UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

The Phantom Protocol
Harnessing Hypercubes and SIS in Post-Quantum Cryptography

Das Dunkle Protokol
Die Nutzung von Hypercubes und SIS in der Post-Quanten-Kryptographie

Masterarbeit

im Rahmen des Studiengangs
IT-Sicherheit
der Universität zu Lübeck

vorgelegt von
Julian M. Behrensen

ausgegeben und betreut von
Prof. Dr. Sebastian Berndt
Prof. Dr. Thomas Eisenbarth

mit Unterstützung von
Paula Arnold, MSc.

Lübeck, den 07. November 2024

Abstract

In this thesis, we explore the optimization of cryptographic protocols in the context of post-quantum security, where we focus on reducing the computational and communication overhead associated with zero-knowledge proofs and syndrome decoding. In particular, we examine the integration of *Small Integer Sharing* (SIS) and hypercube structures to optimize syndrome decoding, a key problem in coding theory with applications in post-quantum cryptography. Our contributions include the development of a novel protocol that significantly reduces the complexity of syndrome decoding in *multi-party computation in the head* (MPCitH) frameworks by leveraging SIS and hypercube geometries. In addition, we present a performance analysis of our optimized protocol, highlighting its efficiency gains compared to existing techniques. This research lays the groundwork for more practical implementations of post-quantum cryptographic protocols, enhancing their feasibility for real-world applications in secure communication and computation.

Zusammenfassung

In dieser Arbeit untersuchen wir die Optimierung kryptographischer Protokolle im Kontext der Post-Quanten-Sicherheit mit einem Schwerpunkt auf der Reduzierung des Rechen- und Kommunikationsaufwands, der mit Zero-Knowledge-Beweisen und Syndrom-Dekodierung einhergeht. Insbesondere betrachten wir die Integration von *Small Integer Sharing* (SIS) und Hypercube-Strukturen zur Optimierung der Syndrom-Dekodierung, einem zentralen Problem in der Kodierungstheorie mit Anwendungen in der Post-Quanten-Kryptographie. Unsere Beiträge umfassen die Entwicklung eines neuartigen Protokolls, das die Komplexität der Syndrom-Dekodierung in *Mehrparteien-Berechnungs-umgebungen* (MPC) durch die Nutzung von SIS und Hypercube-Geometrien erheblich reduziert. Darüber hinaus präsentieren wir eine Leistungsanalyse unseres optimierten Protokolls und heben dessen Effizienzgewinne im Vergleich zu bestehenden Techniken hervor. Diese Forschung bildet die Grundlage für praktischere Implementierungen post-quanten-kryptographischer Protokolle und erhöht deren Anwendbarkeit für reale Anwendungen in sicherer Kommunikation und Berechnung.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

[illegible]

Lübeck, 07. November 2024

Acknowledgements

I would like to express my deepest gratitude to everyone who has supported me throughout the journey to complete my master's thesis. This work would not have been possible without the guidance, encouragement and support of many individuals.
Firstly, I would like to extend my sincere thanks to my advisor, Dr. Sebastian Berndt, whose expertise, guidance, and encouragement were invaluable throughout every stage of this research. Paula Arnolds insightful feedback helped me overcome many challenges. Furthermore, I am grateful to Yara Schütt and Alexander Becker for their constructive criticism and the time devoted to improving this thesis.
On a personal note, I would like to thank my family and friends for their encouragement, patience, and love. A special thanks to my girlfriend Alina Hilck, your words of motivation and unwavering belief in me have been a source of unending strength during challenging times. And thanks to Yannik Pohl for being a great co-working partner, I don't think I would have been as disciplined without you.
Lastly, my heartfelt thanks go to everyone who, directly or indirectly, played a role in the successful completion of this thesis. I am immensely grateful to each of you.

Contents

1 Introduction

In this rapidly evolving digital landscape, data security has become a primary focus for researchers. With more services moving to digital platforms, ensuring the confidentiality, integrity, and authenticity of communications is critical. Cryptographic protocols form the backbone of secure communication and computation, protecting individuals, businesses, and governments from malicious attacks and data breaches. However, while robust against many classical attacks, current encryption and signature schemes are vulnerable to quantum algorithms like Shor's, which can break widely used cryptographic methods. This looming threat has sparked the development of *post-quantum cryptographic* (PQC) protocols that resist both classical and quantum attacks. In response to these challenges, the *National Institute of Standards and Technology* (NIST) launched the PQC standardization process. In this context, *zero-knowledge proofs* (ZKPs) and *multi-party computation* (MPC) protocols have become vital tools.
Zero-knowledge proofs allow one party (the prover) to convince another party (the verifier) that they possess knowledge of a specific piece of information, such as a solution to a mathematical problem, without revealing anything about the information itself. This property is essential in privacy-critical settings, such as authentication, blockchain transactions, and secure voting. Furthermore, MPC protocols enable multiple parties to jointly compute a function on their inputs while keeping these inputs private. MPCs are crucial in collaborative data analysis scenarios where parties wish to compute results from combined datasets without disclosing their data. While most of the introduced PQC protocols have focused on lattice-based cryptographic schemes, such as *Learning with Errors* (LWE), recent research by Yilei Chen [Che24] has uncovered potential vulnerabilities in lattice-based systems, raising concerns about their long-term security. It is important to note here that the paper contained some errors that led to its removal. However, this has still shifted attention to alternative cryptographic schemes, such as code-based cryptography, which has demonstrated stable security for over 40 years.
One of the most promising alternatives is the syndrome decoding problem, a

foundational problem in code-based cryptography. Unlike lattice-based cryptosystems, code-based schemes have remained resilient to both classical and quantum attacks, making them an attractive option for post-quantum security. At its core, the syndrome decoding problem involves finding the closest codeword to a given syndrome, which is computationally hard. The syndrome decoding problem gained further significance when it was integrated into the *MPC-in-the-Head* (MPCitH) framework by Feneuil et al. in 2022 [FJR22]. This innovation demonstrated the potential of syndrome decoding in secure multi-party computation protocols, providing a strong foundation for privacy-preserving cryptographic applications.
Despite their theoretical strengths, the practical deployment of these cryptographic techniques is often hindered by the significant computational and communication overhead required to ensure security. This thesis addresses these efficiency challenges by optimizing the sharing and verifying of a secret. By leveraging the geometric structure of hypercubes and integrating *small integer sharing* (SIS) schemes, this work aims to reduce the associated overheads, thus making these protocols more efficient and feasible for real-world applications.
Building on this, Aguilar-Melchor et al. [AGH$^+$22] introduced the hypercube structure to optimize syndrome decoding in MPCitH, significantly reducing computational costs and bringing the protocol closer to practical deployment. Despite these advancements, further refinements are needed to fully realize the potentiaprotocols' protocols for large-scale applications.
This research contributes to developing more practical post-quantum cryptographic systems, ensuring that they are better equipped to handle future quantum threats by focusing on syndrome decoding and optimizing both computational and communication costs.

1.1 Thesis Overview and Objectives

The primary objective of this thesis is to present novel approaches to reduce the overhead associated with zero-knowledge proofs and syndrome decoding within multi-party computation frameworks. To achieve this, we propose the following key contributions:

1. Optimization of Syndrome Decoding using SIS and Hypercube Structures: Syndrome decoding is a key problem in coding theory used in

several cryptographic applications, particularly post-quantum cryptography. By organizing the decoding process in the structure of a hypercube, we aim to reduce the computational complexity typically associated with this operation. This thesis explores how small integer sharing (SIS) can be integrated with hypercube structures to reduce further the number of operations required during syndrome decoding, particularly in high-dimensional settings.

2. Modified Security Proof: In addition to these optimizations, we provided the modified security proof for our protocol, ensuring that the improvements do not compromise the underlying cryptographic guarantees while maintaining the required security properties.

3. Implementation of the optimized protocol: We implemented the original syndrome decoding in the head protocol, the hypercube-based version, and our protocol in Python to show the practical impact of the different optimizations.

By addressing both computational and communication challenges, this thesis contributes to advancing the field of cryptography, making it more feasible to implement secure, large-scale computations without compromising efficiency or security, thus making post-quantum cryptographic protocols more accessible for real-world applications.

1.2 Structure of the Thesis

This thesis is structured as follows. In Chapter 2, we give an overview of the notation used in the thesis to provide the reader with a fixed set of variables and their meaning. After that, we present the mathematical and cryptographic background in Chapter 3. We introduce the mathematical building blocks and provide an overview of the necessary cryptographic elements, including the theoretical underpinnings of syndrome decoding multi-party computation and zero-knowledge proofs. In Chapter 4, 5, and 6, we introduce the three proposed methods for optimizing syndrome decoding, starting with SIS, followed by the hypercube structure and tree-based pseudorandom number generators. We combine these optimizations in Chapter 7, which forms the main contribution of this thesis. Here, we describe the necessary modifications to the optimization techniques and the final protocol.

In Chapter 8, we look at our protocol in a practical scenario and discuss its performance regarding communication and computational costs. Finally, in Chapter 9, we conclude this thesis by summarizing the main contributions and outlining potential avenues for future research.

2 Notation

Spaces	
$\mathbb{Z}$	For number spaces
$\mathbf{Z}$	For polynomials in big letters and in small letters for vectors
Z	For prover and verifier in texts and in formulas matrices
$\mathfrak{Z}$	For challenge space and sets in general
$\mathcal{Z}$	For algorithms, functions and extractors
Z	For security parameters besides λ and general parameters
Operations	
$\cdot$	Arithmetic multiplication
$/$	Arithmetic division
$+$	Arithmetic plus
$-$	Arithmetic minus
$\circ$	Coordinate wise multiplication
$\oplus$	XOR operation
$\sum$	Coordinate wise sum
$\langle \cdot, \cdot \rangle$	Scalar product
$\lvert \cdot \rvert$	The length of a vector or number of elements in a field
$\cdot \overset{\$}{\leftarrow} \cdot$	Random sampling of a given element or vector
General	
$\mathbf{1}$	A vector of ones
$\mathbf{0}$	A vector of zeros
$[B] = \{0, \ldots, B-1\}$	The numbers from 0 to the given maximum B
sk	Secret key
pk	Public key
$\prod$	A non-specified protocol
$deg(\cdot)$	A function that returns the degree of a given polynomial $(\cdot)$

Multy-Party Computation	
n	Number of parties
N	Number of shares
$\mathbf{x}$	The secret
$[\![x]\!]$	Shares of the secret x
$\Delta\mathbf{x}$	Auxiliary value of the secret
$i \in [N]$	Index of one share
i^*	Index of the hidden share
$j \in [n]$	Index for one party
MPCitH	
ε	The soundness error
$\tilde{\varepsilon}$	The inverse soundness error
t	The number of points for the polynomial evaluation
$z \in \mathbb{F}_{points}$	Point for the polynomial evaluation
τ	The number of protocol executions
P	Prover
V	Verifier
$\tilde{\mathsf{P}}$	General prover (does not need to be honest)
$\tilde{\mathsf{V}}$	General verifier (does not need to be honest)
A	A probabilistic polynomial time adversary
$\mathcal{E}$	The extractor
$\mathcal{S}$	A Simulator
$\epsilon \in \mathbb{F}_{points}$	Challenge points for product verification
h	The commitment hash of the secret shares
$h^{'}$	The commitment hash of the batch product verification shares
$\langle \mathsf{P}, \mathsf{V} \rangle$	Instance of a zero-knowledge between P and V
Syndrome Decoding	
$\mathbb{F}_{SD}$	Finite field of the syndrome decoding
$\mathbb{F}_{poly}$	Finite field extension of $\mathbb{F}_{SD}$ from which the polynomials $\mathbf{S}, \mathbf{Q}, \mathbf{P}, \mathbf{F}$ are selected
$\mathbb{F}_{points}$	Finite field extension of $\mathbb{F}_{poly}$ from which the evaluation points of $\mathbf{S}, \mathbf{Q}, \mathbf{P}, \mathbf{F}$ are selected
$\mathbf{x} \in \mathbb{F}_{SD}^{m}$	The secret of the syndrome decoding
$\mathsf{H} \in \mathbb{F}_{SD}^{m-k \times m}$	The matrix of the syndrome decoding

$\mathsf{y} \in \mathbb{F}_{SD}^{m-k}$	The resulting vector of the syndrome decoding calculation $\mathsf{H} \cdot \mathbf{x} = \mathsf{y}$
$m \in \mathbb{F}_{SD}$	The length of the solution $\mathbf{x}$
$k \in \mathbb{F}_{SD}$	A security parameter for the syndrome decoding problem
$w \in \mathbb{F}_{SD}$	Hamming weight bound
a, b, c	Elements of the Beaver triplet such that $a \cdot b = c$
α, β, v	Communication outputs drawn from $\mathbb{F}_{points}$
$wt(\cdot)$	The function to calculate the hamming weight of a given vector
Small Integer Sharing	
$\mathbf{g} \in \mathbb{Z}_q^n$	The vector containing the elements to choose for the subset sum problem
$h\mathbb{Z}_q$	The final sum of the subset sum problem
$\mathbf{r} \xrightarrow{\$} \{0,1\}^n$	The random vector for the cut-and-choose strategy
M	The number of generated vectors $\mathbf{r}$
$\mathcal{L}$	The set of indices for the challenges of a random vector $\mathbf{r}$
$\mathcal{J}$	The set of indices for the second challenge regarding the opening of shares
$\mathsf{E_v}, \neg\mathsf{E_v}$	An event and the complementary event
$\mathcal{A}_j^a$	The rejection event with $\begin{cases} a = 0 & \text{if } \mathbf{x}_j = 0 \\ a = 1 & \text{it } \mathbf{x}_j = 1 \end{cases}$
Hypercube	
D	The dimension of the hypercube
N_H	Number of shares per hypercube dimension
ls	A leaf-party
ms	A main-party
TreePRG	
hs	The number of sub-trees that share a hidden share

Table 2.1: This table displays the notation used in this thesis, including variable names and their meaning as well as general notations for polynomials, finite fields and other.

3 Preliminaries

This section provides the theoretical foundations for this thesis's cryptographic protocols and optimizations. We begin by outlining key lemmata, cryptographic definitions, and the fundamental concepts of zero-knowledge proofs and *multi-party computation* (MPC). In connection to this, we describe additive secret sharing and the *MPC-in-the-Head* (MPCitH) paradigm, which are integral to the protocol we will optimize in this work.

Additionally, we introduce the syndrome decoding problem, a central problem in code-based cryptography, which serves as the basis for the protocol optimizations explored in later chapters. These preliminaries ensure a comprehensive understanding of the underlying methods used throughout the thesis.

3.1 Lemmata

In this section, we begin by presenting key lemmata that will serve as the mathematical tools underpinning various cryptographic proofs throughout this thesis. The first is the Demillo-Lipton-Schwarz-Zippel lemma, short Schwatz-Zippel lemma, which is critical in verifying polynomial identities over finite fields. This lemma provides a probabilistic bound, ensuring that a non-zero polynomial rarely evaluates to zero on randomly chosen points. In the context of cryptographic protocols, this helps us efficiently verify the correctness of certain computations without revealing sensitive information.

