[j_1].m_0, \ldots, S[j_k].m_0)$
  $gm_1 \leftarrow g(S[j_1].m_1, \ldots, S[j_k].m_1)$
  $S[i] \leftarrow (gm_0, gm_1, \mathsf{ct})$
  $i \leftarrow i + 1$
  **return** ct

**Oracle** $\mathcal{D}^b_{\mathsf{sk}}(i)$ **:**
  **if** $S[i].m_0 = S[i].m_1$
    **return** $\mathsf{Dec}(\mathsf{sk}, S[i].\mathsf{ct})$
  **else**
    **return** $\perp$

### 6.2.2 Multi-Party Computation

This section introduces the notations we use for *Multi-Party Computation* (MPC) protocols. We use additive sharings of polynomials over rings. Let $N$ be the number of parties that interact in the MPC protocol. An $N-$*sharing* of a finite ring element $x \in R$ is an $N$-tuple

$$[\![x]\!] = \left( x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]} \right)$$

such that

$$x = \sum_{i=0}^{N-1} x^{[\![i]\!]} \quad (\text{in } R).$$

We call $x^{[\![i]\!]}$ the $i$-th *share* of $x$. In the protocols we consider, each party $\mathcal{P}_i$ for $i \in [N]$ receives one share $x^{[\![i]\!]}$ for each shared value $x$.
A sharing $[\![x]\!]$ is usually obtained by computing $N-1$ random values $x^{[\![0]\!]}, \ldots, x^{[\![N-2]\!]}$, and setting

$$x^{[\![N-1]\!]} = x - \sum_{i=0}^{N-2} x^{[\![i]\!]}$$

afterwards to obtain a valid sharing $[\![x]\!]$ of $x$.
During the MPC protocol, the parties can perform different computations with their respective shares to compute the output $g(x)$ for some function $g$. In this setup, $\mathcal{P}_i$ outputs its own value of $g$, which we denote by $g_i$.
With these sharings, the parties can perform different computations independently. Assume that party $\mathcal{P}_i$ receives the shares $x^{[\![i]\!]}$ and $y^{[\![i]\!]}$ corresponding to sharings of $x$ and $y$, and let $\alpha \in R$ be a constant. With these shares, the parties can perform the following computations:

- **Addition**: They can compute $[\![x+y]\!]$ by locally setting

  $$(x+y)^{[\![i]\!]} = x^{[\![i]\!]} + y^{[\![i]\!]}$$

  for $i \in [N]$.

- **Multiplication by a constant**: They can compute $\alpha[\![x]\!] = [\![\alpha x]\!]$ by locally setting

  $$(\alpha x)^{[\![i]\!]} = \alpha x^{[\![i]\!]}$$

  for $i \in [N]$.

### 6.2.3 Using the MPCitH Paradigm

The construction of our protocol relies on the *MPC-in-the-Head* (MPCitH) paradigm, which was introduced in [IKOS07]. We consider an MPC protocol between $N$ parties $\mathcal{P}_0, \ldots, \mathcal{P}_{N-1}$, that securely and correctly computes the output of a function $g$, given a secret input $x$. In this setting, the secret $x$ is given by a sharing $[\![x]\!]$, where party $i$ receives the $i$-th share $x^{[\![i]\!]}$. The function $g$ outputs either 1 or 0, corresponding to accept or reject, respectively. In our protocol, we require that the MPC protocol is $(N-1)$-private, meaning that the views of any $N-1$ out of $N$ parties reveal no information about the secret $x$. With this type of MPC protocol, a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ of the knowledge of a witness $x$ with $g(x) = 1$ by constructing a Zero-Knowledge protocol as follows:

- $\mathcal{P}$ generates a random sharing $[\![x]\!]$ of $x$.
- $\mathcal{P}$ simulates ("in the head") all parties of the MPC protocol and sends commitments to each party's view to the verifier.
- $\mathcal{V}$ randomly selects $N-1$ parties whose views the prover must reveal.
- The verifier checks whether these revealed views are consistent with an honest execution of the MPC protocol and the prover's commitments.

Because the MPC protocol is $(N-1)$-private, opening the views of all but one party reveals no information about the secret $x$. Since the choice of the $N-1$ opened parties is random, a malicious prover can cheat with probability $\frac{1}{N}$ by corrupting the computation

of the unopened party. Thus, the *soundness error* of a protocol based on the MPCitH paradigm is (at least) $\frac{1}{N}$. To amplify the soundness to the required security level of $\lambda$ bits, we use $\tau$ repetitions of the protocol, resulting in a soundness error of $N^{-\tau}$.

### 6.2.4 Hypercube Folding

To reduce computational overhead, MPCitH protocols commonly use the *hypercube technique*, introduced in [AGH+23].
Assume the number of parties in the MPC protocol is a power of two, i.e., $N = 2^D$. The hypercube technique transforms one instance of an MPC protocol between $N$ parties into $D$ instances of the MPC protocol between 2 parties.
By computing $D$ instances in parallel, the total soundness error of $\frac{1}{N} = \frac{1}{2^D} = (\frac{1}{2})^D$ is the same as in the original protocol between $N$ parties.
The process of converting a sharing between $N = 2^D$ parties into $D$ sharings between 2 parties is called *folding* of the shares. Consider a sharing of $x$ of the form $x = \sum_{i=0}^{N-1} x^{[\![i]\!]}$. Let $B_j(i)$ denote the $j$-th bit of the binary decomposition of an integer $i$. For any fixed $j \in [D]$, we see that

$$x = \sum_{B_j(i)=0} x^{[\![i]\!]} + \sum_{B_j(i)=1} x^{[\![i]\!]} \tag{6.1}$$

is a sharing of $x$ between 2 parties. In this setting, the first party receives the share

$$x_j^{[\![0]\!]} = \sum_{B_j(i)=0} x^{[\![i]\!]},$$

while the second party receives the share

$$x_j^{[\![1]\!]} = \sum_{B_j(i)=1} x^{[\![i]\!]}.$$

This holds for every $j \in [D]$, resulting in $D$ sharings of $x$ of the form $x = x_j^{[\![0]\!]} + x_j^{[\![1]\!]}$. We refer to this construction by

$$\left(x_j^{[\![0]\!]}, x_j^{[\![1]\!]}\right)_{j\in[D]} \leftarrow \mathsf{Folding}\left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right).$$

We can use the algorithm with running time $\mathcal{O}(N)$ introduced in Section 4.7.2 to compute the folding.

### 6.2.5 Reducing the Communication Cost

To reduce the communication cost of our protocol, we again use PPRFs. We recall the definition and introduce the notation we use in the GGM tree construction for Fherret.

**Definition 6.5.** A family $G$ of puncturable PRFs over $[N]$ is a PRF family $G$ indexed by a key $K$ with domain $[N]$, satisfying the following conditions:

- For each key $K$ and index $i^* \in [N]$, there exists a punctured key $K_{i^*}$ and an algorithm $\mathcal{A}$ such that

$$\text{for each } j \in [N] \setminus \{i^*\} : \mathcal{A}(K_{i^*}, j) = G_K(j).$$

- The punctured key $K_{i^*}$ reveals no information about $G_K(i^*)$.

**GGM trees.** A common method to instantiate PPRFs in practice is the tree-based PRF construction by Goldreich, Goldwasser and Micali, known as *GGM trees* [GGM86]. This construction builds a binary tree of depth $\lceil \log(N) \rceil$ with $N$ leaves, where the root node is labeled with a master root seed. Each node's left and right children are labeled inductively using a length-doubling *pseudorandom generator* (PRG) on the respective parent node. In our scheme, we use a (salted) hash function with domain separation to instantiate this descent function. The $N$ leaves of this tree represent the $N$ shares of the value $x$ that we use in the MPCitH protocol. Given a root seed $r$, we denote the described expansion of this root into $N$ leaves by $\left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right) \leftarrow \mathrm{GGM(r, N)}$. To reveal all $N$ leaves except one leaf $i^*$ in a GGM tree, it suffices to reveal the labels of the siblings of the nodes on the path from the root to the leaf $i^*$. This way, all leaves except the leaf at position $i^*$ can be reconstructed by communicating $\lceil \log(N) \rceil$ nodes, instead of $N-1$ leaves.
To obtain a valid sharing of the value $x$, we need an *offset* $\delta_x$ if we use this basic GGM tree construction. This offset is computed as

$$\delta_x = x - \sum_{i=0}^{N-1} x^{[\![i]\!]}$$

and is broadcast to all parties.
**Consistency for large values.** In MPCitH schemes, the bitsize of the tree nodes is typically set to $\lambda$, while also using a salt of $2\lambda$ bits in the derivation function. Given $\mathrm{salt} \leftarrow \{0,1\}^{2\lambda}$ and a hash function $\mathcal{H}$, we define $\mathcal{H}^{\mathrm{salt}}(\mathrm{msg}) := \mathcal{H}(\mathrm{salt} \parallel \mathrm{msg})$ for any $\mathrm{msg} \in \{0,1\}^*$. If we use the salted hash function for the GGM tree derivation, we use the notation $\mathsf{GGM}_{\mathrm{salt}}$ instead. For our scheme, we require sharings of functions $f \in \mathcal{F}$. We can use the GGM tree construction for $f$ by using internal nodes of the tree with the same bitsize as elements in $\mathcal{F}$. Since this bitsize is often much larger than the desired security parameter $\lambda$, a GGM construction using such large inner nodes is expensive. Instead, we can use the standard GGM tree construction described above with inner nodes of just $\lambda$ bits. For these trees, we can expand each leaf to more than $\lambda$ bits using a pseudorandom generator to achieve the desired bitsize of elements in $\mathcal{F}$. These additional bits require offsets and we have to ensure that the same value is consistently shared among all trees. Indeed, if we use $\tau$ rounds of the MPCitH protocol, we obtain $\tau$ such trees $T_j$, each having one offset value $\delta_{f_j}$ for $j \in [\tau]$. The leaves of each such tree, together with the offset value, correspond to a function

$$f_j = \sum_{i=0}^{N-1} f_j^{[\![i]\!]} + \delta_{f_j}.$$

However, from the verifier's perspective, these $f_j$ do not have to be equal across rounds. A state-of-the-art method for ensuring the desired consistency of sharings across multiple rounds was introduced by Baum et al. in [BBD+23]. The idea is to use a non-interactive version of the SoftspokenOT technique [Roy22] to ensure that the prover provides offsets resulting in a sharing of the same element across multiple rounds based on a probabilistic check. This technique is utilized in many schemes that employ a similar paradigm to MPCitH, known as VOLEitH (Vector Oblivious Linear Evaluation in the Head), such as in [CLY+24a, BFG+24b, Bui24].
In our scheme, we use a subroutine for the consistency check based on the techniques

introduced in Section 5.4.5. We adapt this consistency check to use it in the Fherret protocol and explain it in detail in Section 6.3.3.

## 6.3 The Fherret Scheme

We construct a proof of the correct-and-honest homomorphic evaluation of a private function from $\mathcal{F}$ on a known ciphertext.
In Section 6.3.1, we introduce a basic version of the Fherret scheme to illustrate its fundamental mechanics, although this version is neither secure nor optimized. In Sections 6.3.2 and 6.3.3, we explain the attacks against the basic version and describe the modifications required to make it secure. In Section 6.3.4, we present the adapted scheme constructed to prevent attacks from Sections 6.3.2 and 6.3.3 and prove its security.