Lemma 1: Multi-point Schwartz-Zippel (variant 1)

Let $\mathbf{P} \in \mathbb{F}[X]$ be a non-zero polynomial in one variable of at most degree $d > 0$. Moreover, let $\mathbb{S} \subseteq \mathbb{F}$ be a non-empty set. For any $t \geq 1$:

$$Pr[\mathbf{P}(r_i \overset{\$}{\leftarrow} \mathbb{S}) = 0, \forall i \in [t]] \leq \frac{\binom{d}{t}}{\binom{|\mathbb{S}|}{t}}. \tag{3.1}$$

Lemma 1: Multi-point Schwartz-Zippel (variant 1)

Proof: When uniformly sampling t distinct elements from $\mathbb{S}$, there are $\binom{|\mathbb{S}|}{t}$ possible combinations. However, since the polynomial can have at most d roots, the event of interest can only occur for $\binom{d}{t}$ of these samples [AGH+22].

The multi-point Schwarz-Zippel lemma can be extended to apply to syndrome decoding combined with the MPCitH paradigm. It provides a more nuanced view of polynomials' behavior under certain constraints, which is helpful when considering applications with multiple accounted roots.

Lemma 2: Multi-point Schwarz-Zippel (variant 2)

Let $\mathbf{R} \in \mathbb{F}[X]$ be a polynomial of degree $d > 0$. For any $\mathbb{S} \subset \mathbb{F}$ and any $t, l \geq 1$,

$$Pr_{r_1,\ldots,r_t \xleftarrow{\$} \mathbb{S}}[\#\{i : \mathbf{R}(r_i) = 0\} = l | \{r_i\} \text{ are unique}] \leq \frac{\max_{i \leq d} \left\{ \binom{i}{l} \cdot \binom{|\mathbb{S}|-i}{t-l} \right\}}{\binom{|\mathbb{S}|}{t}}$$

If $t \cdot d \leq l \cdot (|\mathbb{S}| - 1)$, we have:

$$Pr_{r_1,\ldots,r_t \xleftarrow{\$} \mathbb{S}}[\#\{i : \mathbf{R}(r_i) = 0\} = l | \{r_i\} \text{ are unique}] \leq \frac{\left\{ \binom{d}{l} \cdot \binom{|\mathbb{S}|-d}{t-l} \right\}}{\binom{|\mathbb{S}|}{t}}$$

The proof for this lemma is in Appendix C of [FJR22].

With the Schwarz-Zippel lemma providing a probabilistic framework for polynomial verification, we now turn to the XOR lemma. This lemma is crucial for efficiently handling bitwise operations in cryptographic protocols. It allows us to reconstruct integers using XOR and AND operations, which is particularly useful for binary secret sharing and secure computation. This will play an important role in ensuring the correctness of secret reconstruction while maintaining efficiency in the Small Integer Sharing in Section 4.

Lemma 3: in Annex 1 of [Gou01]

For any integers u, v, the following holds:

$$u - v = (u \oplus v) - 2 \cdot (\overline{u} \wedge v) \mod 2^K \tag{3.2}$$

Which can be refactored to the following:

$$u + v = (u \oplus v) + 2 \cdot (u \wedge v) \mod 2^K \tag{3.3}$$

Having established the XOR lemma for handling bitwise operations, we now introduce the Splitting Lemma, which provides a probabilistic method for analyzing subsets of larger sets. This lemma is particularly useful in cryptographic protocols where we need to isolate specific subsets of data while maintaining certain security guarantees. In the context of multi-party computation and secret sharing, the Splitting Lemma helps ensure that security properties hold even when only partial information is revealed, making it essential for the soundness of our protocols.

Lemma 4: Splitting Lemma [PS00]

Given $A \subset X \times Y$ and $Pr\left[(x, y) \in A\right] \geq k$, then for any $\alpha \in [0, 1)$ let:

$$B = \left\{(x, y) \in X \times Y | Pr_{y' \in Y}\left[(x, y') \in A\right] \geq (1 - \alpha) \cdot k\right\} \tag{3.4}$$

Then, the following is true: $Pr[B] \geq \alpha \cdot k$ and $Pr[B|A] \geq \alpha$.

Another crucial lemma for the soundness of MPCitH protocols is the forking lemma. It provides the security of multi-round interactive protocols, such as 5-round protocols, which involve several back-and-forth exchanges between the prover and verifier. The verifier sends multiple challenges across different rounds in these protocols, and the prover must respond correctly to each. The forking lemma ensures that if an adversary can produce valid responses for different challenges (e.g., by making different oracle calls), they can be forced to reveal key information about their strategy, helping to detect potential cheating attempts.

Lemma 5: Forking Lemma for 5-pass protocols [DGV+16]

Given a 5-pass signature scheme $\mathcal{S}$ with a security parameter k. Let $\mathcal{A}$ be a probabilistic polynomial time algorithm that is given public data as input. Furthermore, assume that $\mathcal{A}$ outputs a valid signature $(\sigma_0, \sigma_1, \sigma_2, h_1, h_2)$ for a message m with a non-negligible probability. Then replaying $\mathcal{A}$ with the same tape and the same response to the query corresponding to $\mathcal{O}_1$, but with a different output to the second oracle $\mathcal{O}_2$, will reply with a second distinct and valid signature $(\sigma_0, \sigma_1, \sigma_2', h_1, h_2')$. These two signatures correspond to the same message m and have $h_2 \neq h_2$ with non-negligible probability.

With the key lemmata in place, we can now focus on the core cryptographic concepts that form the foundation of our protocols.

3.2 Cryptographic Definitions

In this section, we describe the core cryptographic concepts relevant to the analysis and design of secure protocols. These definitions provide the necessary formalism for discussing security proofs and the behavior of cryptographic functions. We start with the indistinguishability concept, which states that two distributions are indistinguishable from each other within a given time span. In the context of MPCitH, it ensures that, even if an adversary observes the output or exchanges in the protocol, they should be unable to distinguish whether the data come from an honest execution or a random source.

Indistinguishability

Given two distributions X, Y, a time bound t and a function ϵ. Then X, Y are (t, ϵ)-indistinguishable if for an algorithm running in time t there is a function $\mathcal{D} : \{0,1\}^m \to \{0,1\}$ with $|Pr[\mathcal{D}(X) = 1] - Pr[\mathcal{D}(Y) = 1]| \leq \epsilon(\lambda)$. Note that $\epsilon(\lambda)$ is a negligible function, meaning for sufficiently large λ, its output becomes insignificantly small. We also consider three more specific indistinguishabilities:

1. **Computational indistinguishability:** X, Y are computational indistinguishable if $t = poly(\lambda)$ and ϵ is a negligible function in λ.
2. **Statistically indistinguishability:** X, Y are statistically indistin-

Indistinguishability

guishable when t can be unbounded while ϵ remains a negligible function in λ.

3. **Perfect indistinguishability:** X, Y are perfectly indistinguishable if $\epsilon = 0$, meaning no algorithm, regardless of running time, can distinguish between X and Y with any advantage.

Additionally, indistinguishability is essential for the pseudorandom generation. A *pseudorandom generator* (PRG) is an algorithm that takes a short, truly random input (called a seed) and expands it into a longer, seemingly random output. The output of a PRG is designed to be indistinguishable from a truly random sequence.

Pseudorandom generation (PRG)

Let $\mathcal{G} : \{0,1\}^* \rightarrow \{0,1\}^*$ be a function that takes a binary string as input and returns a binary string. Additionally let $\mathbf{L}(\cdot)$ be a polynomial with $\mathcal{G}(\mathbf{S}) \in \{0,1\}^{\mathbf{L}(\lambda)}$ for any input $\mathbf{S} \in \{0,1\}^{\lambda}$. Following this $\mathcal{G}$ is a (t, ϵ)-secure pseudorandom generator if it satisfies these two properties:

- **Expansion:** $\mathbf{L}(\lambda) > \lambda$, which means that given the input $\mathbf{S}$ the generated output is longer than the input.
- **Pseudorandomness:** The two distributions $\{\mathcal{G}(\mathbf{S}) | \mathbf{S} \leftarrow \{0,1\}^{\lambda}\}$ and $\{\mathcal{F} | \mathcal{F} \leftarrow \{0,1\}^{\mathbf{L}(\lambda)}\}$ are (t, ϵ)-indistinguishable.

These two techniques give us the basis for describing collision-resistant hash functions. These functions generate a seemingly random value based on their input while ensuring that it is computationally infeasible to find a different input such that the function generates the same output (collision).

Collision-Resistant Hash Functions

A function or a family of functions $\mathcal{H} : \{0,1\}^m \rightarrow \{0,1\}^n$ is a collision-resistant hash function if it satisfies the following two conditions for a given security parameter λ:

- **Length compression:** The input string $\{0,1\}^m$ is longer than the output string $\{0,1\}^n$ and therefore we have $m < n$ with a common

Collision-Resistant Hash Functions

choice of $m = n/2$.

- **Hard to find collision:** For all non-uniform *probabilistic polynomial time* (PPT) algorithms $\mathcal{A}$ there exists a negligible function ϵ, such that

$$Pr[(x_0, x_1) \leftarrow \mathcal{A}(1^n, \mathcal{H}) : x_0 \neq x_1 \text{ and } \mathcal{H}(x_0) = \mathcal{H}(x_0)] \leq \epsilon(\lambda)$$

  This means that it is difficult ($\leq \epsilon(\lambda)$) to find two different inputs (x_0, x_1) for the given hash function $\mathcal{H}$, which results in the same hash.

With this in mind, we can introduce cryptographic signatures, which are crucial in ensuring the authenticity and integrity of communications in cryptographic systems. We will give a short and informal definition, as we focus on the protocols that form the bases to produce signatures but not the actual signature generation.

Cryptographic Signature

A cryptographic signature, also called a digital signature, is a technique used to verify a message's authenticity, binding, and integrity. Meaning that a receiver can verify that the sender is who she appears to be and that the message has not been tampered with. Thus, a signature needs to satisfy the following three characteristics:

- **Authenticity:** The sender attaches a signature to the message, ensuring that she is the sender.
- **Binding:** The sender is bound to the content of the message.
- **Integrity:** Neither the sender nor an attacker can manipulate the message after signing.

In addition, these techniques need to provide a signing function and a verification function, both at a reasonable expense. One can implement these characteristics using an asymmetric technique such as the RSA signature or utilize zero-knowledge protocols, thorough, for example, MPCitH (3.8).

Now that we have a basic understanding of cryptographic signatures, we can explore other foundational concepts contributing to secure protocol design.

3.3 Canonical Inclusion

One such concept is the canonical inclusion function, commonly encountered in set theory and cryptography. The canonical inclusion map formalizes the relationship between a subset and its parent set by treating each element of the subset as an element of the parent without modification. Intuitively, this map preserves the identity of elements from the subset within the parent set, ensuring that their properties remain intact. This is essential for the polynomial generation of the syndrome decoding in the head protocol.

Canonical Inclusion

The inclusion map, also known as the inclusion function, insertion, or canonical insertion, is a mathematical function that maps each element of one set to its representation in another set. For this, the first set of elements must be a subset of the second set. The $\hookrightarrow$ denotes the inclusion map, and the first set is a subset of the second.

For example, we are given a set S and a set T. The inclusion map for S to T maps each element of S to itself in the set T and is denoted as $S \hookrightarrow T$. Thus, we also know that $S \subseteq T$ holds.
Another essential technique for MPCitH protocols are zero-knowledge proofs, which we describe in the next section.

3.4 Zero-Knowledge Proofs

This section defines the essential properties of a zero-knowledge proof of knowledge. A zero-knowledge proof is a two-party protocol between a prover P and a verifier V for a given language $L \in NP$. It is represented as $\langle \mathsf{P}, \mathsf{V} \rangle$, where $\langle \mathsf{P}, \mathsf{V} \rangle(x)$ is the execution of the protocol on a given witness (possible solution to the problem) x. The proof must satisfy several essential properties.
The core idea is that the prover must show the verifier that their standard input x belongs to the given language L and satisfies specific properties without revealing any additional information. For example, the prover might have a se-

cret $\mathbf{x}$ related to a vector y and a scalar a such that $\langle \mathbf{x}, \mathsf{y}\rangle = a$ and needs to prove to the verifier that $\mathbf{x}$ satisfies this equation. We formalize this by defining the following three properties: completeness, soundness, and zero-knowledge.

(Perfect) Completeness

The completeness property for a zero-knowledge proof $\langle \mathsf{P}, \mathsf{V}\rangle$ states that if both the prover P and the verifier V follow the protocol honestly, meaning they do not deviate from the agreed-upon set of steps (the protocol, which defines the rules and interactions between the parties), and the prover has a correct witness x, then for every such witness $x \in L$, V always accepts:

$$Pr[\langle \mathsf{P}, \mathsf{V}\rangle(x) = 1] = 1 \tag{3.5}$$

Completeness ensures that an honest prover, using a valid witness, will always convince an honest verifier. This property is necessary for the protocols correctness, ensuring that the protocol functions as intended.

Soundness

The proof of knowledge is so-called sound, with a soundness error ε, if an honest verifier accepts with probability less than ε for a probabilistic polynomial time adversary A using a witness $x \notin L$:

$$Pr[\langle \mathsf{A}, \mathsf{V}\rangle(x) = 1] \leq \varepsilon \tag{3.6}$$

In other words, a prover who does not have a valid witness x cannot successfully convince the verifier of his knowledge with probability greater than ε. Thus, soundness prevents the verifier from being deceived by a malicious prover who presents an invalid witness.

Honest Verifier Zero-Knowledge (HVZK)

The HVZK property states that there is a simulator $\mathcal{S}$ for a proof of knowledge, which operates in probabilistic polynomial time. This simulator can produce output transcripts that are computationally indistinguishable from distributions of transcripts from an honest execution of the protocol without knowing a witness x.

This shows that running the protocol does not reveal any information about the witness to an honest verifier. In the following, we will use zero-knowledge proof as a shorthand for HVZK proof.
With the concept of honest-verifier zero-knowledge established, we now introduce commitment schemes, another essential cryptographic primitive that works in conjunction with zero-knowledge proofs to form the MPCitH protocol.

3.5 Commitments

In connection to zero-knowledge proofs, we need the so-called commitment scheme, which is a cryptographic primitive that allows one to send a public value C, the commitment, to another party to reveal the correlated hidden value later. Revealing this value is called opening and requires a decommitment value D. In order to have a helpful commitment scheme, it is vital that once the party commits to the correlated hidden value, they should not be able to change it later on (binding). In addition, every receiving party should only be able to gain knowledge of the hidden value by using the decommitment value D (hiding). Considering these requirements, we can define the following characteristics of a commitment scheme. The first three properties are essential for any commitment scheme, while the last two ensure the scheme is secure.

Commitment Scheme

Any commitment scheme consists of two *probabilistic polynomial time* (PPT) algorithms, the commitment **com** and the opening algorithm **open**, which we define as:

- **com**(M), given an input $M \in \{0,1\}^* = \mathcal{M}$ the commitment algorithm outputs a tupel $(C, D) \leftarrow$ **com**(M, p) consisting of the commitment C and the decommitment D generated using the commitment randomness p.
- **open**(C, D), given the commitment and a decommitment **open** returns either the hidden value M in case of a matching pair or the bottom symbol $\perp$ otherwise.

These properties, in combination with **correctness**, ensure the correct mode of operation of a commitment scheme, while **binding** and **hiding** define its security.

Commitment Scheme

Correctness ensures that the original value will always be correctly recovered if the commitment is formed and appropriately decommitted. This property is vital for the integrity of any commitment scheme.

- **Correctness:** Given the commitment **com**$(M) \leftarrow (C, D)$ without any manipulation, then **open**(C, D) should always return the hidden value M.

Perfectly binding ensures that a committed value cannot be altered once it has been committed.

- **Perfectly Binding:** In order for a commitment scheme to be perfectly binding, any PPT algorithm for a given security parameter λ must have a probability of zero for finding C, D, D' such that **open**$(C, D) = M$ and **open**$(C, D') = M'$ with $M \neq M'$. This means that given the same commitment C, the probability of finding two different decommitments D and D' such that the PPT algorithm obtains different messages M and M' from the same commitment is zero. Thus, it is impossible to change the hidden value of the commitment afterwards.

We can reduce the binding characteristic for realistic scenarios by using a computationally binding definition. It states that the probability of this algorithm finding such a tuple is a negligible function in λ.
Lastly, perfectly hiding prevents an adversary from gaining any knowledge of the committed value until it is opened.

- **Perfectly Hiding:** A commitment scheme **com**$(M) \rightarrow (C, D)$ is perfectly hiding if for any two messages M, M' the distributions of $\{C : (C, D) \leftarrow$ **com**$(M)\}_{\lambda \in \mathbb{N}}$ and $\{C : (C, D) \leftarrow$ **com**$(M')\}_{\lambda \in \mathbb{N}}$ are perfectly indistinguishable. In other words, it is impossible for any algorithm to identify which message M or M' was used to generate the commitment C, and thus, no information can be gained from C.

The perfectly hiding property can be modified to be statistically or computationally hidden by reducing the bound on the distributions to be statistically or computationally indistinguishable.

It is important to note that a commitment scheme cannot be perfectly binding and hiding. On the one hand, perfectly binding requires the commitment scheme to be deterministic to ensure that the commitments correspond to exactly one input value. On the other hand, the perfectly hiding property requires the commitment scheme to contain randomness so that no information about the input value is leaked. This contradiction becomes clear with the following example:

Hiding Binding Contradiction

Given a perfectly binding commitment scheme **com**$(M) \rightarrow (C, D)$ with $M \in \mathcal{M}$, we know that no other input $M' \in \mathcal{M}$ can correspond to the commitment C. However, an unbounded adversary could now generate a commitment for every message in $\mathcal{M}$ and get a unique commitment for each message. This would inadvertently tell the adversary the input message M for the commitment C and thus break the perfectly hiding property.

In conclusion, the commitment scheme provides a secure way of locking in a value while keeping it hidden until the appropriate time to reveal it. This way, commitment schemes form the backbone of many cryptographic protocols, including MPCitH.

3.6 Additive Secret Sharing

To perform the multi-party computation, we also need to be able to break up our secret into multiple parts and share these with the other members of a protocol. For this, we can use the common technique of *additive secret sharing* (AddSS), where we sample $N - 1$ random values, sum them up, and calculate the difference (auxiliary value) between them and the secret. The auxiliary value is then used as the Nth random value. After that, each random and the auxiliary value can be sent to one of the parties. As long as at least one of the sent values stays hidden, it is impossible to obtain the secret. With this in mind, we can give a formal definition as follows:

Additive Secret Sharing

Let there be n many parties to share the secret x with, and thus N shares that need to be created. The $N-1$ first shares, written as $[\![x]\!]_i$ with $i \in \{1,\ldots,N-1\}$ are generated via a *pseudorandom generator* (PRG) over a finite field $\mathbb{F}$

$$[\![x]\!]_i \xleftarrow{\$} \mathbb{F}^*.$$

Then the last share $[\![x]\!]_N$ is calculated via:

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

Thus, the final output is a tupel of N shares $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$. The reconstruction works by summing up all shares:

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

Using this sharing, obtaining the secret x without knowing all N shares is impossible. Furthermore, each party can perform the following computations and preserve a valid sharing:

- **Addition:** Given two shares or sets of shares $[\![x_A]\!]$, $[\![x_B]\!]$ one can calculate the sum of them.

  $$\forall i \in \{1,\ldots,N\}, [\![x_A + x_B]\!]_i := [\![x_A]\!]_i + [\![x_B]\!]_i$$

  In shorthand this is written as $[\![x_A + x_B]\!] := [\![x_A]\!] + [\![x_B]\!]$

- **Addition with a constant:** Given a constant c and the set of shares $[\![x]\!]$, one calculates the sum by adding the constant to the first share:

  $$\left(\forall i \in \{1,\ldots,N\}\right) \begin{array}{l} [\![x + c]\!]_1 := [\![x]\!]_1 + c \\ [\![x + c]\!]_i := [\![x]\!]_i \ \forall i \in \{2,\ldots,M\} \end{array}$$

  This is denoted as $[\![x + c]\!] := [\![x]\!] + c$

- **Multiplication:** The multiplication of shares can be realized through Beaver triples [Bea92], which require additional communication

Additive Secret Sharing

between the parties. The additional input is a secret-shared triplet $[\![a]\!], [\![b]\!], [\![c]\!]$ where the values a and b are unknown to all parties, but the result $c = a \cdot b$ is published. This triplet can then be used to validate a different triplet by sacrificing it in the process. This means both a and b are revealed to all parties. We will describe how this is used in the multi-party computation in the head protocol in Section 3.9.

- **Multiplication with a constant:** The multiplication with a constant works similar to the addition with a constant, but instead of multiplying the constant with the first share, it is multiplied with every share:

$$\forall i \in \{1, \ldots, N\}, [\![c \cdot x]\!]_i := c \cdot [\![x]\!]_i$$

The shortened notation is $[\![c \cdot x]\!] := c \cdot [\![x]\!]$.

After gaining a good understanding of zero-knowledge proofs, commitment schemes, and additive secret sharing, we will describe the multi-party computation paradigm in the next section.

3.7 Multi-Party Computation

With the *multi-party computation* (MPC) protocol, n parties of the protocol that have a secret x_i can compute a function f on these secrets without leaking any information about their secret to the other parties. The only information that each party is allowed to gain from the protocol is the information that can be reconstructed from the output $y = f(x_1, \ldots, x_n)$. This requires the parties to be honest, meaning they follow the protocol $\prod$ and do not share their secret with any other party.
A more realistic approach is the semi-honest MPC protocol, which commonly utilizes the additive secret sharing introduced in the previous section. Semi-honest means that each party follows the protocol but is allowed to share its secret with the other corrupt parties. Considering the additive secret sharing with N shares, each party needs all other shares to recover the secret x of the protocol. Thus, it is $(N-1)$ secure, i.e. secure, even if $N-1$ parties collaborate.
Moreover, we can define the evaluation function of the MPC protocol as a

boolean decision function $f(x) : \mathbb{Z}^* \rightarrow \{0, 1\}$, which is needed for the Multi-party in the head paradigm described in the next section. The boolean decision function returns a truth value, such as *Accept* (1) or *Reject* (0) instead of a more complex result.

3.8 Multi-Party in the Head Paradigm

For the MPC protocol to be used in a ZK-proof environment, Ishai, Kushilevitz, and Ostrovsky [IKOS07] provided a technique to create such proofs for arbitrary circuits, called *Multi-Party computation in the Head* (MPCitH). It works by creating a two-party protocol with a prover P who wants to convince the second party, the verifier V, of his knowledge of a secret x, for which $f(x) = 1$. The function f is a predicate that has either a unique solution or a difficult-to-find solution and returns either *Accept* $(f(x) = 1)$ or *Reject* $(f(x) = 0)$. Furthermore, f does not need to be deterministic, and thus a *good witness* corresponds to x with $Pr[f(x) = 1] = 1$. Otherwise, $f(x)$ will *Reject* most of the time but has a small *false positive probability* p. We provide an overview of the probability distribution for a good and bad witness in Table 3.1. In the context of the MPCitH paradigm, note that the verifier provides randomness for the evaluation function f. The prover must, therefore, commit to the simulated views before receiving the randomness from the verifier. From this, we know that an honest prover will always convince the verifier, but in the case of a malicious prover $\tilde{\mathsf{P}}$, there is a success probability of cheating for a random selection of $N-1$ parties of $\frac{1}{N}$. With the false positive probability p, we get a resulting soundness error for the zero-knowledge protocol of:

$$\varepsilon = 1 - \left(1 - \frac{1}{N}\right) \cdot (1 - p) = \frac{1}{N} + p - \frac{1}{N} \cdot p \tag{3.7}$$

In this context, the number of shares N and the number of parties n are commonly the same $(N = n)$. The general mode of operation for the prover of the MPCitH protocol is as follows.

1. Generate N shares $[\![x]\!] \leftarrow Share(x)$ of her secret x, which are then distributed among n imaginary parties.
2. Then simulate the decision function f for all the n parties. Since the prover simulates the parties, this step is performed 'in the head'.

Table 3.1: This table shows the probability distribution of the output of an MPC protocol regarding a good and bad witness.

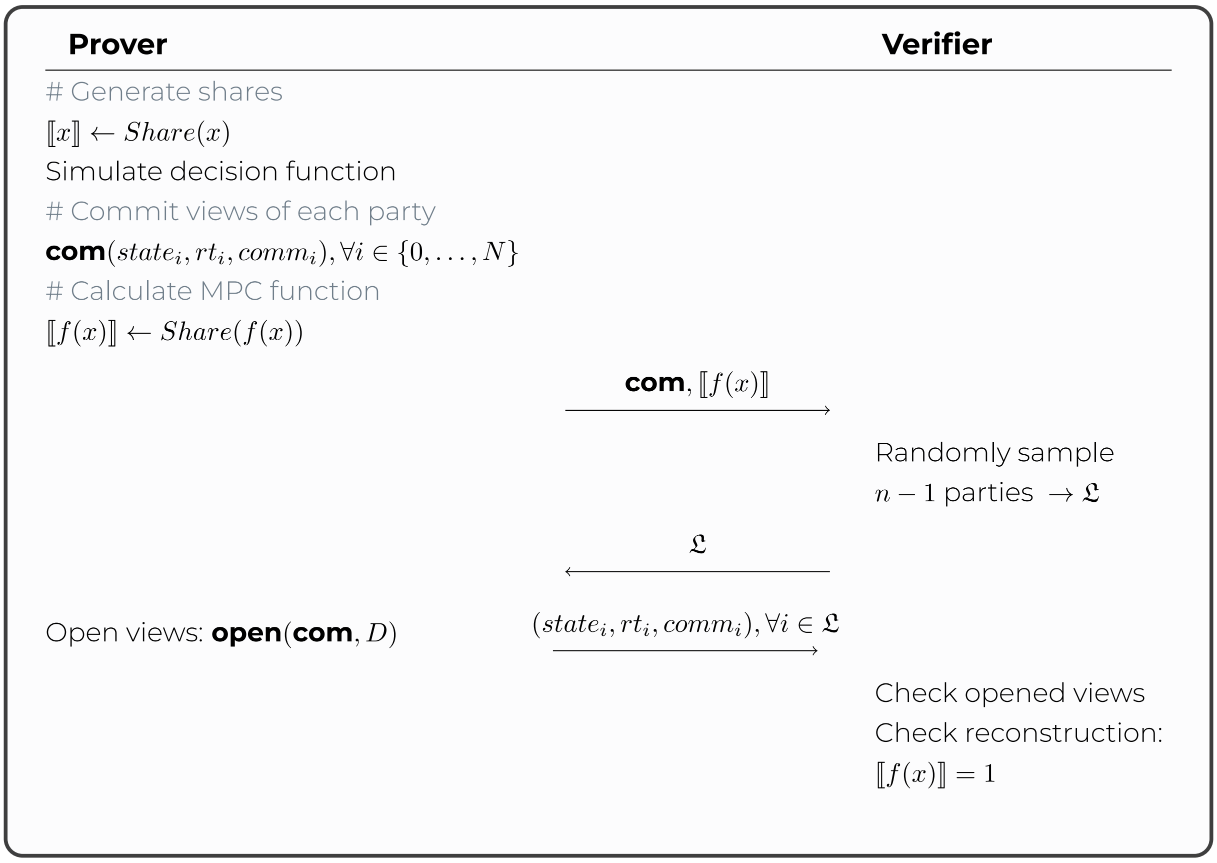
	Accept	**Reject**
good witness x	1	0
bad witness x	p	$1-p$

3. After that, the prover commits to each party's view and sends the commitment to the verifier, such that P cannot change the views later on. The views contain the initial shares ($state$), the secret random tape (rt), which is the private randomness used by each party, and the inbound and outbound communications ($comm$) between the parties.
4. Finally she sends the shares of the computed result $[\![f(x)]\!]$ to the verifier.

After the prover finishes her steps, the verifier will perform the following steps to complete the protocol, including some involvement of the prover.

1. V chooses $(n-1)$ random parties and asks the prover to open her views, which we denote as the first challenge $\mathfrak{L}$.
2. After receiving the opened views, the verifier checks whether the prover performed the MPC protocol honestly by verifying that the opened views result in the sent commitments.
3. Lastly, V agrees if the opened views are consistent and the shares of $f(x)$ reconstruct to 1.

By the definition of AddSS as in Section 3.6, $N-1$ shares are insufficient for secret reconstruction. Therefore, this MPCitH protocol does not leak any information about the secret x to the verifier. Moreover, the random selection of $n-1$ parties implies a soundness error of $\frac{1}{n}$ because a malicious prover would need to cheat on the hidden share to avoid getting caught. The reason for this is that the verifier opens up all other parties and, thus, would notice manipulation in those shares. Thus, the soundness error boils down to correctly guessing the hidden share before the commitment, which has a probability of $\frac{1}{n}$. In order to further reduce the soundness error, one commonly executes the MPCitH protocol multiple times (t times).
This protocol is visualized in Protocol 3.1.


Prover
Verifier
# Generate shares
$[\![x]\!] \leftarrow Share(x)$
Simulate decision function
# Commit views of each party
com$(state_i, rt_i, comm_i), \forall i \in \{0, \dots, N\}$
# Calculate MPC function
$[\![f(x)]\!] \leftarrow Share(f(x))$
com, $[\![f(x)]\!]$
Randomly sample
$n-1$ parties $\rightarrow \mathfrak{L}$
$\mathfrak{L}$
Open views: **open**(**com**, D)
$(state_i, rt_i, comm_i), \forall i \in \mathfrak{L}$
Check opened views
Check reconstruction:
$[\![f(x)]\!] = 1$

Protocol 3.1: This is a visualization of the MPCitH protocol described in this section. It is an interactive protocol, which can be a three-round protocol, as shown here, or a five-round one. In this protocol, the prover must show the verifier that she has a secret x without revealing it to the verifier. This is done through additive secret sharing, commitment, and a function f that has a unique solution.