### 6.3.1 A Simple but Insecure Version of the Scheme

We construct an interactive protocol to prove the correct-and-honest homomorphic evaluation of a private function $f \in \mathcal{F}$ on a vector of ciphertexts $\mathsf{ct}$. The protocol runs between a prover/client $\mathcal{P}$ and a verifier/server $\mathcal{V}$, using the MPCitH paradigm. This initial version of the scheme is neither secure nor optimized but is designed to demonstrate the core principles.
The prover computes a random sharing $[\![f]\!]$ of $f$ between $N$ parties and then evaluates each share $f^{[\![i]\!]}$ on the ciphertext $\mathsf{ct}$, obtaining $\mathsf{ct}_i$ for $i \in [N]$. After that, $\mathcal{P}$ commits to them by hashing all of these ciphertexts and then computes $X$ as the bitwise XOR of all the $\mathsf{ct}_i$.
After receiving the commitment and $X$, $\mathcal{V}$ samples a random index $i^* \in [N]$ and sends it to $\mathcal{P}$. The prover responds by revealing every share of $f$, and the related random coins $\sigma_i$, except $f^{[\![i^*]\!]}$ and $\sigma_{i^*}$ to the verifier. Since the use of random shares ensures that the protocol is $N-1$ private, $\mathcal{V}$ gains no information about $f$.
Using these shares and these random coins, the verifier recomputes the evaluations of $f^{[\![i]\!]}$ on the ciphertext $\mathsf{ct}$, obtaining $\mathsf{ct}_i$ for $i \neq i^*$. Using $X$, $\mathcal{V}$ can also recompute the homomorphic evaluation of $f^{[\![i^*]\!]}$ on $\mathsf{ct}$. After doing this, $\mathcal{V}$ checks if the hashes of the $\mathsf{ct}_i$ match the commitment and obtains the result of the decryption of the homomorphic evaluation by decrypting all the $\mathsf{ct}_i$ and summing them together. A description of the $\tau$ round basic scheme is provided in Figure 6.1.
A malicious prover $\mathcal{P}'$ who attempts to use an invalid sharing must cheat in at least one position. This remains undetected by the verifier only if the cheating position equals $i^*$. By repeating this protocol for multiple rounds, we can decrease the cheating probability until it is negligible.
While this protocol already limits the ability of a malicious prover by checking the honesty of many of its computations, the verifier is still vulnerable to weaker versions of reaction-based attacks. In the following sections, we describe these attacks and outline how to fully protect the scheme against them.

### 6.3.2 Simple Error Correction During Decryption

The basic scheme in Figure 6.1 is still vulnerable to verification oracle attacks. The weakness that an attacker can exploit arises from the construction of the verification process.

**Inputs**:

- Public key $\mathsf{pk}$, function $f$, vector of ciphertexts $\mathsf{ct}$ (Prover)
- Secret key $\mathsf{sk}$, public key $\mathsf{pk}$, vector of ciphertexts $\mathsf{ct}$ (Verifier)

**Step 1: Commitment**

1. Let $f$ be a polynomial in $\mathcal{F}$
2. For $j \in [\tau]$:
   - For $i \in [N-1]$: Sample ${f_j}^{[\![i]\!]} \leftarrow_{\$} \mathcal{F}$
   - Compute the missing share ${f_j}^{[\![N]\!]} \leftarrow f - \sum_{i \in [N-1]} {f_j}^{[\![i]\!]}$
   - For $i \in [N]$: Compute $\mathsf{ct}_{i,j} \leftarrow \mathtt{Eval}(\sigma_{i,j}; \mathsf{pk}, {f_j}^{[\![i]\!]}, \mathsf{ct})$ and $h_{i,j} \leftarrow \mathcal{H}(i \parallel j \parallel \mathsf{ct}_{i,j})$
   - Compute $X_j \leftarrow \bigoplus_{i \in [N]} \mathsf{ct}_{i,j}$
3. Compute $\mathsf{com} \leftarrow \mathcal{H}\left((h_{i,j})_{(i,j) \in [N] \times [\tau]}\right)$
4. Send $(\mathsf{com}, X_0, \ldots, X_{\tau-1})$ to the verifier

**Step 2: Challenge**

1. For $j \in [\tau]$: sample $i_j^* \leftarrow_{\$} [N]$
2. Send $(i_0^*, \ldots, i_{\tau-1}^*)$ to the prover

**Step 3: Response**

1. Send $({f_j}^{[\![i]\!]}, \sigma_{i,j})$ for all $(i,j) \in [N] \times [\tau]$ except for the pairs $(i_j^*, j)$ for all $j \in [\tau]$ to the verifier

**Step 4: Verification**

1. For $j \in [\tau]$ :
   - For $i \in [N] \setminus \{i_j^*\}$: Compute $\mathsf{ct}_{i,j} \leftarrow \mathtt{Eval}(\sigma_{i,j}; \mathsf{pk}, {f_j}^{[\![i]\!]}, \mathsf{ct})$ and $h'_{i,j} \leftarrow \mathcal{H}(i \parallel j \parallel \mathsf{ct}_{i,j})$.
   - Compute $\mathsf{ct}_{i_j^*,j} \leftarrow \bigoplus_{i \neq i_j^*} \mathsf{ct}_{i,j} \bigoplus X_j$ and $h'_{i_j^*,j} \leftarrow \mathcal{H}(i_j^* \parallel j \parallel \mathsf{ct}_{i_j^*,j})$
   - Compute $m_{\mathsf{out},j} \leftarrow \sum_{i \in [N]} \mathtt{Dec}(\mathsf{sk}, \mathsf{ct}_{i,j})$
2. Compute $h' \leftarrow \mathcal{H}\left(\left(h'_{i,j}\right)_{(i,j) \in [N] \times [\tau]}\right)$
3. If $h' = \mathsf{com}$ and all $m_{\mathsf{out},j}$ for $j \in [\tau]$ are equal, output `accept`. Otherwise, output `reject`

**Figure 6.1:** Illustration of the basic ideas of the Fherret scheme (insecure).

In the final step, the user checks whether all $m_{\mathsf{out},j}$ for $j \in [\tau]$ are equal. If they are, the verifier accepts; otherwise, verification fails. Thus, the prover can modify the outcome by changing only one value among all $m_{\mathsf{out},j}$, which can be done with a non-negligible probability of $\frac{1}{N}$. Consequently, this scheme only reduces the effectiveness of the verification oracle by a factor of $N$ while still permitting the same attacks (such as [ZPS12, CCP+24, CSBB24]) based on the user's reaction during decryption.

To see this, consider an FHE scheme with RLWE ciphertexts $\mathsf{ct} = (a, b)$. Assume a server

is executing the scheme with $f$ being the identity function for $\tau - 1$ rounds. In the last round, after computing all the $\mathsf{ct}_{i,\tau-1}$ for $i \in [N]$, the server modifies one of them, say the $I$-th, by introducing some extra noise $e$ to the $b$ component and computing $\mathsf{ct}'_{I,\tau-1} = \mathsf{ct}_{I,\tau-1} + (0, e)$. It also adjusts $X_{\tau-1}$ and $h_{I,\tau-1}$ accordingly. With probability $\frac{1}{N}$, the challenge $i^*_{\tau-1}$ is equal to $I$. This implies that the final hash check succeeds because the user does not recompute $\mathsf{ct}_{i^*_{\tau-1},\tau-1}$ but retrieves it from $X_{\tau-1}$. As a result, the final output of the verification depends only on the unanimity check, which succeeds if and only if $\mathtt{Dec}(\mathsf{ct}'_{I,\tau-1}) = \mathtt{Dec}(\mathsf{ct}_{I,\tau-1})$. This allows the attacker to gather information about the magnitude of the RLWE error of $\mathsf{ct}_{I,\tau-1}$, potentially compromising the security of the FHE scheme.

To counteract these types of attacks, we modify the final check such that the verifier accepts if and only if more than half of the $m_{\mathsf{out},j}$ are equal. This forces an adversary to manipulate at least $\frac{\tau}{2}$ rounds to alter the verification result. To achieve this without failing the hash commitment check, the attacker would need to guess the verifier's challenge correctly for each of these rounds, which happens with a negligible probability of $\left(\frac{1}{N}\right)^{\frac{\tau}{2}}$. This implies that we can safely omit the equal majority check, because if the commitment check succeeds, the probability of the second check failing is negligible. Therefore, we can always expect that more than half of the $m_{\mathsf{out},j}$ are equal with probability almost 1.

To express this concept more easily in the protocols, we define the majority of any finite list $\mathcal{L}$ as:

$$\mathsf{maj}(\mathcal{L}) = \begin{cases} \text{the element in } \mathcal{L} \text{ that appears} > |\mathcal{L}|/2 \text{ times} \\ \perp \ \text{, if such an element does not exist.} \end{cases}$$

**Hypercube version of the scheme.** We can reduce the computational overhead for the prover and verifier by using the hypercube version of the scheme. In particular, we can transform $\tau$ parallel instances of the protocol between $N = 2^D$ parties into $\tau \cdot D$ parallel instances of the protocol between 2 parties. In the basic version of the scheme, the prover has to do $\tau \cdot N$ evaluations on $\mathsf{ct}$, while the verifier has to do $\tau \cdot (N-1)$ such evaluations. By considering the hypercube version instead, we can reduce this to $2\tau D$ evaluations on $\mathsf{ct}$ for the prover and $\tau \cdot D$ evaluations on $\mathsf{ct}$ for the verifier.

To achieve this, we use the hypercube folding algorithm on the leaves of each GGM tree we computed before to obtain $\tau \cdot D$ sharings of $f$ between two parties. Since we need to send one XOR $X_j$ of all $\mathsf{ct}_{i,j}$ per sharing to the verifier, the communication for the hypercube version increases by a factor of $D = \log(N)$. Indeed, we need to send $\tau \cdot D$ values $X_{i,j}$ in the hypercube version instead of $\tau$ values $X_j$ in the base scheme from Figure 6.1.

### 6.3.3 Adding a Consistency Check

Another problem with the scheme in Figure 6.1 is that the verifier cannot check if the prover is always computing shares of the same function during different rounds. Potentially, the prover could use different functions for different rounds without detection.

Even if the prover does not know the message $m$ that is encrypted in $\mathsf{ct}$, having access to a verification oracle could still allow the prover to obtain additional information about $m$. In particular, even considering the majority check from Section 6.3.2, the prover could choose two functions $f_0$ and $f_1$ and perform $\frac{\tau}{2}$ rounds of the protocol by computing shares of $f_0$ and $\frac{\tau}{2}$ rounds using shares of $f_1$. If the verification oracle outputs `accept`, this reveals

that $f_0(m) = f_1(m)$, providing additional information about $m$.[1]

**VOLE consistency check.** As discussed in Section 6.2.5, we reduce the computational overhead by using internal GGM tree nodes of size $\lambda$, initializing each tree $T_j$ with a random element $u_j \leftarrow_\$ \{0,1\}^\lambda$. After expanding each tree to $N$ leaves, we apply a PRG $G\colon \{0,1\}^\lambda \to \mathcal{F}$ to obtain shares in $\mathcal{F}$, then compute offsets $\delta_{f_j}$ to correct the sum to match a common $f$. To ensure the consistency of the functions across all $\tau$ subtrees to the verifier and to avoid the attack explained above, the prover needs to do some extra work to convince the verifier by adding a consistency check. To do that, we adopt a VOLE-in-the-Head–style check [BBD+23, HJ24b].