3.9 Multi-Party Product Verification

As mentioned in Section 3.6, it is instrumental for many MPC and MPCitH protocols to have the ability to check the correctness of the product of shares. For this, we define a triple of sharings $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ of three elements $a, b, c \in \mathbb{F}$ that satisfy $a \cdot b = c$. This triple is called a multiplication triple or *Beaver triple* because of its inventor, Donald Beaver. To verify the correctness of a multiplication triple, [[LN17], [BN19]] proposed an MPC protocol that sacrifices another triple to verify $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$. Given the triple $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ they use a random triple $(\llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket z \rrbracket)$ to verify the correctness of both triple, meaning that $a \cdot b = c$ and $x \cdot y = z$ by only revealing information of the random triple. To achieve this, the protocol performs the following six steps:

1. The parties get a random value $\epsilon \in \mathbb{F}$
2. They calculate locally their share of $\llbracket \alpha \rrbracket = \epsilon \cdot \llbracket a \rrbracket + \llbracket x \rrbracket$ and $\llbracket \beta \rrbracket = \llbracket b \rrbracket + \llbracket y \rrbracket$
3. The parties broadcast their shares $\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket$ to obtain α and β
4. Each party calculates locally $\llbracket v \rrbracket = \epsilon \cdot \llbracket c \rrbracket - \llbracket z \rrbracket + \alpha \cdot \llbracket y \rrbracket + \beta \cdot \llbracket x \rrbracket - \alpha \cdot \beta$
5. They broadcast $\llbracket v \rrbracket$ such that every party has v
6. The parties output *Accept* in case of $v = 0$ and *Reject* otherwise

We can argue the correctness of this protocol, such that each party will always output *Accept* if the multiplication triples are correct because the following holds:

$$\begin{aligned} v &= \epsilon \cdot c - z + \alpha \cdot y + \beta \cdot x - \alpha \cdot \beta \\ &= \epsilon \cdot a \cdot b - x \cdot y + (\epsilon \cdot a + x) \cdot y + (b + y) \cdot x - (\epsilon \cdot a + x) \cdot (b + y) \\ &= 0. \end{aligned}$$

We can define the soundness error of the multi-party product verification by looking at the inverse case of at least one triplet being incorrect. This is equivalent to randomly selecting v in $\mathbb{F}$, such that $v = 0$ is true. The probability of this is $1/|\mathbb{F}|$, which is stated in Lemma 6 from Carsten Baum and Ariel Nof in [BN19].

Lemma 6: [BN19]

If $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ or $(\llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket z \rrbracket)$ is an incorrect multiplication triple then the parties output *Accept* in the protocol described above with a probability at most $1/|\mathbb{F}|$.

This protocol creates considerable communication overhead in MPCitH protocols because the additional triple must be sent for every execution of the protocol. This problem is tackled by [KZ22], reducing the communication cost for z and v through batching. This means that instead of sending z, v in each execution, one sends an instance of them for all executions. This, however, increases the false positive probability. This means it is not an all-time solution, but it should instead be considered carefully whether the increase in p is acceptable.

Batch Product Verification

The protocol batches the verification of t multiplication triples $(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket)$, by sacrificing a random dot-product tuple $(\llbracket x_i \rrbracket, \llbracket y_i \rrbracket)_{i \in [d]}, \llbracket z \rrbracket$. This tuple must satisfy $z = \langle x, y \rangle$. We have the following protocol:

1. The parties get a random value $\epsilon \in \mathbb{F}^d$
2. They calculate locally their share of $\llbracket \alpha \rrbracket = \epsilon \circ \llbracket a \rrbracket + \llbracket x \rrbracket$ and $\llbracket \beta \rrbracket = \llbracket b \rrbracket + \llbracket y \rrbracket$
3. They broadcast their shares $\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket$ to obtain α and β
4. Each party calculates locally $\llbracket v \rrbracket = -\llbracket z \rrbracket + \langle \epsilon, \llbracket c \rrbracket \rangle + \langle \alpha, \llbracket y \rrbracket \rangle + \langle \beta, \llbracket x \rrbracket \rangle - \langle \alpha, \beta \rangle$
5. They broadcast $\llbracket v \rrbracket$ such that every party has v
6. The parties output *Accept* in case of $v = 0$ and *Reject* otherwise

Similar to the non-batching multi-party product verification protocol, each party will output *Accept* if the multiplication triples are correct. This can be seen by the following calculation:

$$\begin{aligned}
v =& -z + \langle \epsilon, c\rangle + \langle \alpha, y\rangle + \langle \beta, x\rangle - \langle \alpha, \beta\rangle \\
=& -\langle x, y\rangle + \langle \epsilon, \langle a, b\rangle\rangle + \langle \alpha, y\rangle + \langle \beta, x\rangle - \langle \alpha, \beta\rangle \\
=& \langle x, y\rangle + \langle \epsilon, \langle a, b\rangle\rangle + \langle \epsilon a + x, y\rangle + \langle b + y, x\rangle - \langle \epsilon a + x, b + y\rangle \\
=& -\langle x, y\rangle + (\langle \epsilon, a\rangle + \langle \epsilon, b\rangle) + (\langle \epsilon \circ a, y\rangle + \langle x, y\rangle) + (\langle b, x\rangle + \langle y, x\rangle) \\
& - (\langle \epsilon \circ a, b\rangle + \langle \epsilon \circ a, y\rangle + \langle x, b\rangle + \langle x, y\rangle) \\
=& -\langle x, y\rangle + \langle \epsilon, a\rangle + \langle \epsilon, b\rangle + \langle \epsilon \circ a, y\rangle + \langle x, y\rangle + \langle b, x\rangle + \langle y, x\rangle \\
& - \langle \epsilon \circ a, b\rangle - \langle \epsilon \circ a, y\rangle - \langle x, b\rangle - \langle x, y\rangle \\
=& (\langle x, y\rangle - \langle x, y\rangle) + \langle \epsilon, a\rangle + \langle \epsilon, b\rangle + \langle \epsilon \circ a, y\rangle + \langle x, y\rangle + \langle b, x\rangle \\
& + \langle y, x\rangle - \langle \epsilon \circ a, b\rangle - \langle \epsilon \circ a, y\rangle - \langle x, b\rangle - \langle x, y\rangle \\
=& 0 + \langle \epsilon, a\rangle + \langle \epsilon, b\rangle + 0 + 0 + \langle b, x\rangle + \langle y, x\rangle - \langle \epsilon \circ a, b\rangle - 0 - \langle x, b\rangle - \langle x, y\rangle \\
=& \langle \epsilon, a\rangle + \langle \epsilon, b\rangle + \langle b, x\rangle + \langle y, x\rangle - \langle \epsilon \circ a, b\rangle - \langle x, b\rangle - \langle x, y\rangle \\
=& 0.
\end{aligned}$$

We can now utilize the Lemma 7 from [KZ22] to define the probability that each party outputs*Accept* with at least one incorrect multiplication triple as $\frac{1}{|\mathbb{F}|^t}$.

Lemma 7: [KZ22]

If at least one of the two multiplication triples $((\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket), (\llbracket x_i \rrbracket, \llbracket y_i \rrbracket)_{i \in [t]}, \llbracket z \rrbracket)$ is incorrect, then the parties output *Accept* with a probability of at most $\frac{1}{|\mathbb{F}|^t}$.

With commitments, additive secret sharing, and multi-party computation in the Head. We will introduce the syndrome decoding problem, which forms the fundamental problem used in the MPCitH protocol. After that, we show the structure of the security proof used for our protocol in Section 7.3.

3.10 Syndrom Decoding

A common problem in code-based cryptographic systems is the *Syndrome decoding* (SD) problem. This problem comes from the vector called syndrome y, which refers to the product of a vector $\mathbf{x}$ and a parity-check matrix H. In this context, the parity-check matrix H is a matrix that defines the code by specifying which linear combinations of bits must add up to zero for $\mathbf{x}$ to be a valid codeword, effectively capturing the error-detecting structure of the code [VG23]. This syndrome reveals information about the error pattern and helps to determine the minimal set of errors needed to correct a received codeword. When the syndrome is equivalent to 0, then $\mathbf{x}$ is a code word. In addition, $\mathbf{x}$ is bound by the hamming weight $wt()$ and a security parameter w, such that $wt(\mathbf{x}) \leq w$, as proposed by [BMVT78]. Here, w is bound by the Gilbert-Varshamov radius. The Gilbert-Varshamov radius can be visualized as a sphere defined for every code word. Thus, if we obtain a code that resides within a sphere of a code word, it will correspond to this codeword with overwhelming probability. This allows for error correction because the syndrome does not need to be equivalent to the codeword but rather resides within the sphere of the codeword to be accepted. In order to maintain the error correction characteristic, no sphere can overlap with another because otherwise, an input that is within the intersection of two spheres cannot be uniquely assigned to a codeword. Furthermore, a code can be uniquely assigned to a code word as long as it resides within a sphere. However, if it resides outside, it belongs to one of the surrounding codeword spheres and is no longer uniquely identifiable. The problem can be defined more precisely as follows:

Syndrome Decoding

- **Input:** Parity-check matrix $\mathsf{H} \leftarrow \mathbb{F}_q^{(m-k)\times m}$ and a syndrome $\mathsf{y} \in \mathbb{F}_q^{m-k}$
- **Challenge:** Find a vector $\mathbf{x} \in \mathbb{F}_q^m$ with $wt(\mathbf{x}) \leq w$ and $\mathsf{H} \cdot \mathbf{x} = \mathsf{y}$

In this case, m describes the length of the solution $\mathbf{x}$, and k is a security parameter with $m > k$. Furthermore, H and $\mathbf{x}$ are drawn uniformly at random during the challenge generation, from which one calculates $\mathsf{y} = \mathsf{H} \cdot \mathbf{x}$. Looking at cryptographically relevant parameters for this problem, then there exists only one $\mathbf{x}'$ such that $wt(\mathbf{x}') \leq w$ and thus $\mathbf{x}' = \mathbf{x}$. In addition w is bound by the Gilbert-Varshamov radius $\tau r_{GV}(m, k)$ which is defined as:

$$w < \tau r_{GV}(m,k) \leftrightarrow \sum_{j=0}^{w-1} \binom{m}{j} \cdot (q-1)^j < q^{m-k} \quad \text{with} \quad q = |\mathbb{F}|$$

To solve the syndrome decoding problem, one can use an algorithm of the following two algorithm families: the *information set decoding* (ISD) or the *generalized birthday algorithms* (GBA) [BBC+19, CTS16]. These algorithms determine how the security parameters of the syndrome decoding problem are chosen, such that any algorithm in those families runs in a time greater than 2^λ to find an $\mathbf{x}$ that satisfies $\mathsf{y} = \mathsf{H} \cdot \mathbf{x}$. This is called λ-security.
In order to use the syndrome decoding as a signature scheme, we need to use the MPCitH protocol based on this problem, which allows us to use the Fiat-Shamir transformation to turn an MPCitH protocol into a signature scheme. We describe syndrome decoding in the head paradigm in the next section.

3.11 Zero-Knowledge Protocol for Syndrome Decoding

The zero-knowledge protocol for the *syndrome decoding* (SD) problem was first introduced by [FJR22] in 2022. We consider an instance of the SD problem (H, y) with a given solution $\mathbf{x} \in \mathbb{F}_{SD}^m$. Then, let $\mathbb{F}_{SD}$ be the field in which the instance is defined. After this, we can assume that H is in standard form without the loss of generality. This means that the parity-check matrix H consists of two matrices $\mathsf{H}' \in \mathbb{F}_{SD}^{(m-k)\times m}$ and I_{m-k} with $\mathsf{H} = (\mathsf{H}'|\mathsf{I}_{m-k})$, where I_{m-k} is the identity matrix of size $m-k$. Furthermore, m and k are selected as described in the previous section. This allows us to rewrite the solution $\mathbf{x}$ as $(\mathbf{x}_A, \mathbf{x}_B)$, where $\mathbf{x}_A$ contains the information for the multiplication with the parity-check matrix H' and $\mathbf{x}_B$ the necessary information for the identify matrix. We can therefore rewrite y as the linear relation:

$$\mathsf{y} = \mathsf{H}' \cdot \mathbf{x}_A + \mathbf{x}_B \tag{3.8}$$

Furthermore, because $\mathbf{x}_A$ corresponds to H' we can calculate $\mathbf{x}_B$ from y and H as $\mathbf{x}_B = \mathsf{y} - \mathsf{H} \cdot \mathbf{x}_A$. This also implies that we only need to send $|\mathbf{x}_A| = (k \cdot \log(|\mathbb{F}_{SD}|)$ bits to reveal the solution of the given instance (H, y). With this in mind, we can now define the MPC protocol for the syndrome decoding problem before we introduce the syndrome decoding in the head paradigm.

MPC protocol of the Syndrome Decoding Problem [FJR22]

Given the syndrome decoding problem instance described above, we consider a field extension $\mathbb{F}_{poly} \supseteq \mathbb{F}_{SD}$ such that $|\mathbb{F}_{SD}| \geq m$. Lets recall that

MPC protocol of the Syndrome Decoding Problem [FJR22]

m is the length of the solution vector $\mathbf{x}$ with $\mathbf{x} \in \mathbb{F}_{SD}^m$. We further denote $\phi : \mathbb{F}_{poly} \hookrightarrow \mathbb{F}_{SD}$ as the canoncial inclusion of $\mathbb{F}_{SD}$ into $\mathbb{F}_{poly}$, meaning that each element of $\mathbb{F}_{SD}$ is treated as an element of $\mathbb{F}_{poly}$. Furthermore, we define a bijection γ between $\{1, \ldots, |\mathbb{F}_{poly}|\}$ and $\mathbb{F}_{poly}$. This can be considered as having an index for each polynomial in $\mathbb{F}_{poly}$. However, because of the bijection given i as an index, we get the corresponding polynomial $\mathbf{P}$ via $\gamma(i)$ or the corresponding index i given a polynomial $\mathbf{P}$ via $\gamma(\mathbf{P})$. To ease this notation, we will write γ_i instead of $\gamma(i)$.
The MPC protocol must verify that the shares $\mathbf{x}$ are a valid solution by checking $\mathsf{y} = \mathsf{H} \cdot \mathbf{x}$ and $wt(\mathbf{x}) \leq w$. The input of this protocol is $[\![\mathbf{x}_A]\!]$; thus, we can build a correct sharing $[\![\mathbf{x}]\!]$ using the Equation 3.8 and will have $\mathsf{y} = \mathsf{H} \cdot \mathbf{x}$ by construction. After that, we need to prove that $wt(\mathbf{x}) \leq w$ using the multi-party product verification. This can be done using the following four polynomials where X is a variable over which the given polynomials are constructed:

- The first polynomial $\mathbf{S} \in \mathbb{F}_{poly}[X]$ satisfies:

  $$\forall i \in [m], \mathbf{S}(\gamma_i) = \phi(\mathbf{x}_i)$$

  and $deg(\mathbf{S}) \leq m - 1$. In other words, this polynomial contains the values $\mathbf{x}_i$ at its interpolation points γ_i because γ_i returns the polynomial with index i. Evaluating $\mathbf{S}$ on this polynomial gives us an interpolation point of $\mathbf{S}$ at point γ_i. This point is equal to $\phi(\mathbf{x}_i)$, which is the value of $\mathbf{x}$ at position i mapped to the polynomial field $\mathbb{F}_{poly}$. Thus, this polynomial $\mathbf{S}$ is unique and dependent on $\mathbf{x}$ such that it represents $\mathbf{x}$ in $\mathbb{F}_{poly}$.