Consider the function $\vec{f} \in \mathcal{F}$ with $\log(|\mathcal{F}|) > \lambda$. We use vectorial notation to be consistent with VOLE literature. Let $\vec{u}_j \leftarrow_\$ \{0,1\}^\lambda$. First, we compute a GGM tree family $(T_0, \ldots, T_{\tau-1})$ for the small value $\vec{u}_j$ as described in Section 6.2.5 to obtain the leaves

$$\left(\vec{u}_j^{[\![0]\!]}, \ldots, \vec{u}_j^{[\![N-1]\!]}\right) \leftarrow \mathrm{GGM}_{\mathrm{salt}}\left(\vec{u}_j, N\right) \text{ for } j \in [\tau].$$

After computing the hypercube folding of each such sharing, we obtain

$$\left(\vec{u}_j^{[\![0]\!]_i}, \vec{u}_j^{[\![1]\!]_i}\right)_{i\in[D]} \leftarrow \mathsf{Folding}\left(\vec{u}_j^{[\![0]\!]}, \ldots, \vec{u}_j^{[\![N-1]\!]}\right)$$

for each $j \in [\tau]$, where $D = \log(N)$. In total, we obtain $\tau \cdot D$ such hypercube sharings. For each such sharing, the prover knows both shares, while the verifier learns only one of the two from the PPRF key. Indeed, the verifier learns all leaves of $T_j$ except one leaf $\vec{u}_j^{[\![i^*]\!]}$. Therefore, by using the folding algorithm, the verifier cannot recompute the hypercube shares that contain $\vec{u}_j^{[\![i^*]\!]}$ in the summation (6.1), depending on the bit decomposition of the index $i^*$.

To derive the shares corresponding to $\vec{f}$, we use a PRG on the leaves of each subtree. In particular, we derive these shares using

$$\vec{f}_j^{[\![i]\!]} \leftarrow \mathrm{PRG}_0\left(\vec{u}_j^{[\![i]\!]}\right)$$

for each $(i,j) \in [N] \times [\tau]$, where $\mathrm{PRG}_0\colon \{0,1\}^\lambda \to (\mathbb{F}_p)^{|\mathcal{F}|}$ is a pseudorandom generator. To correct these pseudorandom values in order to obtain a sharing of $\vec{f}$, we compute the offset values

$$\delta_{\vec{f}_j} = \vec{f} - \sum_{i=0}^{N-1} \vec{f}_j^{[\![i]\!]}$$

for each $j \in [\tau]$. For the consistency check described in Section 6.3.3, we also need to derive an additional value from the leaves. In particular, we derive $z_j{}^{[\![i]\!]} \leftarrow \mathrm{PRG}_1\left(\vec{u}_j^{[\![i]\!]}\right)$ for each $(i,j) \in [N] \times [\tau]$, where $\mathrm{PRG}_1\colon \{0,1\}^\lambda \to \mathbb{F}_{p^l}$ is a pseudorandom generator. We discuss the required size of $l$ depending on $p$ and $\lambda$ in Section 6.4. We set $z_0 = \sum_{i=0}^{N-1} z_0{}^{[\![i]\!]}$ and compute the offset values $\delta_{z_j} = z_0 - \sum_{i=0}^{N-1} z_j{}^{[\![i]\!]}$ for $j \in [\tau]^*$.

By construction, we obtain $\tau$ values of $\vec{f}_j$ and $z_j$ from the GGM tree family and the offset

[1] When working on a field, the set where two polynomials are equal is small by the Schwartz–Zippel lemma. Instead, when working modulo a power of 2 this set might grow noticeably and allow for easier attacks.

values, one for each of the $\tau$ subtrees. In particular, we have that

$$\vec{f_j} = \sum_{i=0}^{N-1} \vec{f_j}^{[\![i]\!]} + \delta_{\vec{f_j}} \text{ and } z_j = \sum_{i=0}^{N-1} {z_j}^{[\![i]\!]} + \delta_{z_j}$$

for $j \in [\tau]$, where $\delta_{z_0} = 0$. The idea of the consistency check is to convince the verifier that the values of $(\vec{f_j}, z_j)$ are equal across all rounds $j \in [\tau]$, which implies the desired equality of $\vec{f_j}$ across all rounds. In particular, we want that $(\vec{f_j}, z_j) = (\vec{f}, z)$ for $j \in [\tau]$. The purpose of $z$ is to act as a one time pad to hide the value of $\vec{f}$ in the equality test. In the hypercube setting, each subtree $T_j$ induces a vectorial function of the form

$$\vec{g}_{i,j}(b) = \left(\vec{f_j}^{[\![0]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + b \cdot \left(\vec{f_j}, z_j\right),$$

where the verifier chooses $b_{i,j}$ and learns $\vec{g}_{i,j}(b_{i,j}) = \left(\vec{f_j}^{[\![b_{i,j}]\!]_i}, {z_j}^{[\![b_{i,j}]\!]_i}\right)$. Using this representation, we can pick random coefficients $\alpha_m \in \mathbb{F}_{p^l}$ for $m \in [\tau \cdot D]$ and define the function

$$\begin{aligned} \vec{G}^{(j)}_{\vec{f},z}(\Delta_j) &= \sum_{i=0}^{D-1} \alpha_{j \cdot D+i}\, \vec{g}_{i,j}(b_{i,j}) \\ &= \sum_{i=0}^{D-1} \alpha_{j \cdot D+i} \left(\vec{f_j}^{[\![0]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + \Delta_j \cdot \left(\vec{f_j}, z_j\right), \end{aligned}$$

corresponding to each the subtree $T_j$, where $\Delta_j = \sum_{i=0}^{D-1} \alpha_{j \cdot D+i}\, b_{i,j}$. We can write this function as

$$\vec{G}^{(j)}_{\vec{f},z}(\Delta_j) = \vec{A}^{(j)}_{G_{\vec{f},z}} + \Delta_j \cdot (\vec{f_j}, z_j), \tag{6.2}$$

where $\vec{A}^{(j)}_{G_{\vec{f},z}} = \vec{G}^{(j)}_{\vec{f},z}(0)$. Note that each $\vec{G}^{(j)}_{\vec{f},z}$ is an affine function known by the prover, while the verifier chooses the evaluation point $\Delta_j^*$ and learns the value $\vec{G}^{(j)}_{\vec{f},z}(\Delta_j^*)$ using the PPRF key. After the prover has committed to the GGM tree family, the verifier computes a random vector $\vec{\mu} \in (\mathbb{F}_{p^l})^{|\mathcal{F}|}$ and sends it to the prover. The consistency check relies on the following Lemma:

**Lemma 6.6.** Let $\vec{f}, \vec{f'} \in (\mathbb{F}_p)^{|\mathcal{F}|}$ and $z, z' \in \mathbb{F}_{p^l}$ with $(\vec{f}, z) \neq (\vec{f'}, z')$. Let $\vec{\mu} \xleftarrow{\$} (\mathbb{F}_{p^l})^{|\mathcal{F}|}$. Then $\Pr\left[z + \vec{\mu} \cdot \vec{f} = z' + \vec{\mu} \cdot \vec{f'}\right] \leq p^{-l}$, where the probability is on the choice of $\vec{\mu}$.

*Proof.* The proof is analogous to the proof of Lemma 5.4. □

Multiplying (6.2) by $(\vec{\mu}, 1)$, we obtain

$$\begin{aligned} \vec{G}^{(j)}_{\vec{f},z}(\Delta_j) \cdot (\vec{\mu}, 1) &= \vec{A}^{(j)}_{G_{\vec{f},z}} \cdot (\vec{\mu}, 1) + \Delta_j (\vec{f_j}, z_j) \cdot (\vec{\mu}, 1) \\ &= \vec{A}^{(j)}_{G_{\vec{f},z}} \cdot (\vec{\mu}, 1) + \Delta_j [z_j + \vec{f_j} \cdot \vec{\mu}] \end{aligned}$$

Note that, if all $(\vec{f_j}, z_j)$ are indeed equal for all rounds, then the values $z_j + \vec{f_j} \cdot \vec{\mu}$ are identical for all $j \in [\tau]$. Therefore, the prover responds by sending all of the coefficients

$a^{(j)} = \vec{A}^{(j)}_{G_{\vec{f},z}} \cdot (\vec{\mu}, 1) \in \mathbb{F}_{p^l}$ for $j \in [\tau]$ and just one coefficient $c = z + \vec{f} \cdot \vec{\mu}$ to the verifier. With the PPRF key, the verifier can recompute $\Delta^*_j$ and recompute the values of $\vec{G}^{(j)}_{\vec{f},z}(\Delta^*_j)$ for $j \in [\tau]$. Finally, the verifier checks if

$$\vec{G}^{(j)}_{\vec{f},z}(\Delta^*_j) \cdot (\vec{\mu}, 1) = a^{(j)} + c\Delta^*_j$$

for all $j \in [\tau]$. If this consistency check succeeds, the verifier proceeds with the protocol, otherwise the verifier rejects. As seen in Theorem 5.5, the false positive rate of this consistency check is bounded by $\tau^2 p^{-l}$.
We provide the pseudocode for the Fherret scheme including the consistency check. In Figure 6.2, we describe the adapted Commitment scheme including the consistency check. In Figure 6.3, we describe the respective Challenge and Response algorithms between prover and verifier. In Figure 6.4, we provide the non-interactive version of theses algorithms by using the random oracle $\mathcal{H}$ with proper domain separation to derive the challenges. The Verification algorithm including the consistency check is given in Figure 6.5.

**Remark 6.7** (Using AES in the tree derivation)**.** As explained above, using internal nodes of just $\lambda$ bits reduces the size of the PPRF key of our scheme compared to a version using big internal nodes of size $|\mathcal{F}|$. For $\lambda = 128$, using internal nodes of size $\lambda$ bits has another advantage: We can use an internal tree derivation function based on the AES block cipher instead of using a (salted) hash function, which is a technique introduced in [BCC$^+$24b]. This allows us to take advantage of the AES instruction set used in many recent CPU architectures, resulting in an improvement of the running time of our scheme. For the adapted tree derivation based on AES, we need two AES keys $K_0$ and $K_1$ to derive the children of a node on a given level of the tree in the correlated GGM tree setting [GYW$^+$23]. The left child of a node $Y$ is set to $\mathrm{AES}_{K_0}(Y) \oplus \mathrm{AES}_{K_1}(Y)$, while the right child is given by $Y \oplus \mathrm{AES}_{K_0}(Y) \oplus \mathrm{AES}_{K_1}(Y)$ to achieve a XOR preserving construction. Instead of sending the salt of size $2\lambda$ bits, we send the two AES keys $K_0, K_1$ of size $\lambda$ bits each in this version. Therefore, the communication cost remains the same.

### 6.3.4 A Secure Version of the Scheme

In this section, we prove the security of the Fherret scheme.
**Prover privacy.** In many works on verifiable computation (VC) over fully homomorphic encryption (FHE), circuit privacy has been modeled as a context-hiding property of the VC scheme [BCFK21, GNS23]. While this model is useful for describing security when applying zk-SNARKs in the ciphertext space, recent VC works that operate in the plaintext space [ACGS24] have introduced the notion of honest-verifier prover privacy (HVPP) to describe the security of the prover's function.
We adapt this definition to prove that, any information a semi-honest verifier $\mathcal{V}$ can compute by participating in the protocol, $\mathcal{V}$ could compute using only its input and prescribed output. To show this we provide a simulator $\mathcal{S}$ that is able to produce an output computationally indistinguishable from a transcript of the real protocol using only elements that are already known to $\mathcal{V}$. We remark that considering a semi-honest verifier is in line with the majority of the works on circuit privacy.[2]

[2]This definition (and the security of our scheme) can be adapted to the malicious verifier setting by considering malicious circuit private FHE schemes [DD22, OPP14].

**Inputs**:

- Public key $\mathsf{pk}$, function $f$, vector of ciphertexts $\mathsf{ct}$ (Prover)
- Secret key $\mathsf{sk}$, public key $\mathsf{pk}$, vector of ciphertexts $\mathsf{ct}$ (Verifier)

**Step 1: Commitment (Prover)**

1. Let $f$ be a polynomial in $\mathcal{F}$ of degree $d$ with $k$ variables
2. For $(i,j) \in [D] \times [\tau]$ : Sample random coins $\sigma_{i,j}$ for the FHE `Eval` and compute $h^{\sigma}_{i,j} \leftarrow \mathcal{H}(i \parallel j \parallel \sigma_{i,j})$
3. Compute $h^{\sigma} \leftarrow \mathcal{H}\left(\left(h^{\sigma}_{i,j}\right)_{i,j \in [D] \times [\tau]}\right)$
4. Sample salt $\leftarrow_{\$} \{0,1\}^{2\lambda}$ for the GGM tree derivation
5. For $j \in [\tau]$:
   - Sample $\vec{u}_j \leftarrow_{\$} \{0,1\}^{\lambda}$
   - Compute $T_{\vec{u}_j}$: $\left(\vec{u}_j^{[\![0]\!]}, \ldots, \vec{u}_j^{[\![N-1]\!]}\right) \leftarrow \mathsf{GGM}_{\text{salt}}\left(\vec{u}_j, N\right)$
   - For $i \in [N]$: Compute $\vec{f}_j^{[\![i]\!]} \leftarrow \mathrm{PRG}_0\left(\vec{u}_j^{[\![i]\!]}\right)$ and ${z_j}^{[\![i]\!]} \leftarrow \mathrm{PRG}_1\left(\vec{u}_j^{[\![i]\!]}\right)$
   - $\left(\vec{f}_j^{[\![0]\!]_i}, \vec{f}_j^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{f}_j^{[\![0]\!]}, \ldots, \vec{f}_j^{[\![N-1]\!]}\right)$
   - $\left({z_j}^{[\![0]\!]_i}, {z_j}^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left({z_j}^{[\![0]\!]}, \ldots, {z_j}^{[\![N-1]\!]}\right)$
6. Compute $z_0 \leftarrow \sum_{i=0}^{N-1} {z_0}^{[\![i]\!]}$
7. For $j \in [\tau]$:
   - Compute $\delta_{\vec{f}_j} \leftarrow \vec{f} - \sum_{i=0}^{N-1} \vec{f}_j^{[\![i]\!]}$ and $\delta_{z_j} = z_0 - \sum_{i=0}^{N-1} {z_j}^{[\![i]\!]}$
   - For $i \in [D]$:
     - Let ${f_j}^{[\![0]\!]_i} \leftarrow \vec{f}_j^{[\![0]\!]_i}$ and ${f_j}^{[\![1]\!]_i} \leftarrow \vec{f}_j^{[\![1]\!]_i} + \delta_{\vec{f}_j}$
     - $\mathsf{ct}^{(0)}_{i,j} \leftarrow \mathtt{Eval}\left(\sigma_{i,j}; \mathsf{pk}, {f_j}^{[\![0]\!]_i}, \mathsf{ct}\right)$
     - $\mathsf{ct}^{(1)}_{i,j} \leftarrow \mathtt{Eval}\left(\sigma_{i,j}; \mathsf{pk}, {f_j}^{[\![1]\!]_i}, \mathsf{ct}\right)$
     - Compute $h^{(0)}_{i,j} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}^{(0)}_{i,j}\right)$ and $h^{(1)}_{i,j} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}^{(1)}_{i,j}\right)$
     - Compute $X_{i,j} \leftarrow \mathsf{ct}^{(0)}_{i,j} \oplus \mathsf{ct}^{(1)}_{i,j}$
8. Compute $\mathsf{com} \leftarrow \mathcal{H}\left(\text{salt} \parallel h^{\sigma} \parallel \left(h^{(0)}_{i,j}, h^{(1)}_{i,j}\right)_{(i,j) \in [D] \times [\tau]} \parallel \left(\delta_{\vec{f}_j}\right)_{j \in [\tau]} \parallel \left(\delta_{z_j}\right)_{j \in [\tau]^*}\right)$
9. Send commit $\leftarrow \left(\mathsf{com}, \text{salt}, h^{\sigma}, \left(X_{i,j}\right)_{(i,j) \in [D] \times [\tau]}, \left(\delta_{\vec{f}_j}\right)_{j \in [\tau]}, \left(\delta_{z_j}\right)_{j \in [\tau]^*}\right)$ to the verifier

**Figure 6.2:** Commitment of the hypercube Fherret scheme including the consistency check.

**Definition 6.8** (Honest-verifier prover-privacy (HVPP), adapted from [ACGS24])**.** A protocol is said to satisfy honest-verifier prover privacy if there exists a probabilistic polynomial-time (PPT) simulator $\mathcal{S}$ such that for every function $f \in \mathcal{F}$, the following distributions are computationally indistinguishable:

$$\{\mathsf{sk}, \mathcal{S}(\lambda, \mathsf{pk}, m, \mathsf{ct}, f(m))\} \approx_{\mathsf{c}} \{sk, \mathbf{View}_{\mathcal{V}}(\lambda, \mathsf{sk}, \mathsf{pk}, m, \mathsf{ct}, f)\},$$

**Step 2: First Challenge (Verifier)**

1. Sample $\vec{\mu} \xleftarrow{\$} (\mathbb{F}_{p^l})^{|\mathcal{F}|}$
2. For $m \in [\tau \cdot D]$: Sample $\alpha_m \xleftarrow{\$} \mathbb{F}_{p^l}$
3. Send $\vec{\mu}, (\alpha_m)_{m \in [\tau \cdot D]}$ to the prover

**Step 3: First Response (Prover)**

1. Compute $c \leftarrow z_0 + \vec{f} \cdot \vec{\mu}$
2. For $j \in [\tau]$:
   - Compute $\vec{A}^{(j)}_{G_{\vec{f}}} \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \, \vec{f}_j^{[\![0]\!]_i}$
   - Compute $\vec{A}^{(j)}_{G_z} \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \, z_j^{[\![0]\!]_i}$
   - Compute $a^{(j)} \leftarrow \vec{A}^{(j)}_{G_z} + \vec{A}^{(j)}_{G_{\vec{f}}} \cdot \vec{\mu}$
3. Send $c, \left(a^{(j)}\right)_{j \in [\tau]}$ to the verifier

**Step 4: Second Challenge (Verifier)**

1. For $(i, j) \in [D] \times [\tau]$: Compute $b^*_{i,j} \leftarrow_{\$} \{0, 1\}$
2. Send $\left(b^*_{i,j}\right)_{(i,j) \in [D] \times [\tau]}$ to the prover

**Step 5: Second Response (Prover)**

1. Compute the PPRF key $K \leftarrow K_{\left(b^*_{i,j}\right)_{(i,j) \in [D] \times [\tau]}}$
2. Send $K$, $(\sigma_{i,j})_{i,j}$ for all $(i, j) \in [D] \setminus \{i^*_0, \ldots, i^*_{\tau-1}\} \times [\tau]$ and the hashes $\left(h^{\sigma}_{i^*_j, j}\right)_{j \in [\tau]}$ to the verifier

**Figure 6.3:** Interactive version of the challenges and responses of Fherret.

where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(\lambda)$. $\mathbf{View}_{\mathcal{V}}$ denotes the view of the verifier $\mathcal{V}$ during an execution of the protocol on the client's input $(m, \mathsf{ct})$, where $\mathsf{ct} \leftarrow \mathtt{Enc}(\mathsf{sk}, m)$, on the server's input $f$ and the security parameter $\lambda$.

**Theorem 6.9.** The scheme from Figure 6.2 satisfies HVPP.

*Proof.* We write explicitly $\mathbf{View}_{\mathcal{V}}(\lambda, \mathsf{sk}, \mathsf{pk}, m, \mathsf{ct}, f)$ as

$$\left(\mathsf{pprf\_keys}, \mathsf{offsets}, (X_j)_{j \in [\tau]}, \mathsf{com}\right),$$

where

$$\mathsf{pprf\_keys} = \{(i^*_j)_{j \in [\tau]}, K_{i^*_{pre}}, K_{(i^*_0, \ldots, i^*_{\tau-1})}, \mathrm{salt}\},$$

and

$$\mathsf{offsets} = \{(\delta_f)_{j \in [\tau]}, (\delta_z)_{j \in [\tau]^*}, (a^{(j)})_{j \in [\tau]}, c\}.$$

We can construct $\mathcal{S}(\lambda, \mathsf{pk}, m, \mathsf{ct}, f(m))$ in the following way:

1. Sample $g \leftarrow_{\$} \mathcal{F}$ and compute $\bar{g} \leftarrow g + (f(m) - g(m))$.

**Step 2: First Challenge (Prover)**
Compute $\vec{\mu}, (\alpha_m)_{m\in[\tau\cdot D]} \leftarrow \mathcal{H}(\text{commit})$
**Step 3: First Response (Prover)**

1. Compute $c \leftarrow z_0 + \vec{f} \cdot \vec{\mu}$
2. For $j \in [\tau]$:
   - Compute $\vec{A}^{(j)}_{G_{\vec{f}}} \leftarrow \sum_{i=0}^{D-1} \alpha_{j\cdot D+i}\, \vec{f}_j^{[\![0]\!]_i}$
   - Compute $\vec{A}^{(j)}_{G_z} \leftarrow \sum_{i=0}^{D-1} \alpha_{j\cdot D+i}\, z_j^{[\![0]\!]_i}$
   - Compute $a^{(j)} \leftarrow \vec{A}^{(j)}_{G_z} + \vec{A}^{(j)}_{G_{\vec{f}}} \cdot \vec{\mu}$

**Step 4: Second Challenge (Prover)**
For $(i,j) \in [D] \times [\tau]$:
$b^*_{i,j} \leftarrow \mathcal{H}\left(\text{commit} \parallel \vec{\mu} \parallel (\alpha_m)_{m\in[\tau\cdot D]} \parallel c \parallel \left(a^{(j)}\right)_{j\in[\tau]}\right)$
**Step 5: Second Response (Prover)**

1. Compute the PPRF key $K \leftarrow K_{\left(b^*_{i,j}\right)_{(i,j)\in[D]\times[\tau]}}$
2. Send $K$, $c$, $\left(a^{(j)}\right)_{j\in[\tau]}$, the random coins $(\sigma_{i,j})_{i,j}$ for all $(i,j) \in [D]\backslash\{i^*_0, \ldots, i^*_{\tau-1}\}\times[\tau]$ and the hashes $\left(h^{\sigma}_{i^*_j,j}\right)_{j\in[\tau]}$ to the verifier

**Figure 6.4:** Non-interactive version of the challenges and responses of Fherret.

2. Follow honestly the protocol by using $\mathsf{pk}, \mathsf{ct}, \bar{g}$ up to step (6).
3. In step (7), looking at the second challenge $(b^*_{i,j})_{(i,j)\in[D]\times[\tau]}$, substitute the ciphertexts that are hidden to the verifier by using the simulator from circuit privacy

$$\mathsf{ct}_{i,j}^{b^*_{i,j}} \leftarrow \mathcal{S}_{\mathsf{CP}}(\mathsf{pk}, f_j^{[\![b^*_{i,j}]\!]_i}(m)).$$

4. Complete the rest of the protocol honestly.

Having a look at the outputs of the simulator we have that:

1. The pprf_keys and offsets behave like random elements in the ROM.
2. $(X_j)_{j\in[\tau]}$ and com are computationally indistinguishable from a real execution of the protocol because of the circuit privacy simulator.