- The second polynomial $\mathbf{Q} \in \mathbb{F}_{poly}[X]$ is defined as:

  $$\mathbf{Q}(X) := \prod_{i \in \mathbf{e}} (X - \gamma_i)$$

  where $\mathbf{e}$ is a vector containing specific indices of $\mathbf{x}$, with $\mathbf{e} \subset [m]$ such that $|\mathbf{e}| = w$ and $\{i \in [m] : \mathbf{x}_i \neq 0\} \subset \mathbf{e}$ and thus implying $deg(\mathbf{Q}) = w$. The product is then calculated over the difference between the variable X and the polynomial corresponding to the po-

MPC protocol of the Syndrome Decoding Problem [FJR22]

sition at which the solution is $\mathbf{x}_i = 1$, thus creating the polynomial $\mathbf{Q}$ with its coefficients being the masked polynomials corresponding to $\mathbf{x}_i = 1$. We can, therefore, say that the roots of $\mathbf{Q}(X)$ are the polynomials that can be mapped to the indices of $\mathbf{x}$ where $\mathbf{x}_i = 1$.

- The last polynomial $\mathbf{P} \in \mathbb{F}_{poly}[X]$ is defined as:

$$\mathbf{P} := (\mathbf{Q} \cdot \mathbf{S})/\mathbf{F} \quad \text{with} \quad \mathbf{F}(X) := \prod_{i=1} (X - \gamma_i).$$

  Here $\mathbf{F}(X)$ is similar to $\mathbf{Q}(X)$, but instead of having its roots at $\mathbf{x}_i = 1$, $\mathbf{F}$ has its roots at every position of $\mathbf{x}$ because the product is done over all indices corresponding to $\mathbf{x}_i \forall i \in [m]$.

After defining these four polynomials, we note a few additional important characteristics:

- We know that $\mathbf{Q}$ is a monic polynomial (polynomial where its leading coefficient is 1) of degree w and for every $i \in [m]$, we have:

$$\mathbf{x}_i \neq 0 \to i \in \mathbf{e} \to \mathbf{Q}(\gamma_i) = 0.$$

  In other words, the roots of $\mathbf{Q}$ correspond to the polynomial in $\mathbb{F}_{poly}$, which are connected to the non-zero positions of $\mathbf{x}$ via the bijection γ.

- The polynomial $\mathbf{F}$ divides $\mathbf{Q} \cdot \mathbf{S}$. For this, it is important to note that for every $i \in [m]$, we have:

$$(\mathbf{Q} \cdot \mathbf{S})(\gamma_i) = 0$$

  which is because of $\mathbf{S}(\gamma_i) \neq 0 \to \mathbf{x}_i \neq 0 \to \mathbf{Q}(\gamma_i) = 0$. In other words at every position of $\mathbf{x}$ where $\mathbf{x} = 0$ we know that $\mathbf{S}(\gamma_i) = 0$ and at every position of $\mathbf{x}$ with $\mathbf{x}_i = 1$ we know that $\mathbf{Q}(\gamma_i) = 0$, thus the product between them is 0. Therefore, $\mathbf{P}$ is well defined. $\mathbf{F}$ prevents the prover from defining $\mathbf{P}$ in a way that $(\mathbf{Q} \cdot \mathbf{S})(\gamma_i) = 0$ holds for many positions in $\mathbb{F}_{poly}$ which are outside of $\mathbb{F}_{SD}$ and thus allowing for a false positive acceptance of the verifier.

MPC protocol of the Syndrome Decoding Problem [FJR22]

- The polynomial $\mathbf{P}$ is of at most degree $\deg(\mathbf{P}) \leq w-1$, ensuring it acts as a lower-order corrective term in the equation $\mathbf{Q}\cdot\mathbf{S}-\mathbf{P}\cdot\mathbf{F}=0$, balancing the degrees on both sides. This degree constraint aligns with the requirement that the Hamming weight $wt(\mathbf{x}) \leq w$, as $\mathbf{P}$, must offset the degrees introduced by $\mathbf{Q}$ and $\mathbf{F}$ while keeping the verification equation consistent.

In order for the prover P to prove that she knows a solution $\mathbf{x}$ with $wt(\mathbf{x}) \leq w$ she needs to convince the verifier V that she has a polynomial $\mathbf{P}$ with $deg(\mathbf{P}) \leq w-1$ and another polynomial $\mathbf{Q}$ with $deg(\mathbf{Q}) = w$ such that $\mathbf{Q}\cdot\mathbf{S}-\mathbf{P}\cdot\mathbf{F}=0$. If $\mathbf{S}$ and $\mathbf{F}$ were built as described above, then V can deduce that:

$$\begin{aligned}&\forall i \in [m], (\mathbf{Q}\cdot\mathbf{S})(\gamma_i) = \mathbf{P}(\gamma_i)\cdot\mathbf{F}(\gamma_i) = 0\\ &\qquad \to \forall i \in [m], \mathbf{Q}(\gamma_i) = 0 \quad \text{or} \quad \mathbf{S}(\gamma_i) = \psi(\mathbf{x}_i) = 0.\end{aligned}$$

This deduction is crucial because it shows the verifier that for each $i \in [m]$, either $\mathbf{Q}(\gamma_i) = 0$ (indicating a zero in $\mathbf{x}$) or $\mathbf{S}(\gamma_i) = \psi(\mathbf{x}_i) = 0$ (indicating a non-zero entry in $\mathbf{x}$ at a root of $\mathbf{Q}$). This ensures that $wt(\mathbf{x}) \leq w$ verifies that the provers solution meets both the syndrome condition and the weight limit, as the MPC protocol requires.

Using the earlier described polynomials and the deduction above, we can explain the MPCitH paradigm to prove the following: Since the verifier knows that $\mathbf{Q}$ has at most w many roots, she can conclude that $\psi(\mathbf{x}_i) \neq 0$ is in at most w positions. Thus, she knows $wt(\mathbf{x}) \leq w$ and that the shared secret $\mathbf{x}$ is valid.

3.11.1 Syndrome Decoding in the Head (SDitH)

To describe the SDCitH paradigm, we will use an MPC protocol, which on input $\mathbf{x}, \mathbf{P}$ and $\mathbf{Q}$ outputs *Accept* if $\mathbf{Q}\cdot\mathbf{S}-\mathbf{P}\cdot\mathbf{F}=0$ holds and *Reject* otherwise, except with a small false positive probability. Each party's input is a share of $[\![x_A]\!]$, $[\![\mathbf{Q}]\!]$ and $[\![\mathbf{P}]\!]$. For the sharing of $\mathbf{x}$ and $\mathbf{P}$, we can use the additive secret sharing described in Section 3.6. This also works for the polynomial $\mathbf{P}$ because sharing a polynomial is defined as sharing its coefficients, which can be stored in a vector and shared in the same way as $\mathbf{x}$. Regarding $\mathbf{Q}$, we will additively share all

of its coefficients except for the leading one, which is publicly known because $\mathbf{Q}$ is a monic polynomial, and thus, its leading coefficient is 1. This further allows P to convince the verifier that $deg(\mathbf{Q}) = w$, which is essential, as otherwise a malicious prover $\tilde{\mathsf{P}}$ could use the zero polynomial for $\mathbf{Q}$, which would result in an arbitrary solution for $\mathbf{Q} \cdot \mathbf{S} - \mathbf{P} \cdot \mathbf{F} = 0$.
Given these inputs, the protocol calculates $\mathbf{S}$ from $\mathbf{x}_A$ and verifies $\mathbf{Q} \cdot \mathbf{S} = \mathbf{P} \cdot \mathbf{F}$ by evaluating both sides of the relation at t random points $z_1, \ldots, z_t$. We are following the Schwartz-Zippel lemma from Section 1 to know that the probability of observing $\mathbf{Q}(z_j) \cdot \mathbf{S}(z_j) = \mathbf{P}(z_j) \cdot \mathbf{F}(z_j)$ for all $j \in [t]$ is low. Furthermore, this includes that the larger the set from which the evaluation points z_j are sampled, the smaller the false positive probability p. Because of that, we sample the evaluation points from a field extension $\mathbb{F}_{points}$ of the field $\mathbb{F}_{poly}$, allowing us to sample more points for the evaluation and thus decreasing p. For this verification, we must use the batch product verification protocol described in Section 3.9 to prevent any information leakage of the secret $\mathbf{x}$. In short, the prover must first build t multiplication triples $(\llbracket a_j \rrbracket, \llbracket b_j \rrbracket, \llbracket c_j \rrbracket)$ for random elements $a_j, b_j, c_j \in \mathbb{F}_{points}$ which satisfy $a_j \cdot b_j = c_j$ for every $j \in [t]$. After that, she includes these in the party's inputs where each party obtains their corresponding share of the random elements $(\llbracket a_j \rrbracket, \llbracket b_j \rrbracket, \llbracket c_j \rrbracket)$.

Syndrome Decoding in the Head (SDitH)

With all this in mind, we can now describe the MPCitH protocol for the syndrome decoding.

1. Each party samples t evaluation points $z_1, \ldots, z_t \in \mathbb{F}_{points}$.
2. The parties compute locally $\llbracket \mathbf{x} \rrbracket$ from $\llbracket x_A \rrbracket$ using the Equation 3.8
3. After that the parties compute locally $\llbracket \mathbf{S} \rrbracket(z_j)$, $\llbracket \mathbf{Q} \rrbracket(z_j)$, as well as $\llbracket (\mathbf{P} \cdot \mathbf{F}) \rrbracket(z_j), \quad \forall j \in [t]$. We note here that $\llbracket \mathbf{S}(z_j) \rrbracket$ can be computed from $\llbracket \mathbf{x} \rrbracket$ by using the Lagrange interpolation:

$$\llbracket \mathbf{S} \rrbracket(z_j) = \sum_{i \in [m]} \llbracket \mathbf{x} \rrbracket \cdot \prod_{l \in [m], \neq i} \frac{z_j - \gamma_l}{\gamma_i - \gamma_l}.$$

   This calculation does not require any interaction between the parties. Furthermore we can simplify the computation of $\llbracket (\mathbf{P} \cdot \mathbf{F}) \rrbracket(z_j)$ to $\llbracket \mathbf{P} \rrbracket(t_j) \cdot \mathbf{F}(z_j)$ since $\mathbf{F}$ is publicly known.

Syndrome Decoding in the Head (SDitH)

4. The parties now run the product verification of the multiplication triples $[\![\mathbf{S}]\!](z_j), [\![\mathbf{Q}]\!](z_j), [\![(\mathbf{P}\cdot\mathbf{F})]\!](z_j)$ by sacrificing the triple $([\![a_j]\!], [\![b_j]\!], [\![c_j]\!])$ for every $j \in [t]$. For this, they perform the following steps:
   a) Each party samples a random $\epsilon_j \in \mathbb{F}_{points}$.
   b) Every party locally calculates
   $$[\![\alpha_j]\!] = \epsilon_j \cdot [\![\mathbf{Q}]\!](z_j) + [\![a_j]\!] \quad \text{and} \quad [\![\beta_j]\!] = [\![\mathbf{S}]\!](z_j) + [\![b_j]\!].$$
   c) After that they broadcast their shares $([\![\alpha_j]\!], [\![\beta_j]\!])$ to the other parties to obtain α_j and β_j.
   d) With the obtained α_j and β_j they can locally compute:
   $$[\![v_j]\!] = \epsilon_j \cdot [\![(\mathbf{P}\cdot\mathbf{F})]\!](z_j) - [\![c_j]\!] + \alpha \cdot [\![b_j]\!] + \beta_j \cdot [\![a_j]\!] - \alpha_j \cdot \beta_j.$$
   e) Finally each party broadcasts $[\![v_j]\!]$ such that every party obtains v_j.
5. If each party obtains v_j such that $v = 0$ then they return *Accept* otherwise they return *Reject*.

In this context, it is essential to note that it is not required to specify the random values z_j and ϵ_j as it is part of the challenge provided by the verifier V in the zero-knowledge setting.

The behavior of the SDitH protocol can be represented by a function $\mathcal{F}$, which takes $\mathbf{x}, \mathbf{Q}, \mathbf{P}$ and the t multiplication triples and returns probabilistically *Accept* or *Reject*. The probabilistic behavior of this function is due to the randomness contained in the product verification protocol, more precisely in the random evaluation points $z_1, \ldots, z_t$ and the random challenges $\epsilon_1, \ldots, \epsilon_t$.

The protocol will return *Accept* given a solution $\mathbf{x}$ that satisfies $wt(\mathbf{x}) \leq w$ and correctly computed polynomial $\mathbf{P}$ and $\mathbf{Q}$ with a probability of one. However, whenever any protocol inputs do not follow the above protocol or $\mathbf{x}$ does not satisfy $wt(\mathbf{x}) \leq w$, the protocol returns *Reject* with probability $1 - p$, where p is the false positive probability. Thus, the protocol follows the same output Table 3.1 as the MPCitH protocol described in Section 3.8.