□

**Verifier security.** To protect the verifier from malicious behavior, we must prove that an adversary cannot exploit the decryption or verification process to extract sensitive information. In particular, such attacks require the adversary to create a dependency between the verifier's output and private components, such as the verifier's secret key sk or the message $m$ encrypted in the ciphertext ct.
In cryptographic literature, reaction-based attacks are typically modeled using ideal oracles which provide abstract interfaces to the verifier's behavior. These oracles allow us to reason about what an attacker could learn under well-defined access patterns.

**Step 6: Verification and Decryption (Verifier)**

1. Use $K$ and salt to recompute and fold all trees (with one missing leaf each) and expand the leaves using $\mathrm{PRG}_0, \mathrm{PRG}_1$
2. For $(i,j) \in [D] \setminus \{i_0^*, \ldots, i_{\tau-1}^*\} \times [\tau]$: Compute $h_{i,j}^{\sigma} \leftarrow \mathcal{H}(i \parallel j \parallel \sigma_{i,j})$
3. Compute $h^{\sigma'} \leftarrow \mathcal{H}\left(\left(h_{i,j}^{\sigma}\right)_{i,j \in [D] \times [\tau]}\right)$
4. For $(i,j) \in [D] \times [\tau]$:
   - Denote the known bit position of the respective hypercube sharing of $\vec{f}_j^{[\![\cdot]\!]_i}$ by $b_{i,j} \leftarrow 1 - b_{i,j}^*$
   - Recompute ${f_j}^{[\![b_{i,j}]\!]_i} \leftarrow \vec{f}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\, \delta_{\vec{f}_j}$
   - $\mathsf{ct}_{i,j}^{(b_{i,j})} \leftarrow \mathtt{Eval}\left(\sigma_{i,j}; \mathsf{pk}, {f_j}^{[\![b_{i,j}]\!]_i}, \mathsf{ct}\right)$
   - $h_{i,j}^{(b_{i,j})} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}_{i,j}^{(b_{i,j})}\right)$
   - $\mathsf{ct}_{i,j}^{(1-b_{i,j})} \leftarrow \mathsf{ct}_{i,j}^{(1-b_{i,j})} \oplus X_{i,j}$
   - $h_{i,j}^{(1-b_{i,j})} \leftarrow \mathcal{H}\left(i \parallel j \parallel \mathsf{ct}_{i,j}^{(1-b_{i,j})}\right)$
5. Set $\delta_{z_0} \leftarrow 0$
6. For $j \in [\tau]$:
   - Compute $\Delta_j^* \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i}\, b_{i,j}$
   - Compute $\vec{G}_{\vec{f}}^{(j)}(\Delta_j^*) \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \left(\vec{f}_j^{[\![b_{i,j}]\!]} + b_{i,j}\, \delta_{\vec{f}_j}\right)$
   - Compute $\vec{G}_z^{(j)}(\Delta_j^*) \leftarrow \sum_{i=0}^{D-1} \alpha_{j \cdot D + i} \left(z_j{}^{[\![b_{i,j}]\!]} + b_{i,j}\, \delta_{z_j}\right)$
7. Compute $h' \leftarrow \mathcal{H}\left(\mathrm{salt} \parallel h^{\sigma'} \parallel \left(h_{i,j}^{(0)}, h_{i,j}^{(1)}\right)_{(i,j) \in [D] \times [\tau]} \parallel \left(\delta_{\vec{f}_j}\right)_{j \in [\tau]} \parallel \left(\delta_{z_j}\right)_{j \in [\tau]^*}\right)$
8. If $\vec{G}_z^{(j)}(\Delta_j^*) + \vec{G}_{\vec{f}}^{(j)}(\Delta_j^*) \cdot \vec{\mu} = a^{(j)} + c\, \Delta_j^*$ for all $j \in [\tau]$, $h^{\sigma'} = h^{\sigma}$ and $h' = \mathsf{com}$ output `accept`. Otherwise, output `reject`.
9. Compute $m_{\mathsf{out},i,j} \leftarrow \mathtt{Dec}\left(\mathsf{sk}, \mathsf{ct}_{i,j}^{(b_{i,j})}\right) + \mathtt{Dec}\left(\mathsf{sk}, \mathsf{ct}_{i,j}^{(1-b_{i,j})}\right)$
10. Output $m_{\mathsf{maj}} \leftarrow \mathsf{maj}\left(\{m_{\mathsf{out},i,j}\}_{(i,j) \in [D] \times [\tau]}\right)$

**Figure 6.5:** Verification of the hypercube Fherret scheme including the consistency check.

To establish the security of our scheme against such attacks, we show that the outputs of these oracles can be efficiently simulated using only public information and the expected output of the protocol. This implies that an adversary cannot obtain any additional useful information beyond what is provided as input to the simulator. To support these claims, we begin by proving the following theorem on the scheme from Figure 6.2.

**Theorem 6.10** (Function Commitment)**.** Let the verifier's output be not $\perp$. Then, in the Random Oracle Model (ROM), there exists a polynomial-time extractor that can recover the function $f \in \mathcal{F}$ used by the server with negligible failure probability. Furthermore, except with negligible probability, the output $m_{\mathsf{maj}}$ computed by the verifier satisfies:

$$m_{\mathsf{maj}} = f(m),$$

where $m$ is the message encrypted in the input ciphertext and $f$ is the extracted function.

*Proof.* Having a meaningful output means that the consistency check and the hash checks are successful. If the consistency check is successful, the function

$$f_j^{[\![0]\!]_i} + f_j^{[\![1]\!]_i} + \delta_{f_j}$$

is always the same for each $j \in [\tau]$ and $i \in [D]$, where the $f_j^{[\![b]\!]_i}$ are the functions reconstructed during verification from $\mathsf{pprf\_keys}$. Since each $f_j^{[\![b]\!]_i}$ is computed by using a PRG, that we model as a random oracle, the probability of constructing a working consistency proof compatible with $\mathsf{pprf\_keys}$ without querying the random oracle to obtain the $f_j^{[\![b]\!]_i}$ is negligible.
This means that we can recover all of the $f_j^{[\![b]\!]_i}$ by reading the random oracle memory. Now, since the hash checks verify, we know that the $\mathsf{offsets}$ sent by the verifier are the same ones used in the consistency check. Since this check also verifies we can extract the prover's function as the common value

$$f := f_j^{[\![0]\!]_i} + f_j^{[\![1]\!]_i} + \delta_{f_j}.$$

Finally, thanks to the hash checks, we know that all the ciphertexts queried in a challenge are computed correctly as

$$\mathsf{ct}_{i,j}^{b^*_{i,j}} \leftarrow \mathtt{Eval}(\sigma_{i,j}; \mathsf{pk}, f_j^{[\![b_{i,j}]\!]_i}, \mathsf{ct}),$$

therefore the probability of modifying the majority of these ciphertexts without the hash check failing is negligible.
This implies that at least half of the decrypted ciphertexts will correctly decrypt to $f(m)$. □

This theorem establishes two essential properties:

1. The prover is bound to a specific function $f \in \mathcal{F}$ in order for the verifier to produce a valid output.
2. The protocol's output consistently corresponds to the evaluation of $f$ on the original message $m$.

**Verifier security against $\mathsf{IND\text{-}CPA^D}$-style attacks and verification oracle attacks.** Correctness oracles reveal to the attacker if the result of the homomorphic evaluation of a function $f$ on a ciphertext encrypting $m$ gets decrypted to something different from $f(m)$. Verification oracles describe scenarios where the ciphertext returned by the prover is tampered with in a way that might change the verifier's output of the protocol.
We show that both of these oracles can be simulated from elements already available to the prover.

**Theorem 6.11.** Considering the protocol from Figure 6.2, there exist two probabilistic polynomial-time (PPT) simulators $\mathcal{S}_{\mathsf{Ver}}$ and $\mathcal{S}_{\mathsf{Corr}}$, such that

$$\mathcal{O}_{\mathsf{Ver}}(\mathsf{sk}, \mathcal{T}) \approx_{\mathsf{c}} \mathcal{S}_{\mathsf{Ver}}(\mathsf{pk}, \mathcal{T}) \quad \text{and} \quad \mathcal{O}_{\mathsf{Corr}}(\mathsf{sk}, \mathcal{T}) \approx_{\mathsf{c}} \mathcal{S}_{\mathsf{Corr}}(\mathsf{pk}, \mathcal{T}),$$

where we define the transcript of a protocol as

$$\mathcal{T} \leftarrow \mathbf{Tr}_{\mathcal{V}}(\mathsf{pk}, \mathsf{ct}, f),$$

and $\mathcal{O}_{\mathsf{Ver}}, \mathcal{O}_{\mathsf{Corr}}$ as the verification oracle and as the correctness oracle, respectively.

*Proof.* Deferred to the proof of Theorem 6.12. □

To illustrate this more concretely, we refer to the attack strategies in [CCP+24, CSBB24]. The adversary encrypts zero and then maliciously evaluates a function homomorphically. For example, the adversary homomorphically multiplies the ciphertext by a large power of two without bootstrapping, despite the FHE evaluation algorithm requiring it. This breaks correctness, yielding a nonzero result.
However, in our scheme, the verifier detects inconsistencies by comparing the expected ciphertexts from honest homomorphic evaluation against those produced by the adversary's modified evaluation. Therefore, the verifier rejects these ciphertexts without decrypting them. Thus, with these two oracles, an attacker gains no information about the secret key $\mathsf{sk}$ or the original message $m$.
**Verifier security against Decryption Oracle attacks.** Decryption oracles reveal the output of the protocol to the attacker, typically the decryption of the ciphertext returned by the prover, asking the attacker to infer additional information from this, such as attempting to recover the challenger's secret key $\mathsf{sk}$.
We show that we can describe the leakage deriving from a decryption oracle in our scheme in detail.

**Theorem 6.12.** Considering the protocol from Figure 6.2, there exists a probabilistic polynomial-time (PPT) simulator $\mathcal{S}_{\mathsf{Dec}}$, such that

$$\mathcal{O}_{\mathsf{Dec}}(\mathsf{sk}, \mathcal{T}) \approx_{\mathsf{c}} \mathcal{S}_{\mathsf{Dec}}(\mathsf{pk}, \mathcal{T}, f(m)),$$

where we define the transcript of a protocol as

$$\mathcal{T} \leftarrow \mathbf{Tr}_{\mathcal{V}}(\mathsf{pk}, \mathsf{ct}, f),$$

$f$ as the (extractable) function used by the prover, $m$ as the message encrypted in the input ciphertext from the verifier and $\mathcal{O}_{\mathsf{Dec}}$ as the decryption oracle.

*Proof.* We prove Theorem 6.11 and Theorem 6.12. First, we observe that the verification protocol (Figure 6.5) can be carried out almost entirely, specifically steps (1) to (8), without requiring access to sensitive elements from either the verifier or the prover. In particular, the verifier's secret key $\mathsf{sk}$, input message $m$, and the prover's function $f$ are not needed.
We refer to this partial verification, which performs only the consistency and hash checks, as $\mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T})$, where we define $\mathcal{T}$ as the transcript of the protocol

$$\mathcal{T} \leftarrow \mathbf{Tr}_{\mathcal{V}}(\mathsf{pk}, \mathsf{ct}, f).$$

We define the three oracles as follows:

$$\mathcal{S}_{\mathsf{Ver}}(\mathsf{pk}, \mathcal{T}) = \mathcal{S}_{\mathsf{Corr}}(\mathsf{pk}, \mathcal{T}) = \begin{cases} \mathsf{true} \text{ if } \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{true} \\ \mathsf{false} \text{ if } \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{false} \end{cases}$$

$$\mathcal{S}_{\mathsf{Dec}}(\mathsf{pk}, \mathcal{T}, f(m)) = \begin{cases} \mathsf{f(m)} \text{ if } \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{true} \\ \perp \quad \text{if } \mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{false} \end{cases}$$

Theorem 6.10 grants us that, except with negligible probability, the output of the protocol is $f(m)$ if $\mathcal{V}_{\mathsf{pub}}(\mathsf{pk}, \mathcal{T}) = \mathsf{true}$ and that it is $\perp$ otherwise. This means that these simulators will behave in the same way as the oracles they simulate. □

As expected in such an attack, this reveals $f(m)$ and provides some information about $m$. However, crucially, the output remains independent of the secret key $\mathsf{sk}$.

**Public verifiability.** We observe that the verification protocol (Figure 6.5) can be carried out almost entirely, specifically steps (1) to (8), without requiring access to sensitive elements from either the verifier or the prover. In particular, the verifier's secret key $\mathsf{sk}$, the input message $m$, and the prover's function $f$ are not needed.

This property allows the verifier to delegate the computationally expensive portion of the verification process to an external trusted party. The trusted party can independently verify the prover's honest behavior, compute the intermediate ciphertexts up to step (8), and then return these ciphertexts to the verifier. The verifier can then complete the protocol by simply decrypting the final ciphertexts.

## 6.4 Parameters

Fherret is not scheme-specific and can be applied to any (non-approximate) FHE scheme. While we make particular choices regarding schemes and parameters, these choices are not mandatory for implementing Fherret. In this proof of concept, we focus on estimating the runtime for both the prover and the verifier, and we discuss implementation-related optimizations.

### 6.4.1 Setting Fherret Parameters

Depending on the efficiency of the hash function used in the tree derivation, many MPCitH schemes can feasibly compute sharings for up to $N = 2^{16}$ parties within a reasonable computation time. To reduce the communication cost of Fherret, it is desirable to use a large number of parties $N$ and a small number of rounds $\tau$. In contrast, the computational overhead for both the prover and the verifier in constructing the trees is reduced when using a smaller number of parties $N$ and a larger number of rounds $\tau$.

To achieve a desired security level of $\lambda$ bits, we must choose $N$ and $\tau$ such that $\log(N) \cdot \frac{\tau}{2} \geq \lambda$. Therefore, depending on the desired trade-offs in efficiency and communication, the user can select these parameters accordingly.

Including the consistency check, we must choose parameters $N = 2^D$ and $\tau$ such that the overall security of the scheme is at least $\lambda$ bits.

By the union bound, the cheating probability of a malicious prover is upper bounded by $\frac{1}{N^{\tau/2}} + \tau^2 p^{-l}$, where $p$ is the plaintext modulus of the FHE scheme. For $\lambda = 128$, we can, for instance, set $l = \lceil 256/\log(p) \rceil$ to achieve a negligible failure probability in the consistency check.

The total communication cost of Fherret includes:

- a global commitment hash com of size $2\lambda$
- a salt of size $2\lambda$
- a commitment to the random coins of size $2\lambda$
- $\tau \cdot D = 2\lambda$ ciphertexts $\mathsf{ct}_{i,j}$
- $\tau$ offsets $\vec{f_j}$ and $(\tau - 1)$ offsets $z_j$ of size $\tau \cdot |f| + (\tau - 1) \cdot 2\lambda$

- the vector $\vec{\mu} \in (\mathbb{F}_{p^l})^{|\mathcal{F}|}$ that can be recomputed using a PRG on a seed of size $\lambda$
- the elements $(\alpha_m)_{m\in[\tau\cdot D]}$ of size $\tau \cdot D \cdot 2\lambda \approx 4\lambda^2$
- the vectors $\left(a^{(j)}\right)_{j\in[\tau]}$ of size $\approx \tau \cdot |f|$
- one element $c$ of size $2\lambda$
- a PPRF key of size $\tau \cdot \lambda D \approx 2\lambda^2$
- the random coins $\sigma_{i,j}$ of size $(\tau - 1) \cdot D \cdot \lambda \approx 2\lambda^2$
- the hashes $(h^{\sigma}_{i^*_j,j})_{j\in[\tau]}$ of size $\tau \cdot 2\lambda$

In all computations, we use that $\tau D \approx 2\lambda$. This yields a total bit communication size of

$$\text{size} \approx 2\tau \cdot |f| + 2\lambda \cdot |\mathsf{ct}| + 8\lambda^2 + 4\tau\lambda + 9\lambda.$$

A comparison of the resulting bit-security level and communication size for various values of $D$ and $\tau$ is provided in Table 6.1.

| $D$ | $\tau$ | bit security level | size (bits) |
|---|---|---|---|
| 15 | 18 | 135 | $270 \cdot \|\mathsf{ct}\| + 36 \cdot \|f\| + 156735$ |
| 14 | 19 | 133 | $266 \cdot \|\mathsf{ct}\| + 38 \cdot \|f\| + 152817$ |
| 13 | 20 | 130 | $260 \cdot \|\mathsf{ct}\| + 40 \cdot \|f\| + 146770$ |
| 12 | 22 | 132 | $264 \cdot \|\mathsf{ct}\| + 44 \cdot \|f\| + 152196$ |
| 11 | 24 | 132 | $264 \cdot \|\mathsf{ct}\| + 48 \cdot \|f\| + 153252$ |
| 10 | 26 | 130 | $260 \cdot \|\mathsf{ct}\| + 52 \cdot \|f\| + 149890$ |

**Table 6.1:** Examples of parameter choices for $D$ and $\tau$ and the resulting bit security and communication size of the Fherret protocol using $l = \lceil 2\lambda/\log(p) \rceil$.

### 6.4.2 Choosing $\mathcal{F}$

In the definition of circuit privacy, the choice of the function space from which the evaluated function is drawn is crucial. Most works on circuit privacy consider function spaces consisting of all polynomials with a fixed degree $d$ and a fixed number of variables $k$. We denote this space as $\mathcal{F}_{d,k}$.

In $\mathcal{F}_{d,k}$, each function contains $\binom{d+k+1}{k+1}$ monomials. Therefore, the size of its description is proportional to the number of monomials, multiplied by the bit length of the message space (that in our case is the bit size of the plaintext modulus).

In specific applications, the topology of the function (i.e., the structure of the monomials) may not need to remain hidden, and only the coefficients are considered sensitive. In such cases, the additive sharing can be performed over a significantly smaller function space $\mathcal{F}$. This allows the performance of homomorphic evaluation over a random function in $\mathcal{F}$ to better approximate the actual cost of evaluating the prover's function.

However, we stress that this optimization is not suitable for all scenarios. It can only be safely applied to function spaces that are large enough to resist attacks capable of recovering the function with a limited number of queries.

| | | Fherret (Prover) | | | | Fherret (Verifier) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | single core | | 128 par. threads | | single core | | 128 par. threads | |
| $\mathcal{F}_{d,k}$ | $\texttt{Eval}(f)$ | time | overhead | time | overhead | time | overhead | time | overhead |
| $\mathcal{F}_{2,4}$ | 0.18 s | 37.5 s | × 208 | 9.6 s | × 53 | 22.5 s | × 128 | 8.8 s | × 49 |
| $\mathcal{F}_{3,4}$ | 0.55 s | 93 s | × 169 | 17.7 s | × 32 | 51 s | × 93 | 15.6 s | × 28 |
| $\mathcal{F}_{4,4}$ | 1.42 s | 187 s | × 131 | 32.6 s | × 22 | 104 s | × 73 | 25.9 s | × 18 |
| $\mathcal{F}_{5,4}$ | 3.12 s | 356 s | × 114 | 58 s | × 19 | 189 s | × 61 | 43 s | × 14 |
| $\mathcal{F}_{6,4}$ | 12.33 s | 1639 s | × 133 | 135 s | × 11 | 714 s | × 58 | 92 s | × 7.5 |
| $\mathcal{F}_{8,4}$ | 91 s | – | – | 379 s | × 4.2 | – | – | 244 s | × 2.7 |
| $\mathcal{F}_{10,4}$ | 219 s | – | – | 879 s | × 4 | – | – | 553 s | × 2.5 |

**Table 6.2:** Computational overhead of Fherret compared to evaluating a random function from $\mathcal{F}_{d,k}$.

### 6.4.3 Improving the Timings for Repeated FHE Evaluations

In step (7) of the commitment phase (Figure 6.2) and step (4) of the verification phase (Figure 6.5), we require the homomorphic evaluation of a total of $4\lambda$ and $2\lambda$ random functions from $\mathcal{F}$, respectively.
We can significantly reduce the runtime of these evaluations by leveraging the structure of the function space $\mathcal{F}_{d,k}$. Specifically, each polynomial in $\mathcal{F}_{d,k}$ can be expressed as a sum of monomials of the form $x_1^{i_1} \cdots x_k^{i_k}$, for $0 \leq i_1 + \cdots + i_k \leq d$, each multiplied by its corresponding coefficient $c_{i_1,\ldots,i_k}$.
Crucially, the homomorphic evaluation of these monomials is independent of the coefficients and thus common to all the functions being evaluated. This allows us to compute all relevant monomials once and reuse them across all function evaluations.
This reuse strategy becomes especially impactful when $\mathcal{F}$ includes polynomials of high degree or a large number of variables. In such cases, amortizing the cost of monomial evaluations significantly reduces the overall overhead associated with repeated FHE evaluations.

### 6.4.4 Implementation and Running Times

We provide the running times of our scheme based on a proof-of-concept implementation in C++. As the underlying FHE scheme, we use the BGV-RNS scheme, as implemented in the OpenFHE library [BAB+22]. We use a plaintext modulus $p = 65537$ and perform a packing of 12 messages per ciphertext.
Currently, the major FHE libraries supporting BGV and BFV do not provide implementations that guarantee circuit privacy. Therefore, **our proof-of-concept implementation ensures security only for the verifier**. Nonetheless, since Fherret's overhead is proportional to the cost of FHE evaluations, the *relative* impact on performance would remain comparable even when circuit privacy is enforced.
Our implementation targets the $\lambda = 128$ bit security level, leveraging the optimizations discussed in Section 6.3.3. Among the various parameter choices listed in Table 6.1, the configuration $(D, \tau) = (13, 20)$ offered the best trade-off between tree generation and FHE evaluation costs. This choice resulted in the shortest overall running times for both the prover and the verifier.
In Table 6.3, we report the running times of Fherret for various choices of the function space $\mathcal{F}$. Specifically, we consider executions that guarantee circuit privacy over function

spaces $\mathcal{F}_{d,k}$, defined by polynomials of fixed depth $d$ and a fixed number of variables $k$.