Finally, we need to define the false positive probability p, which defines the probability of the MPC protocol returning *Accept* even though it was executed with a non-valid solution. For this, we denote the number of elements in $\mathbb{F}_{points}$ as $\Delta := |\mathbb{F}_{points}|$ and define a bad witness, representing a non-valid solution, as a solution $\mathbf{x}$ with $wt(\mathbf{x}) > w$. Furthermore, the false positive probability is relevant in the case of a falsely built polynomial $\mathbf{P}$ or $\mathbf{Q}$, which leads to $\mathbf{Q} \cdot \mathbf{S} \neq \mathbf{P} \cdot \mathbf{F}$. This relation is evaluated at t points; thus, the probability that the relation holds for i out of the t evaluations is:

$$Pr[\mathbf{Q}(z_j) \cdot \mathbf{S}(z_j) - \mathbf{P} \cdot \mathbf{F}(z_j) = 0] = \frac{\max_{l \leq m+w-1} \left\{ \binom{l}{i} \cdot \binom{\Delta - l}{t-i} \right\}}{\binom{\Delta}{t}}.$$

The maximum over $l \leq m + w - 1$ results from the maximum degree the polynomial $\mathbf{Q} \cdot \mathbf{S} - \mathbf{P} \cdot \mathbf{F}$ can have. This maximum degree holds due to an extension of the Schwartz-Zippel Lemma, as described in the Schwartz-Zippel variant 2 Lemma 2. The probability of obtaining *Accept* is $\left(\frac{1}{\Delta}\right)^{t-i}$. In other words, this describes the probability of getting $t - i$ false positives in the multiplication triple verification. Combining these probabilities results in a global false positive probability p of:

$$p \leq \sum_{i=0}^{t} \frac{\max_{l \leq m+w-1} \left\{ \binom{l}{i} \cdot \binom{\Delta - l}{t-i} \right\}}{\binom{\Delta}{t}} \cdot \left(\frac{1}{\Delta}\right)^{t-i}. \tag{3.9}$$

The false positive probability p plays an important role in describing the soundness error ε. Let us first recall that the soundness error describes the prover convincing the verifier without knowing the secret $\mathbf{x}$. Given the soundness error of the MPCitH protocol without the false positive event of $\frac{1}{N}$, we need to add the false positive event to the situation in which the prover did not cheat successfully. We denote this situation by the counter probability $1 - \frac{1}{N}$. After that, we can describe the soundness error as the counter probability of a malicious prover not being able to guess the correct share $(1 - \frac{1}{N})$ and the probability that no false positive event occurs $(1 - p)$. Thus, we have the following soundness error:

$$\varepsilon = 1 - \left(1 - \frac{1}{N}\right) \cdot (1 - p) = \frac{1}{N} + p - \frac{1}{N} \cdot p. \tag{3.10}$$

3.11.2 Communication Costs

Before we outline the proof for the soundness error, as well as the completeness and zero-knowledge property, we will look at the protocol's performance, which can be determined by its communication cost. For this, we exclude the communication costs of the challenges because their impact is comparably small and becomes irrelevant when turning the protocol into a non-interactive one, which is needed to obtain a signature scheme. Thus, we can split the communication costs into the following three parts:

- **Com** $:= h$, the hash of N commitments
- **Resp**$_1$ $:= h^{'}$, the hash of N hashes of the output of the MPC protocol
- **Resp**$_2$ $:= (state_i, p_i)_{i \neq i^*}$, **com**$_{i^*}$, $\{[\![\alpha_j]\!]_{i^*}\}, \{[\![\beta_j]\!]_{i^*}\}$ for all $j \in [t]$

Here $state_i$ either consists of a PRG seed of λ bits in case of $i \neq N$ or $state_i$ consists of the following four parts:

- A PRG seed of λ bits,
- the share of the secret $[\![x_A]\!]_N$ of $k \cdot \log_2(|\mathbb{F}_{SD}|)$ bits.
- The polynomial shares $[\![\mathbf{Q}]\!]_N$ and $[\![\mathbf{P}]\!]_N$, which are polynomials of degree w for the proof of $wt(\mathbf{x}) \leq w$. They are each $w \cdot \log(|\mathbb{F}_{poly}|)$ bits.
- And the shares $\{[\![c_j]\!]_N\}_{j \in [t]}$, which are t points in $\mathbb{F}_{points}$ and are each $|\mathbb{F}_{points}|$ bits. These $\{[\![c_j]\!]_N\}_{j \in [t]}$ are part of the batch product verification.

In addition, **com**$_{i^*}$ is a commitment of $2 \cdot \lambda$ bits and $\{[\![\alpha_j]\!]_{i^*}\}, \{[\![\beta_j]\!]_{i^*}\}$ for all $j \in [t]$ are elements of $|\mathbb{F}_{points}|$ bits. We can also describe the costs for the first two parts of the communication, namely the commitment **Com** and the first result **Resp**$_1$ with $2 \cdot \lambda$ bits each because the hash functions return a $2 \cdot \lambda$ long bit string. Thus, the resulting communications costs are:

$$\begin{aligned} costs_{comm} &= 4 \cdot \lambda && \textbf{Com} \text{ and } \mathrm{Res}_1 \\ &+ \lambda && \text{PRG seed} \\ &+ k \cdot \log_2(|\mathbb{F}_{SD}|) && [\![\mathbf{x}_A]\!] \\ &+ (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) && [\![\mathbf{Q}]\!]_N, [\![\mathbf{P}]\!]_N \\ &+ 2 \cdot t \cdot \log_2(|\mathbb{F}_{Points}|) && \{[\![\alpha_j]\!]_{i^*}, [\![\beta_j]\!]_{i^*}, [\![c_j]\!]_N\}, \forall j \in [t] \\ &+ 2 \cdot \lambda && \textbf{com}_{i^*} \end{aligned}$$

In order for us to achieve a soundness error of $2^{-\lambda}$, we can run the protocol τ times to achieve $\varepsilon^{\tau} \leq 2^{-\lambda}$. This would increase the communication cost by a factor of τ. However, we can save some cost by merging the hashes h and $h^{'}$ of all runs together and get a single instance of h and $h^{'}$. The resulting cost function is:

$$\begin{aligned} costs_{comm} = 4 \cdot \lambda + \tau \cdot (\lambda + k \cdot \log_2(|\mathbb{F}_{SD}|) + (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) \\ + 2 \cdot t \cdot \log_2(|\mathbb{F}_{Points}|) + 2 \cdot \lambda) \end{aligned} \tag{3.11}$$

3.11.3 Security Proof

In this section, we will outline the security proof for the syndrome decoding in the head protocol. This proof consists of three parts: completeness, honest-verifier zero-knowledge, and soundness. For this, we will follow the security proof in the paper [FJR22], which also provides detailed proofs of the zero-knowledge and soundness property in Appendices E and F.

Completeness

The completeness proof follows the completeness definition of 3.4. The proof of [FJR22] states that for any sampling of the random coins of the prover P and the verifier V, a prover that genuinely performs every step of the protocol will always convince the verifier and thus pass all checks in the protocol.

Honest-Verifier Zero-Knowledge

This proof follows the general idea of the honest-verifier zero-knowledge Definition 3.4 to show that an honest verifier does not gain any information about the secret. For this, [FJR22] assumes that the used PRG be (t, ε_{PRG})-secure as well as the commitment scheme be (t, ε_{Com})-hiding. Thus, there exists an efficient simulator $\mathcal{S}$, which outputs a $(t, \varepsilon_{PRG} + \varepsilon_{Com})$-secure transcript, that is indistinguishable from the transcript produced by the protocol. For this $\mathcal{S}$ is given a random challenge $i^{'}$. After that, they create a successful transcript of the SDitH protocol and modify it until they arrive at the transcript produced by $\mathcal{S}$ without adding information about $\mathbf{x}$. This shows that no information has been leaked.

Soundness

Similar to the first two proofs, this proof follows the idea of the soundness Definition 3.4 and the proof of [FJR22]. Let there be an efficient malicious prover $\tilde{\mathsf{P}}$ that tries to convince the verifier V on input (H, y) to know the secret $\mathbf{x}$. Assuming she is successful with a probability of:

$$\tilde{\varepsilon} := Pr[\langle \tilde{\mathsf{P}}, \mathsf{V}, (\rangle \mathsf{H}, \mathsf{y}) \rightarrow 1] > \varepsilon \tag{3.12}$$

For a soundness error of:

$$\varepsilon = 1 - \left(1 - \frac{1}{N}\right) \cdot (1-p) = \frac{1}{N} + p - \frac{1}{N} \cdot p \tag{3.13}$$

Furthermore, p is defined as in Equation 3.9. Then there exists a probabilistic extraction algorithm $\mathcal{E}$, which has access to a rewindable black box and either produces a witness $\mathbf{x}^{'}$, such that $\mathsf{H}\mathbf{x}^{'} = \mathsf{y}$ with $wt(\mathbf{x}^{'}) \leq w$ or a commit collision. The expected number of calls $\mathcal{E}$ makes is based on generating two distinct accepting transcripts, which depend on the difference between $\tilde{\varepsilon}$ and ε. The average number of calls is then upper bounded by:

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 - \tilde{\varepsilon} \cdot \frac{2 \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}\right) \tag{3.14}$$

As long as the malicious prover has a probability of finding either a suitable $\mathbf{x}'$ or a commit collision with a probability of less than ε in these given number of calls, the protocol is considered to be secure.

Soundness

This soundness error can be modified to fit as close to $\frac{1}{N}$ as we want by changing t and Δ accordingly.

By detailing essential lemmata, cryptographic definitions, zero-knowledge proofs, and multi-party computation fundamentals, we establish the theoretical framework necessary for the used optimizations and the following combination of them. We start with the three different optimizations, namely the small integer secret sharing (Chapter 4), the hypercube structure (Chapter 5), and the One Tree to Rule them All technique (Chapter 6) in the next three chapters. After that, we will dive into how these can be combined to form our protocol in Chapter 7. We start by describing Small Integer Secret Sharing.

4 Small Integer Sharing

The first optimization for the syndrome decoding in the head protocol we consider in this thesis is the *Small Integer Sharing* (SIS) or small integer secret sharing technique introduced by [FMRV22]. This optimization aims to reduce the communication costs of the MPCitH protocol by modifying the additive secret sharing. They introduce the technique in connection with the subset sum problem. We will, therefore, consider an instance of the subset sum problem of $(\mathbf{g}, h) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. From this we have $\mathbf{x} \in \{0,1\}^n$ and

$$\sum_{j=1}^{n} \mathbf{x}_j \cdot \mathbf{g}_j = h \mod q$$

They follow the MPCitH paradigm to build a zero-knowledge protocol to prove the knowledge of a solution $\mathbf{x}$ for the sub-set sum problem. For this, they build an MPC protocol with an honest-but-curious party taking the shares of the secret $\mathbf{x}$ as the input. This computation will only be successful in case of a valid $\mathbf{x}$. They utilize a modified version of the *additive secret sharing* (AddSS) described in Section 3.6. Let us recall that AddSS works by generating $N-1$ random shares and calculating the Nth share as the difference between the sum of the $N-1$ shares and the secret $\mathbf{x}$. These shares can be summed to obtain the secret. Instead they generate N random shares and calculate an auxiliary share $\Delta\mathbf{x}$ separately as:

$$\begin{cases} [\![\mathbf{x}]\!]_i & \overset{\$}{\leftarrow} (\mathbb{Z}_q)^n,\ \forall i \in [N] \\ \Delta\mathbf{x} & \leftarrow \mathbf{x} - \sum_{i=1}^{N} [\![\mathbf{x}]\!]_i \mod q. \end{cases} \tag{4.1}$$

This sharing introduces an additional share. However, because the communications costs of a sharing in the MPCitH paradigm are determined by the cost of sending the auxiliary value, here $\Delta\mathbf{x}$, they obtain a sharing cost of $n \cdot \log_2(q)$ bits.

To evaluate the still existing problem of their optimization, we give an example of the communications costs for this sharing. Given $n = 256$ and $q = 256$, then the sharing costs are $256 \cdot \log_2(256) = 2^{16}$ bits $= 8$KB. To achieve a soundness error of 2^{-128}, they need to repeat the protocol 16 times, and thus, the costs of sharing the secret $\mathbf{x}$ would already be 128KB, which is beyond practical. To reduce these communication costs, they sample the shares from a smaller number space, reducing the communication cost of each share. We will describe this process in more detail in the next section.

4.1 Sharing on Integers and Opening with Abort

This optimization further improves the communication costs of the secret sharing of the previous section. Note that $\mathbf{x}$ is a binary vector (i.e. $\mathbf{x} \in \{0,1\}^n$) in the subset sum scenario, as well as in the SD environment. This allows us to define the sharing over small integers instead of the binary space. Thus, this sharing is the same as defined in the previous section. However, instead of sampling each share $[\![\mathbf{x}]\!]_i$ over $\mathbb{Z}_q^n$, they introduce a new security parameter A, which defines the upper bound of the number space from which the shares can be sampled. This A can be relatively small and thus allows them to reduce the communication costs, which we will show later in this Section 4.1.3. The Small Integer Sharing is defined as follows:

Small Integer Sharing (SIS)

Given the secret $\mathbf{x}$ and a security parameter A, we can calculate the additive sharing in the following manner:

$$\begin{cases} [\![\mathbf{x}]\!]_i & \xleftarrow{\$} \{0,\dots,A-1\}^n \text{ for all } i \in [N] \\ \Delta\mathbf{x} & \leftarrow \mathbf{x} - \sum_{i=1}^{N} [\![\mathbf{x}]\!]_i \end{cases} \tag{4.2}$$

However, before we can examine the improvement, we need to address the information leakage that comes with this sharing step. Note that $\Delta\mathbf{x}$ is not in $\{0,\dots,A-1\}^n$, but rather in $\{0,\dots,N\cdot(A-1)\}^n$.

The information leak results from the differing distribution of $\Delta\mathbf{x}_j$ depending on whether $\mathbf{x}_j = 0$ or $\mathbf{x}_j = 1$. An illustration can be seen in the left distribution of Figure 4.1. To address this problem, they introduce an abort technique, which transforms the probability mass function of $\Delta\mathbf{x}_j$ back into a non-revealing state. Let us first look at the exploit that results from this changing distribution. Recall that the verifier V will ask the prover P at the end of the protocol to open

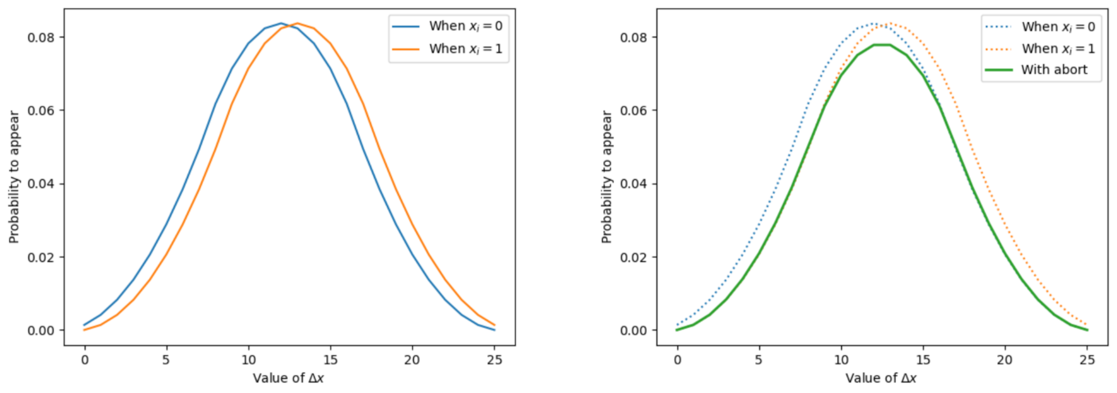


Figure 4.1: This figure shows the probability mass function of $\Delta\mathbf{x}_j$ depending on whether $\mathbf{x}_j = 0$ or $\mathbf{x}_j = 1$ (on the left) and additionally of $\Delta\mathbf{x}_j$ with the introduced abort technique (on the right). For both plots we set $N = 3$ and $A = 9$ [FMRV22].

all but one view. To identify this unopened view, we will denote the index of this view with i^*. Thus, the verifier will have access to the following views $\{[\![\mathbf{x}]\!]_i\}_{i \neq i^*}$ and $\Delta\mathbf{x}$.