| | Commitment (Prover) | | | | | | Verification (Verifier) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MPCitH | | | FHE | | total | MPCitH | | | FHE | | total |
| $\mathcal{F}_{d,k}$ | GGM | PRG + Fold | $\mathcal{H}$ | Monomials | Coeff | | GGM | PRG + Fold | $\mathcal{H}$ | Monomials | Coeff | |
| Single core | | | | | | | | | | | | |
| $\mathcal{F}_{2,4}$ | 0.4 s | 3.2 s | 4.2 s | 0.12 s | 29.4 s | 37.5 s | 0.4 s | 3.0 s | 4.2 s | 0.12 s | 14.7 s | 22.5 s |
| $\mathcal{F}_{2,10}$ | 0.4 s | 15.7 s | 4.3 s | 0.67 s | 130 s | 151 s | 0.4 s | 14.9 s | 4.4 s | 0.67 s | 65 s | 86 s |
| $\mathcal{F}_{2,20}$ | 0.4 s | 67 s | 4.6 s | 2.5 s | 458 s | 534 s | 0.4 s | 49 s | 4.7 s | 2.6 s | 235 s | 291 s |
| $\mathcal{F}_{3,4}$ | 0.4 s | 7.5 s | 4.4 s | 0.4 s | 80 s | 93 s | 0.4 s | 7.8 s | 4.5 s | 0.4 s | 38 s | 51 s |
| $\mathcal{F}_{4,4}$ | 0.4 s | 14.6 s | 4.4 s | 1.1 s | 166 s | 187 s | 0.4 s | 13.9 s | 4.4 s | 1.1 s | 84 s | 104 s |
| $\mathcal{F}_{5,4}$ | 0.4 s | 24.7 s | 4.5 s | 2.5 s | 323 s | 356 s | 0.4 s | 22.6 s | 4.4 s | 2.5 s | 160 s | 189 s |
| $\mathcal{F}_{6,4}$ | 0.4 s | 41.6 s | 8.6 s | 9.3 s | 1576 s | 1639 s | 0.4 s | 34.4 s | 8.6 s | 9.3 s | 659 s | 714 s |
| Using 128 threads in parallel for the FHE evaluation | | | | | | | | | | | | |
| $\mathcal{F}_{2,4}$ | 0.4 s | 3.2 s | 4.2 s | 0.12 s | 1.8 s | 9.6 s | 0.4 s | 3.0 s | 4.2 s | 0.12 s | 0.93 s | 8.8 s |
| $\mathcal{F}_{2,10}$ | 0.4 s | 15.7 s | 4.3 s | 0.67 s | 7.1 s | 28 s | 0.4 s | 14.9 s | 4.4 s | 0.67 s | 3.7 s | 25 s |
| $\mathcal{F}_{2,20}$ | 0.4 s | 67 s | 4.6 s | 2.5 s | 24.4 s | 103 s | 0.4 s | 49 s | 4.7 s | 2.6 s | 12.4 s | 72 s |
| $\mathcal{F}_{3,4}$ | 0.4 s | 7.5 s | 4.4 s | 0.4 s | 4.9 s | 17.7 s | 0.4 s | 7.8 s | 4.5 s | 0.4 s | 2.5 s | 15.6 s |
| $\mathcal{F}_{4,4}$ | 0.4 s | 14.6 s | 4.4 s | 1.1 s | 11.7 s | 32.6 s | 0.4 s | 13.7 s | 4.4 s | 1.1 s | 6.0 s | 25.9 s |
| $\mathcal{F}_{5,4}$ | 0.4 s | 24.7 s | 4.5 s | 2.5 s | 24.9 s | 58 s | 0.4 s | 22.6 s | 4.4 s | 2.5 s | 12.3 s | 43 s |
| $\mathcal{F}_{6,4}$ | 0.4 s | 41.6 s | 8.6 s | 9.3 s | 74 s | 135 s | 0.4 s | 34.4 s | 8.6 s | 9.3 s | 37 s | 92 s |
| $\mathcal{F}_{8,4}$ | 0.4 s | 94 s | 8.7 s | 31 s | 239 s | 379 s | 0.4 s | 75 s | 9.0 s | 31 s | 123 s | 244 s |
| $\mathcal{F}_{10,4}$ | 0.4 s | 200 s | 9.0 s | 72 s | 587 s | 879 s | 0.4 s | 159 s | 9.3 s | 71 s | 303 s | 553 s |

**Table 6.3:** Running times of Fherret for $\lambda = 130$ and $(D, \tau) = (13, 20)$ using twin AMD EPYC 9374F processors running at 3.85 GHz.

The table includes partial timings to illustrate the asymptotic scaling behavior of different components of the protocol. Notably, the tree derivation process depends only on the security parameter and is independent of $\mathcal{F}$. In contrast, the PRG expansion and hypercube folding steps scale linearly with the number of monomials in $\mathcal{F}$. The column labeled $\mathcal{H}$ aggregates the timings for all steps occurring after the FHE evaluations; this includes hashing, the generation (or verification) of the VOLE consistency check, and the construction of the PPRF key. Among these, the hashing time is the dominant factor and also scales linearly with the number of monomials.
We also break down the FHE evaluation timings into two components: the evaluation of monomials and the evaluation of the $4\lambda$ sets of coefficients ($2\lambda$ for the verifier). By examining single-core timings, we quantify the impact of the optimization introduced in Section 6.4.3. Without this optimization, monomial evaluation would be redundantly performed 520 times by the prover and 260 times by the verifier. By sharing monomial computations across evaluations, we reduce the total homomorphic evaluation time by approximately 67% to 80%, as shown in Table 6.2. We see that the savings increase as the size of $\mathcal{F}$ grows. We also report timings using 128 parallel threads for FHE evaluations, demonstrating the scalability of Fherret. This parallelization enables efficient evaluation even for functions with large depth or high number of variables. For our examples, the prover overhead reaches as low as 114 times the cost of evaluation a random function, while the verifier overhead has a factor as low as 58. These factors drastically improve when using parallelization (a factor as low as 4 for the prover and a factor as low as 2.5 for the verifier).

CHAPTER 7

# Conclusion

This dissertation explored new directions in the design of post-quantum cryptographic schemes within the MPCitH and VOLEitH paradigms, grounded in a new hardness assumption called Subfield Bilinear Collision (SBC) problem. We developed a theoretical and practical framework around this assumption to demonstrate its application to efficient, compact, and conceptually simple post-quantum secure digital signature schemes, while also extending the same principles to the domain of Fully Homomorphic Encryption.
One central observation that is evident throughout this dissertation is the fact that the SBC problem fits the structure of MPCitH and VOLEitH protocols exceptionally well. The fact that we can test for solutions of SBC using linear polynomials enables straightforward computation and verification in the multiparty setting, resulting in small computational overhead and compact signature sizes. Moreover, SBC itself is a win-win assumption: Current state-of-the-art attacks against SBC are not feasible in practice. If, on the other hand, there are new attacks that efficiently break the SBC problem, these attacks can be used to improve the state-of-the-art techniques for computing discrete logarithms in small characteristic finite fields.
Beyond the hardness assumption itself, this dissertation introduced several generic techniques that improve the efficiency of MPCitH and VOLEitH schemes beyond the specific context of SBC. We presented a fast hypercube folding algorithm with linear running time $\mathcal{O}(N)$ in the number of parties $N$. Furthermore, we introduced the correlated GGM forest, a layered construction using the cGGM tree derivation and the hypercube folding technique to correlate multiple cGGM trees across several rounds of the protocol. By using the cGGM forest, we reduce the communication cost of the scheme by removing the first $\lambda$ bits from the required offset values that are sent as part of the signature.
Finally, we introduced several variants of MinMax commitments. Using these, we can commit to the cGGM forest in an implicit way, resulting in a further improvement of the signature size. Together, these techniques form a cohesive toolkit for building lighter and faster post-quantum signature schemes within the MPCitH and VOLEitH paradigms. Importantly, these optimizations are independent of the SBC problem itself, making them applicable to a wide range of other assumptions and constructions.
By extending MPCitH and VOLEitH concepts to the FHE setting, we developed Fherret, a proof system for correct-and-honest homomorphic evaluation when using circuit privacy. Fherret demonstrates that ideas originally designed for zero-knowledge and signature protocols can also strengthen privacy-preserving computation, offering protection against reaction-based attacks with significantly lower overhead compared to zk-SNARK-based solutions.

# Bibliography


[AAA+24] Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Isaac Andrés Canales Martínez, Jorge Chávez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan Komada Eriksen, Tako Boris Fouotsa, Décio Luiz Gazzoni Filho, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez Henríquez, Sina Schaeffler, and Benjamin Wesolowski. SQIsign. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[AAB+24] Gor Adj, Nicolas Aragon, Stefano Barbero, Magali Bardet, Emmanuele Bellini, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Andre Esser, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, Matthieu Rivain, Luis Rivera-Zamarripa, Carlo Sanna, Jean-Pierre Tillich, Javier Verbel, and Floyd Zweydinger. Mirath (merger of MIRA/MiRitH). Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[ABB+23] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, Matthieu Rivain, and Jean-Pierre Tillich. MIRA. Technical report, National Institute of Standards and Technology, 2023. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`.

[ABB+24a] Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Victor Dyseryn, Andre Esser, Thibauld Feneuil, Philippe Gaborit, Mukul Kulkarni, Victor Mateu, Marco Palumbi, Lucas Perin, Matthieu Rivain, Jean-Pierre Tillich, and Keita Xagawa. PERK. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[ABB+24b] Carlos Aguilar Melchor, Slim Bettaieb, Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Nicolas Gama, Shay Gueron, James Howe, Andreas Hülsing, David Joseph, Antoine Joux, Mukul Kulkarni, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. SDitH — Syndrome Decoding in the Head. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[ABB+24c] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Romaric Neveu, Matthieu Rivain, Jean-Pierre Tillich, and Adrien Vinçotte.

RYDE. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[ABPS24] Shahla Atapoor, Karim Baghery, Hilder V. L. Pereira, and Jannik Spiessens. Verifiable FHE via Lattice-based SNARKs. *IACR Communications in Cryptology*, 1(1), 2024.

[ACGS24] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part V*, volume 15488 of *LNCS*, pages 302–334. Springer, Singapore, December 2024.

[AFI+24] Rika Akiyama, Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Koha Kinjo, Haruhisa Kosuge, Satoshi Nakamura, Shingo Orihara, Tsuyoshi Takagi, and Kimihiro Yamakoshi. QR-UOV. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[AGH+23] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the SDitH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 564–596. Springer, Cham, April 2023.

[AGHV22] Adi Akavia, Craig Gentry, Shai Halevi, and Margarita Vald. Achievable CCA2 relaxation for homomorphic encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 70–99. Springer, Cham, November 2022.

[Ame05] American National Standards Institute, Inc. ANSI X9.62 public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA), November 16, 2005.

[ARZV22] Gora Adj, Luis Rivera-Zamarripa, and Javier Verbel. MinRank in the head: Short signatures from zero-knowledge proofs. Cryptology ePrint Archive, Report 2022/1501, 2022.

[AV21] Adi Akavia and Margarita Vald. On the privacy of protocols based on CPA-secure homomorphic encryption. Cryptology ePrint Archive, Report 2021/803, 2021.

[BAB+22] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R. V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915, 2022.

[BBB+20] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. *An Algebraic Attack on Rank Metric Code-Based Cryptosystems*, page 64–93. Springer International Publishing, 2020.

[BBB+24a] Marco Baldi, Alessandro Barenghi, Michele Battagliola, Sebastian Bitzer, Marco Gianvecchio, Patrick Karl, Felice Manganiello, Alessio Pavoni, Gerardo Pelosi, Paolo Santini, Jonas Schupp, Edoardo Signorini, Freeman Slaughter, Antonia Wachter-Zeh, and Violetta Weger. CROSS — Codes and Restricted Objects Signature Scheme. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[BBB+24b] Marco Baldi, Alessandro Barenghi, Luke Beckwith, Jean-François Biasse, Tung Chou, Andre Esser, Kris Gaj, Patrick Karl, Kamyar Mohajerani, Gerardo Pelosi, Edoardo Persichetti, Markku-Juhani O. Saarinen, Paolo Santini, Robert Wallace, and Floyd Zweydinger. LESS — Linear Equivalence Signature Scheme. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[BBB+24c] Carsten Baum, Lennart Braun, Ward Beullens, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[BBC+20] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier A. Verbel. Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 507–536. Springer, Cham, December 2020.

[BBD+23] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.

[BBD+24] Joppe W. Bos, Olivier Bronchain, Léo Ducas, Serge Fehr, Yu-Hsuan Huang, Thomas Pornin, Eamonn W. Postlethwaite, Thomas Prest, Ludo N. Pulles, and Wessel van Woerden. HAWK. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[BBFR24] Ryad Benadjila, Charles Bouillaguet, Thibauld Feneuil, and Matthieu Rivain. MQOM — MQ on my Mind. Technical report, National Institute

of Standards and Technology, 2024. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures.

[BBGK24] Slim Bettaieb, Loïc Bidoux, Philippe Gaborit, and Mukul Kulkarni. Modelings for generic PoK and Applications: Shorter SD and PKP based Signatures. Cryptology ePrint Archive, Paper 2024/1668, 2024.

[BCC+24a] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. MAYO. Technical report, National Institute of Standards and Technology, 2024. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures.

[BCC+24b] Dung Bui, Eliana Carozza, Geoffroy Couteau, Dahmun Goudarzi, and Antoine Joux. Faster signatures from MPC-in-the-head. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 396–428. Springer, Singapore, December 2024.

[BCC+24c] Dung Bui, Eliana Carozza, Geoffroy Couteau, Dahmun Goudarzi, and Antoine Joux. Short Signatures from Regular Syndrome Decoding, Revisited. Cryptology ePrint Archive, Paper 2024/252, 2024.

[BCD+24] Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang. UOV — Unbalanced Oil and Vinegar. Technical report, National Institute of Standards and Technology, 2024. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures.

[BCFK21] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 528–558. Springer, Cham, May 2021.

[BDK+21] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 266–297. Springer, Cham, May 2021.

[BFG+24a] Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, and Matthieu Rivain. Dual support decomposition in the head: Shorter signatures from rank SD and MinRank. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part II*, volume 15485 of *LNCS*, pages 38–69. Springer, Singapore, December 2024.

[BFG+24b] Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, and Matthieu Rivain. Dual support decomposition in the head: Shorter signatures from rank SD and MinRank. Cryptology ePrint Archive, Report 2024/541, 2024.

[BFJT09] Charles Bouillaguet, Pierre-Alain Fouque, Antoine Joux, and Joana Treger. A family of weak keys in HFE (and the corresponding practical key-recovery). Cryptology ePrint Archive, Report 2009/619, 2009.

[BFR24] Ryad Benadjila, Thibauld Feneuil, and Matthieu Rivain. MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. In *2024 IEEE European Symposium on Security and Privacy*, pages 468–485. IEEE Computer Society Press, July 2024.

[BFS96] Jonathan F. Buss, Gudmund Skovbjerg Frandsen, and Jeffery O. Shallit. The Computational Complexity of Some Problems of Linear Algebra. *BRICS Report Series*, 3(33), June 1996.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

[BKPV23] Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier Verbel. Biscuit. Technical report, National Institute of Standards and Technology, 2023. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[Bui24] Dung Bui. Shorter VOLEitH signature from multivariate quadratic. Cryptology ePrint Archive, Report 2024/465, 2024.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Berlin, Heidelberg, December 2013.

[CCC+25] Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Verifiable Computation for Approximate Homomorphic Encryption Schemes. Cryptology ePrint Archive, Paper 2025/286, 2025.

[CCCM22] Bhuvnesh Chaturvedi, Anirban Chakraborty, Ayantika Chatterjee, and Debdeep Mukhopadhyay. A practical full key recovery attack on TFHE and FHEW by inducing decryption errors. Cryptology ePrint Archive, Report 2022/1563, 2022.

[CCP+24] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the IND-CPA$^{\mathrm{D}}$ security of exact FHE schemes. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2505–2519. ACM Press, October 2024.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, Berlin, Heidelberg, May 2013.

[CGG16] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. Attacking FHE-based applications by software fault injections. Cryptology ePrint Archive, Report 2016/1164, 2016.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

[CKP+24] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. VERITAS: Plaintext encoders for practical verifiable homomorphic encryption. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2520–2534. ACM Press, October 2024.

[CLY+24a] Hongrui Cui, Hanlin Liu, Di Yan, Kang Yang, Yu Yu, and Kaiyi Zhang. ReSolveD: Shorter signatures from regular syndrome decoding and VOLE-in-the-head. Cryptology ePrint Archive, Report 2024/040, 2024.

[CLY+24b] Hongrui Cui, Hanlin Liu, Di Yan, Kang Yang, Yu Yu, and Kaiyi Zhang. ReSolveD: Shorter signatures from regular syndrome decoding and VOLE-in-the-head. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part I*, volume 14601 of *LNCS*, pages 229–258. Springer, Cham, April 2024.

[CSBB24] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical CPA$^D$ security of "exact" and threshold FHE schemes and libraries. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 3–33. Springer, Cham, August 2024.

[DD22] Nico Döttling and Jesko Dujmovic. Maliciously circuit-private FHE from information-theoretic principles. In Dana Dachman-Soled, editor, *ITC 2022*, volume 230 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl, July 2022.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Berlin, Heidelberg, April 2015.

[DPS20] Jintai Ding, Albrecht Petzoldt, and Dieter S. Schmidt. *Multivariate Public Key Cryptosystems*. Springer US, 2020.

[DSA13] Angsuman Das, Dutta Sabyasachi, and Adhikari Avishek. Indistinguishability against chosen ciphertext verification attack revisited: The complete picture. *Provable Security: 7th International Conference, ProvSec*, 2013.

[Fau02] Jean Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83, New York, NY, USA, 2002. ACM.

[Fen22] Thibauld Feneuil. Building MPCitH-based signatures from MQ, MinRank, rank SD and PKP. Cryptology ePrint Archive, Report 2022/1512, 2022.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 844–855. ACM Press, November 2014.

[FJR22] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 541–572. Springer, Cham, August 2022.

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 124–154. Springer, Cham, May 2020.

[FR23] Thibauld Feneuil and Matthieu Rivain. MQOM — MQ on my Mind. Technical report, National Institute of Standards and Technology, 2023. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

[FSS11] Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1,1): Algorithms and complexity. *Journal of Symbolic Computation*, 46(4):406–437, 2011.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[GBK+24] Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. Blind zkSNARKs for private proof delegation and verifiable computation over encrypted data. Cryptology ePrint Archive, Report 2024/1684, 2024.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GGW24] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 449–487. Springer, Cham, August 2024.

[GNS21] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. Cryptology ePrint Archive, Report 2021/322, 2021.

[GNS23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. *Journal of Cryptology*, 36(4):41, October 2023.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques.* Cambridge University Press, 2001.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC*, pages 212–219. ACM Press, May 1996.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Berlin, Heidelberg, December 2013.

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 330–362. Springer, Cham, April 2023.

[HJ23] Janik Huth and Antoine Joux. MPC in the head using the subfield bilinear collision problem. Cryptology ePrint Archive, Report 2023/1685, 2023.

[HJ24a] Janik Huth and Antoine Joux. MPC in the head using the subfield bilinear collision problem. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 39–70. Springer, Cham, August 2024.

[HJ24b] Janik Huth and Antoine Joux. VOLE-in-the-head signatures from Subfield Bilinear Collisions. Cryptology ePrint Archive, Paper 2024/1537, 2024.

[HJ25] Janik Huth and Antoine Joux. Vole-in-the-Head Signatures from Subfield Bilinear Collisions. In *ASIACRYPT 2025*. Springer-Verlag, 2025.

[HJS25] Janik Huth, Antoine Joux, and Giacomo Santato. Fherret: Proof of FHE Correct-and-Honest Evaluation with Circuit Privacy from MPCitH. Cryptology ePrint Archive, Paper 2025/700, 2025.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[Jou14] Antoine Joux. A new index calculus algorithm with complexity $L(1/4+o(1))$ in small characteristic. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 355–379. Springer, Berlin, Heidelberg, August 2014.

[JP22] Antoine Joux, , and Cécile Pierrot. Algorithmic aspects of elliptic bases in finite field discrete logarithm algorithms. *Advances in Mathematics of Communications*, 0(0):0–0, 2022.

[JV11] Antoine Joux and Vanessa Vitse. Cover and decomposition index calculus on elliptic curves made practical. application to a seemingly secure curve over $\mathbb{F}_{p^6}$. Cryptology ePrint Archive, Report 2011/020, 2011.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. Cryptology ePrint Archive, Report 2018/475, 2018.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[KLS25] Seongkwang Kim, Byeonghak Lee, and Mincheol Son. Shorter VOLE-in-the-Head-based Signatures from Vector Semi-Commitment. Cryptology ePrint Archive, Paper 2025/1077, 2025.

[KZ20] Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 3–22. Springer, Cham, December 2020.

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Cham, October 2021.

[LMSV12] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On CCA-secure somewhat homomorphic encryption. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 55–72. Springer, Berlin, Heidelberg, August 2012.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, Berlin, Heidelberg, August 1986.

[MN24] Mark Manulis and Jérôme Nguyen. Fully homomorphic encryption beyond IND-CCA1 security: Integrity through verifiability. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 63–93. Springer, Cham, May 2024.

[NLDD21] Deepika Natarajan, Andrew Loveless, Wei Dai, and Ronald Dreslinski. CHEX-MIX: Combining Homomorphic Encryption with Trusted Execution Environments for Two-party Oblivious Inference in the Cloud. Cryptology ePrint Archive, Paper 2021/1603, 2021.

[OPP14] Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 536–553. Springer, Berlin, Heidelberg, August 2014.

[Riv87] Ronald L Rivest. A method for obtaining digital signature and public-key cryptosystems. *ACM*, 21:2, 1987.

[Roy22] Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 657–687. Springer, Cham, August 2022.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, February 1978.

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

[SB93] Miles E. Smid and Dennis K. Branstad. Response to comments of the NIST proposed digital signature standard. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 76–88. Springer, Berlin, Heidelberg, August 1993.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

[Spa12] Pierre-Jean Spaenlehauer. *Solving multi-homogeneous and determinantal systems: algorithms, complexity, applications.* Phd thesis, Université Pierre et Marie Curie (Univ. Paris 6), October 2012.

[VKH23] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption. *arXiv preprint arXiv:2301.07041*, 2023.

[WCD+24] Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Jan Adriaan Leegwater, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang. SNOVA. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

[ZLW+25] Zhelei Zhou, Yun Li, Yunchen Wang, Zhaomin Yang, Bingsheng Zhang, Cheng Hong, Tao Wei, and Wenguang Chen. ZHE: Efficient Zero-Knowledge Proofs for HE Evaluations. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3087–3105. IEEE Computer Society, 2025.

[ZPS12] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. Reaction attack on outsourced computing with fully homomorphic encryption schemes. In Howon Kim, editor, *ICISC 11*, volume 7259 of *LNCS*, pages 419–436. Springer, Berlin, Heidelberg, November / December 2012.

[ZWL+25] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, and Xianhui Lu. FHE-SNARK vs. SNARK-FHE: From analysis to practical verifiable computation. Cryptology ePrint Archive, Report 2025/302, 2025.