Let us consider the simpler case of $n = 1$. We, thus, have $\mathbf{x} \in \{0, 1\}$ and $[\![\mathbf{x}]\!] \in \{0, \dots, A-1\}$, so a sharing through one integer. After opening all but one share, the verifier can compute the following:

$$\mathbf{x} - [\![\mathbf{x}]\!]_{i^*} \quad \text{via} \quad \Delta\mathbf{x} + \sum_{i \neq i^*} [\![\mathbf{x}]\!]_i$$

Note that $\Delta\mathbf{x}$ is the difference between the sum of the shares and the secret $\mathbf{x}$, which in turn allows the verifier to compute the difference between $\mathbf{x}$ and the unopened share $[\![x]\!]_{i^*}$. For a shorter notation, we will denote $Y = \mathbf{x} - [\![\mathbf{x}]\!]_{i^*}$, which represents the underlying random value over the uniform random sampling of $[\![\mathbf{x}]\!]_{i^*}$. From this, a dishonest verifier can calculate the probability of Y being either the highest $(-A+1)$ or lowest value (0) of the sharing:

$$Pr(Y = -A+1) = \begin{cases} \frac{1}{A} & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases} \quad \text{and} \quad Pr(Y = 1) = \begin{cases} 0 & \text{if } x = 0 \\ \frac{1}{A} & \text{if } x = 1. \end{cases}$$

In turn, we have the probability for every other value is:

$$Pr[Y = y] = \frac{1}{A} \quad \text{for every } y \in \{-A+2, \dots, 0\}.$$

For a better understanding of the information leakage, we will look at a small example:

SIS probability example

Firstly, we will consider the case $Y = -A + 1$ with $n = 1$ as above, set $\mathbf{x} = 1$, and choose $A = 9$. If we now sample the single sharing of $\mathbf{x}$ as $[\![\mathbf{x}]\!] \in \{0, \ldots, A-1\} = A - 1 = 8$ and calculate $\Delta\mathbf{x} = \mathbf{x} - [\![\mathbf{x}]\!] = 1 - 8 = 7$, we can see that the probability of $Y = -A + 1 = -9 + 1 = 8$ is 0. The reason for this is, that $[\![\mathbf{x}]\!]$ can at most be 8 and thus we calculate $Y = \mathbf{x} - [\![\mathbf{x}]\!]_{i^*}$ which results in $Y = 1 - 8 = 7 \neq 8$. For $\mathbf{x} = 0$ we can see that $[\![\mathbf{x}]\!]$ must be equal to $A - 1 = 8$ in order for $Y = -A + 1 = 8$, which has a probability of $\frac{1}{A}$. This shows the verifier that $\mathbf{x}_i$ must be 0.
Secondly, we will consider the same parameters as in the first case but look at $Y = 1$. Again, we start with $\mathbf{x} = 1$. In this case we can see that in order for Y to equal 1 we would need to sample $[\![\mathbf{x}]\!] = 0$ to obtain $\mathbf{x} - [\![\mathbf{x}]\!]_{i^*} = 1 - 0 = 1$, which has a probability of $\frac{1}{A}$. On the other hand, if we have $\mathbf{x} = 0$ we would need to sample $[\![\mathbf{x}]\!]$ as -1 because of $0 - (-1) = 1$, to achieve $Y = 1$. However, this is impossible by definition ($[\![\mathbf{x}]\!] \in \{0, \ldots, A - 1\}$). Thus, the verifier knows that $\mathbf{x}_i = 1$.

This example also gives a good intuition of the leaked information. More formally, if the verifier observes $\mathbf{x} - [\![\mathbf{x}]\!]_{i^*} = -A+1$ then she learns $(x, [\![\mathbf{x}]\!]_{i^*}) = (0, -A + 1)$. If she instead observes $\mathbf{x} - [\![\mathbf{x}]\!]_{i^*} = 1$ then she knows that $(x, [\![\mathbf{x}]\!]_{i^*}) = (1, 0)$. Thus, we can see that there are two scenarios in which the sharing reveals information about the secret. We abort in those two cases to ensure that no information is leaked. However, this must be done before revealing both $\{[\![\mathbf{x}]\!]_i\}_{i \neq i^*}$ and $\Delta\mathbf{x}$, but after committing to the shares in order for the protocol to preserve its soundness. This modification changes the distribution shown in Figure 4.1 (distribution on the right) and does not leak any information. Furthermore, the abort probability does not leak any information about the secret $\mathbf{x}$, as it is $\frac{1}{A}$ for both cases. If we consider the general case of $n \geq 1$, then the prover needs to apply the abort technique to every coordinate of $\mathbf{x}$, which can be shortened to the following three options:

- If there exists a $j \in [n]$ such that $\mathbf{x}_j = 1$ and the share $[\![\mathbf{x}_j]\!]_{i^*} = 0$, then the prover aborts.

- If there exists a $j \in [n]$ such that $\mathbf{x}_j = 0$ and the share $[\![\mathbf{x}_j]\!]_{i^*} = A - 1$, then the prover aborts.
- Otherwise, the prover continues.

In addition, we can give the probability to abort, which is also called the *rejection rate*. This rejection rate is the result of considering the counter probability of choosing any of the two cases above of $(1 - \frac{1}{A})$ for all n positions and the counter probability of having all n positions being an element that does not lead to an abort $\big(1 - (1 - \frac{1}{A})^n\big)$. It can be approximated by n/A as an upper bound if A is chosen large enough, because for large A we have $(1 - \frac{1}{A})^n \approx e^{-n \cdot \frac{1}{A}} \approx 1 - \frac{n}{A} \to 1 - (1 - \frac{n}{A}) = \frac{n}{A}$. This further highlights the necessity of choosing the security parameter A to be greater than n. More precisely we asymptotically have $A = \theta(n) = \theta(\lambda)$. This is an exponential improvement compared to the original $q = 2^{\theta(\lambda)}$. The rejection rate is therefore given by:

$$Pr[\text{abort}] = 1 - \left(1 - \frac{1}{A}\right)^n \leq \frac{n}{A} \tag{4.3}$$

With this asymptotic improvement, we will examine the communication costs of the Small Integer Sharing of $\mathbf{x}$. In the protocol, $\Delta\mathbf{x}$ is the highest costing vector of the sharing because it contains the difference between the secret and the sum of the shares, which results in the highest value elements. Thus, $\Delta\mathbf{x} \in \{-N \cdot (A-1) + 1, \ldots, 0\}$, which results in a sending cost of $n \cdot \log_2(N \cdot (A-1))$ bits. However, in order for the prover to save communication cost, she can make use of sending $\mathbf{x} - [\![\mathbf{x}]\!]_{i^*}$, which follows the relation $\mathbf{x} - [\![\mathbf{x}]\!]_{i^*} = \Delta\mathbf{x} + \sum_{i \neq i^*} [\![\mathbf{x}]\!]_i$ and reveals the same amount of information to the verifier. This underlying random value $(\mathbf{x} - [\![\mathbf{x}]\!]_{i^*})$ is uniformly distributed over $\{-A+2, \ldots, 0\}$ and thus takes $n \cdot \log_2(A-1)$ bits to send. The verifier can also recover $\Delta\mathbf{x}$ from $\mathbf{x} - [\![\mathbf{x}]\!]_{i^*}$ by computing

$$\Delta\mathbf{x} = (\mathbf{x} - [\![\mathbf{x}]\!]_{i^*}) - \sum_{i \neq i^*} [\![\mathbf{x}]\!]_i.$$

Another advantage of the Small Integer Sharing is the independence from q regarding its communication costs. This becomes clear if we look at the additive secret sharing, described in 4.1, which samples uniformly at random shares from $\mathbb{Z}_q^n$. Thus, we have n elements, which can be as big as q. By using the SIS, the largest element is upper bounded by A, which is much smaller than q; for example, $A = 2^{16}$ compared to $q = 2^{256}$. It is essential to note here that the selection of A depends on the trade-off between the communication costs and the rejection rate. Considering the previous A of 2^{16} and $n = 256$, we get a communication cost of 0.5KB with a rejection rate of 0.0038. However, if we reduce A to 2^8, we get a communication cost of 0.255KB, but the rejection rate spikes up to 0.63, which is impractical.

It is important to note that the abort event does not impact the protocol's soundness. Recall that the soundness error upper bounds the probability that someone who does not know the secret can complete the verifiers challenge. The abort event does not increase this probability because a malicious prover $\tilde{\mathsf{P}}$ can abort as many times as she wants, claiming that the sharing would leak information about the secret; this does not help to convince the verifier V of her knowledge. For example, $\tilde{\mathsf{P}}$ might sample a random party $i^{'}$ and generate a wrong share for $i^{'}$, which would appear correct. If V selects a different party than $i^{'}$, she might decide to abort, but this does not help her convince V and, further, does not increase the soundness error. This is because the malicious prover would need to guess the correct party a priori, which has the same probability as the verifier choosing to keep the wrong share hidden.

In order for us to use the SIS in the context of the subset sum problem through multi-party computation, we need to show that two properties are satisfied. The first is that the shared secret can be translated to the relation:

$$\sum_{j=1}^{n} \mathbf{x}_j \cdot \mathbf{g}_j = s \mod q$$

This can be translated to:

$$\sum_{j=1}^{n} [\![\mathbf{x}_j]\!] \cdot \mathbf{g}_j = [\![s]\!] \mod q$$

For this we need to compute a sharing of s, which can be done via $[\![s]\!]_i := \sum_{j=1}^{n} [\![\mathbf{x}_j]\!]_i \cdot \mathbf{g}_j \mod q$. Each party then commits these shares to the verifier. Similarly to the sharing of $\mathbf{x}$, the verifier can check whether $[\![s]\!]_i$ was computed correctly and whether the relation $\sum_{j=1}^{N} [\![s_j]\!] = [\![s]\!] \mod q$ holds or not. The second property shows that the sharing $[\![\mathbf{x}]\!]$ needs to encode a binary vector, the secret $\mathbf{x}$. This is not inherently given, as the sharing is defined over $\{0, \ldots, A-1\}$, and the correctness of the linear relation does not ensure that $\mathbf{x}$ is a binary vector. Thus, we need to add another step to prove this property, which we will detail in the next section.

4.1.1 Binarity Proof from Masking and Cut-and-Choose Strategy

This approach relies on the masking technique, which we combine with a cut-and-choose strategy to prove that $[\![\mathbf{x}]\!]$ encodes a binary vector. This works by generating a random vector $\mathbf{r} \in \{0,1\}^n$, which will be used instead of the original secret $\mathbf{x}$. Thus, we apply the sharing from Section 4.1 to the random vector $\mathbf{r}$ and have the prover calculate and commit the masked secret $\tilde{\mathbf{x}} := \mathbf{x} \oplus \mathbf{r} \in \{0,1\}^n$. Here $\oplus$ represents the XOR operator as stated in the notation Section 2.1. From this, we utilize the shares $[\![\mathbf{r}]\!]$ of $\mathbf{r}$ as the input of the MPC protocol, which allows us to be independent of the secret. Then, the parties can obtain a share of the secret through the masked secret $\tilde{\mathbf{x}}$ via the following linear relation in $[\![\mathbf{r}]\!]$:

$$[\![\mathbf{x}]\!] = (\mathbf{1} - \tilde{\mathbf{x}}) \circ [\![\mathbf{r}]\!] + \tilde{\mathbf{x}} \circ (\mathbf{0} - [\![\mathbf{r}]\!]) \tag{4.4}$$

The $\circ$ operator denotes the coordinate-wise multiplication. In addition the verifier can deduce the auxiliary value $\Delta\mathbf{x}$ from the shares $[\![\mathbf{r}]\!]$ and the corresponding $\Delta\mathbf{r}$ as follows:

$$\Delta\mathbf{x} = (\mathbf{1} - \tilde{\mathbf{x}}) \circ \Delta\mathbf{r} + \tilde{\mathbf{x}} \circ (\mathbf{1} - \Delta\mathbf{r}) \tag{4.5}$$

By masking and making the sharing independent of the secret, we can apply a cut-and-choose strategy to prove that $[\![\mathbf{r}]\!]$ encodes a binary vector $\mathbf{r}$, which implies that $\mathbf{x}$ is a binary vector through $\mathbf{x} = \tilde{\mathbf{x}} \oplus \mathbf{r}$. For the cut-and-choose strategy the prover generates multiple instances of the data that they want to prove is correct. After that, the verifier chooses a subset of these instances that the prover opens. Then, the verifier checks these opened instances, and if they match the given criteria, the verifier trusts that the unopened instances also satisfy the criteria.
In order for the cut-and-choose strategy to work for SIS, the prover P generates M many binary vectors $\mathbf{r}^l$, where $l \in \mathcal{L}$ with $\mathcal{L}$ being the corresponding challenge space. It calculates their corresponding shares $[\![\mathbf{r}^l]\!]$. These vectors are similar to the random shares of the AddSS in Section 3.6 in practice calculated through a tree-based pseudorandom generator, which we will look at later in Section 6. For now, we can consider this structure as an additional optimization, which allows us to reduce the number of shares sent. Now, P commits to these sharings, determines the masked vectors $\tilde{\mathbf{x}}^l = \mathbf{x} \oplus \mathbf{r}^l$, and commits those as well. Hereupon, the verifier asks the prover to open all the sharings

but one and checks whether they correspond to a binary vector $\mathbf{r}^l$. It is essential to note that none of the $\tilde{\mathbf{x}}$ vectors corresponding to any of the opened $\mathbf{r}^l$ are opened because the verifier would otherwise be able to recover the secret $\mathbf{x}$ through the equations above. This strategy provides a soundness error of $\frac{1}{M}$, which results in a combined soundness error for the entire protocol of

$$\varepsilon = \max\left\{\frac{1}{M}, \frac{1}{N}\right\}. \tag{4.6}$$

The communication cost of the protocol with the binarity proof from masking and cut-and-choose strategy in combination with a rejection rate of $1-(1-\frac{1}{A})^n$ as:

$$\begin{aligned} COST = 4 \cdot \lambda & \qquad \textbf{Com} \text{ and } \text{Res}_1 \\ + \lambda \cdot \log_2(M) & \qquad \text{cut-and-choose} \\ + n \cdot \log_2(A-1) & \qquad \mathbf{r} - [\![\mathbf{r}]\!]_{i^*} \\ + n & \qquad \tilde{\mathbf{x}} \\ + \lambda \cdot \log_2(N) & \qquad \text{Challenge } \mathcal{L} \\ + 2 \cdot \lambda & \qquad \textbf{com}_{i^*} \end{aligned}$$

In order to give a better intuition for this protocol, we provide the following example of the SIS and binarity proof from the masking and cut-and-choose strategy.

SIS example

Assume the number of shares $N = 2$, the number of parties $n = 2$, and the security parameter $A = 9$ for the number space of the Small Integer Sharing and the secret $\mathbf{x} = (0, 0, 1)$ with a length of $m = 3$. We now generate a random masking vector $\mathbf{r} \in \{0, 1\}^m = (1, 0, 1)$ and the two random shares for the mask $[\![\mathbf{r}]\!]_1 = (3, 5, 7), [\![\mathbf{r}]\!]_2 = (2, 3, 4)$. After this, we

SIS example

calculate the auxiliary vector for $\mathbf{r}$ via:

$$\begin{aligned}
\Delta\mathbf{r} &= \mathbf{r} - \sum_{i=1}^{m} [\![\mathbf{r}]\!] \\
&= \begin{pmatrix}1\\0\\1\end{pmatrix} - \left(\begin{pmatrix}3\\5\\7\end{pmatrix} + \begin{pmatrix}2\\3\\4\end{pmatrix}\right) \\
&= \begin{pmatrix}1\\0\\1\end{pmatrix} - \begin{pmatrix}5\\8\\11\end{pmatrix} = \begin{pmatrix}-4\\-8\\-10\end{pmatrix}
\end{aligned}$$

The prover P now sends $\tilde{\mathbf{x}} = x \oplus \mathbf{r}$ to each party as well as the corresponding share $[\![\mathbf{r}]\!]$ and the auxiliary value $\Delta\mathbf{r}$. From these each party can locally compute their share of $\mathbf{x}$ using Equation 4.4:

$$\begin{aligned}
[\![\mathbf{x}]\!]_1 &= (\mathbf{1} - \tilde{\mathbf{x}}) \circ [\![\mathbf{r}]\!]_1 + \tilde{\mathbf{x}} \circ (\mathbf{0} - [\![\mathbf{r}]\!]_1) \\
&= \left(\begin{pmatrix}1\\1\\1\end{pmatrix} - \begin{pmatrix}1\\0\\0\end{pmatrix}\right) \circ \begin{pmatrix}3\\5\\7\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} \circ \left(\begin{pmatrix}0\\0\\0\end{pmatrix} - \begin{pmatrix}3\\5\\7\end{pmatrix}\right) \\
&= \begin{pmatrix}0\\1\\1\end{pmatrix} \circ \begin{pmatrix}3\\5\\7\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} \circ \begin{pmatrix}-3\\-5\\-7\end{pmatrix} \\
&= \begin{pmatrix}0\\5\\7\end{pmatrix} + \begin{pmatrix}-3\\0\\0\end{pmatrix} \\
&= \begin{pmatrix}-3\\5\\7\end{pmatrix}
\end{aligned}$$

SIS example

$$\begin{aligned}
[\![\mathbf{x}]\!]_2 &= (\mathbf{1} - \tilde{\mathbf{x}}) \circ [\![\mathbf{r}]\!]_2 + \tilde{\mathbf{x}} \circ (\mathbf{0} - [\![\mathbf{r}]\!]_2) \\
&= \left(\begin{pmatrix}1\\1\\1\end{pmatrix} - \begin{pmatrix}1\\0\\0\end{pmatrix}\right) \circ \begin{pmatrix}2\\3\\4\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} \circ \left(\begin{pmatrix}0\\0\\0\end{pmatrix} - \begin{pmatrix}2\\3\\4\end{pmatrix}\right) \\
&= \begin{pmatrix}0\\1\\1\end{pmatrix} \circ \begin{pmatrix}2\\3\\4\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} \circ \begin{pmatrix}-2\\-3\\-4\end{pmatrix} \\
&= \begin{pmatrix}0\\3\\4\end{pmatrix} + \begin{pmatrix}-2\\0\\0\end{pmatrix} \\
&= \begin{pmatrix}-2\\3\\4\end{pmatrix}
\end{aligned}$$

Furthermore we can locally calculate $\Delta\mathbf{x}$ using Equation 4.5:

$$\begin{aligned}
\Delta\mathbf{x} &= (\mathbf{1} - \tilde{\mathbf{x}}) \circ \Delta\mathbf{r} + \tilde{\mathbf{x}} \circ (\mathbf{1} - \Delta\mathbf{r}) \\
&= \left(\begin{pmatrix}1\\1\\1\end{pmatrix} - \begin{pmatrix}1\\0\\0\end{pmatrix}\right) \circ \begin{pmatrix}-4\\-8\\-10\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} \circ \left(\begin{pmatrix}1\\1\\1\end{pmatrix} - \begin{pmatrix}-4\\-8\\-10\end{pmatrix}\right) \\
&= \begin{pmatrix}0\\1\\1\end{pmatrix} \circ \begin{pmatrix}-4\\-8\\-10\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} \circ \begin{pmatrix}5\\9\\11\end{pmatrix} \\
&= \begin{pmatrix}0\\-8\\-10\end{pmatrix} + \begin{pmatrix}5\\0\\0\end{pmatrix} \\
&= \begin{pmatrix}5\\-8\\-10\end{pmatrix}
\end{aligned}$$

After calculating the shares $[\![\mathbf{x}]\!]$ and the corresponding auxiliary value $\Delta\mathbf{x}$, we can calculate the secret $\mathbf{x}$ similarly to the additive secret sharing

SIS example

as follows:

$$\begin{aligned}
\mathbf{x} &= \Delta\mathbf{x} + \sum_{i=1}^{m} [\![\mathbf{x}]\!] \\
&= \begin{pmatrix} 5 \\ -8 \\ -10 \end{pmatrix} + \left(\begin{pmatrix} -3 \\ 5 \\ 7 \end{pmatrix} + \begin{pmatrix} -2 \\ 3 \\ 4 \end{pmatrix} \right) \\
&= \begin{pmatrix} 5 \\ -8 \\ -10 \end{pmatrix} + \begin{pmatrix} -5 \\ 8 \\ 11 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}
\end{aligned}$$

4.1.2 Proof

In order for us to prove the correctness of Small Integer Sharing with masking and cut-and-choose, we need to show that one can reconstruct the shares $\mathbf{x}$ and the corresponding auxiliary value $\Delta\mathbf{x}$ from the independent secret $\mathbf{r}$ and its shares, as well as the masked secret $\tilde{\mathbf{x}}$. The Equation 4.4 for this reconstruction in the original paper [FMRV22] is faulty; thus, we give the correctness proof of the reconstruction using Lemma 3.

SIS correctness proof

$$\mathbf{x} = \Delta\mathbf{x} + \sum_{i=1}^{m} [\![\mathbf{x}]\!]_i$$

$$= (\mathbf{1} - \tilde{\mathbf{x}}) \circ \Delta\mathbf{r} + \sum_{i=1}^{m} (\mathbf{1} - \tilde{\mathbf{x}}) \circ [\![\mathbf{r}]\!]_i + \tilde{\mathbf{x}} \circ (\mathbf{0} - [\![\mathbf{r}]\!]_i)$$

$$= \Delta\mathbf{r} - \tilde{\mathbf{x}} \circ \Delta\mathbf{r} + \tilde{\mathbf{x}} - \tilde{\mathbf{x}} \circ \Delta\mathbf{r} + (1 - \tilde{\mathbf{x}}) \circ \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i + \tilde{\mathbf{x}} \circ \left(\mathbf{0} - \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i\right)$$

$$= \Delta\mathbf{r} - 2 \cdot (\tilde{\mathbf{x}} \circ \Delta\mathbf{r}) + \tilde{\mathbf{x}} - \tilde{\mathbf{x}} \circ \Delta\mathbf{r} + \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i - \tilde{\mathbf{x}} \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i - \tilde{\mathbf{x}} \circ \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i$$

$$= \Delta\mathbf{r} - 2 \cdot (\tilde{\mathbf{x}} \circ \Delta\mathbf{r}) + \tilde{\mathbf{x}} - \tilde{\mathbf{x}} \circ \Delta\mathbf{r} + \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i - 2 \cdot \left(\tilde{\mathbf{x}} \circ \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i\right)$$

We know that $\mathbf{r} = \Delta\mathbf{r} + \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i$ and thus get:

$$= \mathbf{r} - 2 \cdot (\tilde{\mathbf{x}} \circ \Delta\mathbf{r}) + \tilde{\mathbf{x}} - 2 \left(\cdot \tilde{\mathbf{x}} \sum_{i=1}^{m} [\![\mathbf{r}]\!]_i\right)$$

We can rewrite the shares as follows $\sum_{i=1}^{m} [\![\mathbf{r}]\!]_i = \mathbf{r} - \Delta\mathbf{r}$

$$= \mathbf{r} - 2 \cdot (\tilde{\mathbf{x}} \circ \Delta\mathbf{r}) + \tilde{\mathbf{x}} - 2 \cdot (\tilde{\mathbf{x}} \circ (\mathbf{r} - \Delta\mathbf{r}))$$

$$= \mathbf{r} - 2 \cdot (\tilde{\mathbf{x}} \circ \Delta\mathbf{r}) + \tilde{\mathbf{x}} - 2 \cdot (\tilde{\mathbf{x}} \circ \mathbf{r}) + 2 \cdot (\tilde{\mathbf{x}} \circ \Delta\mathbf{r})$$

$$= \mathbf{r} + \tilde{\mathbf{x}} - 2 \cdot (\tilde{\mathbf{x}} \circ \mathbf{r})$$

Using the Equation 3.3 from Lemma 3

$$= (\mathbf{r} \oplus \tilde{\mathbf{x}}) + 2 \cdot (\mathbf{r} \wedge \tilde{\mathbf{x}}) - 2 \cdot (\tilde{\mathbf{x}} \circ \mathbf{r})$$

$$= \mathbf{x} + 2 \cdot (r \wedge \tilde{\mathbf{x}}) - 2 \cdot (\tilde{\mathbf{x}} \circ \mathbf{r})$$

Lastly we know that for binary vectors we have $\circ = \wedge$

$$= \mathbf{x} + 2 \cdot (\tilde{\mathbf{x}} \circ \mathbf{r}) - 2 \cdot (\tilde{\mathbf{x}} \circ \mathbf{r})$$

$$= \mathbf{x}$$

4.1.3 Performance Analysis

Before we dive into the security proof of the SIS protocol, we will give a high-level asymptotic complexity analysis. Given the security parameter λ, we will show that the protocol achieves a communication cost of $\theta(\lambda^2)$ and an asymp-

totic computation time of $\theta(\lambda^4)$.
Regarding the asymptotic communication cost of the binarity proof, Feneuil et al. [FMRV22] show that they achieve costs of

$$costs_{comm} = \theta(\lambda \log_2(A) + \lambda \log_2(N))$$

given the assumption that $M = N$ is optimal for the communication cost in connection with the stated soundness error in Equation 4.6. Furthermore, assuming a low constant rejection rate, achieved by choosing $A = \theta(n \cdot \tau) = \theta(\frac{\lambda^2}{\log_2(N)})$ one can calculate the communication cost for τ iterations through:

$$costs_{comm} = \theta\left(\lambda^2 \cdot \frac{\log_2(A)}{\log_2(N)} + \lambda^2\right) = \theta\left(\frac{\lambda^2}{\log_2(N)} \cdot \log_2(\frac{\lambda^2}{\log_2(N)}) + \lambda^2\right)$$

From this equation with $N = \theta(\lambda)$, we get an asymptotic communication cost of $\theta(\lambda^2)$.
The asymptotic computation time for each repetition results from the complexity of the multiplication between elements from $\mathbb{Z}_q$, which is the space of the subset sum problem, and elements from the space of the small integer problem A. The resulting computational costs are

$$costs_{comp} = \theta(N \cdot n \cdot \log_2(q) \cdot \log_2(A)).$$

Thus, we get an asymptotic computational cost for one repetition of $\theta(\lambda^3 {\cdot} \log_2(\lambda)$ and for τ repetition of $\theta(\lambda^4)$. With this in mind, we will continue with the security proof of the SIS protocol.
For a more detailed analysis of the communication and computational costs, see Section 3.5 of [FMRV22].

4.2 Security Proof: Cut-and-Choose Strategy

In this section, we give a high-level description of the security proof for the SIS protocol with masking and cut-and-choose strategy. We refer the interested reader to Section 4.2 of [FMRV22].
Note that the described protocol of Section 4.1 with cut-and-choose can be optimized by performing a global cut-and-choose strategy, which is needed for the security proof. Instead of τ executions of the cut-and-choose strategy, the prover generates M random vectors $\mathbf{r}$ with their related shares $[\![\mathbf{r}]\!]$ and $\tilde{\mathbf{x}}$.

This allows the verifier to ask to open all but τ many $\mathbf{r}$. This enables us to have τ many trusted $\mathbf{r}$, but in turn increases the soundness error to $\varepsilon = \max(\frac{1}{M-\tau}, \frac{1}{N})$.

Similar to the security proof of the syndrome decoding in Section 3.11.3, we will state the protocol's theorems for completeness, zero-knowledge, and soundness.

Completeness

Given an honest prover P and a solution $\mathbf{x}$ for the subset sum instance $(\mathbf{g}, h) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, then she convinces the verifier V with probability

$$\left(1 - \frac{1}{A}\right)^{n \cdot \tau}$$

Further, we need to denote the probability of the protocol aborting, which follows the rejection rate in Equation 4.3. Thus, the abort probability for any of the τ iteration is

$$Pr[\text{abort}] = 1 - \left(1 - \frac{1}{A}\right)^{n} \tag{4.7}$$

Proof: Given any sampling of the random coins of P and V, if the protocol's computation is executed honestly and there is no abort, V will pass all the checks. Consequently, the completeness probability is 1 minus the probability of an abort event given by Equation 4.7, which implies the theorem statement.

Honest Verifier Zero-Knowledge

Assuming the PRG used in the protocol is (s, ε_{PRG})-secure and the commitment scheme **com** is (s, ε_{com})-hiding. There is an efficient simulator $\mathcal{S}$ that, given random challenges $\mathfrak{J}$ and $\mathfrak{L}$, produces a transcript that is $(s, \tau \cdot \varepsilon_{PRG} + \tau\varepsilon_{com})$-indistinguishable from an actual transcript of the protocol.

Proof: Firstly, we show the independence of the secret $\mathbf{x}$ and some values and events that occur in the transcript. For this, we argue that the abort event is independent of the secret, i.e.

$$Pr[abort|\mathbf{x}] = Pr[abort]$$

Honest Verifier Zero-Knowledge

which ensures that the abortion of the protocol does not leak any information about the secret. Thus, we can denote the probability of an abort without the secret as the probability of the abort event with the secret $Pr[abort]$. This is detailed in Appendix E.1 of [FMRV22].
Secondly, we need to show that in case of no abort event, no additional values leak information about the secret, namely the value $\tilde{\mathbf{x}}$. We can argue that because the sent coordinates $\mathbf{r}^l - [\![\mathbf{r}^l]\!]_{i^*}$ are uniformly sampled from the distribution $\{-A+2, \dots, 0\}$ and $y^l = \mathbf{r}^l - [\![\mathbf{r}^l]\!]_{i^*}$, we know that they do not leak any information in connection with the random shares $\{[\![\mathbf{r}^l]\!]_i\}_{i \neq i^*}$. Following this, we know that $\mathbf{r}$ does not leak any information and is uniformly sampled over $\{0,1\}^n$. Thus, we know that $\tilde{\mathbf{x}}$ is independent of the secret $\mathbf{x}$.
After this, we can create a simulator $\mathcal{S}$ that simulates the protocol without knowing a correct solution $\mathbf{x}$ and has access to an oracle of a malicious probabilistic polynomial time verifier $\tilde{\mathsf{V}}$. This simulator then outputs a transcript that is independent of the transcript provided by a correct execution of the protocol of an honest prover.
For the detailed proof, see Appendix E.3 and F.3 of [FMRV22].

Soundness

Assuming that an efficient malicious prover $\tilde{\mathsf{P}}$ on input $(\mathbf{g}, h)$ convinces an honest verifier V on input $(\mathbf{g}, h)$ with a probability of

$$\tilde{\varepsilon} = Pr[\langle \tilde{\mathsf{P}}(\mathbf{g}, h), \mathsf{V}(\mathbf{g}, h) \rangle = 1] > \varepsilon.$$

We can give the soundness error as:

$$\varepsilon = \max_{M-\tau \le k \le M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot N^{k-M+\tau}} \right\}$$

where $\binom{k}{M-\tau} \cdot \binom{M}{M-\tau}^{-1}$ is the probability of the malicious prover $\tilde{\mathsf{P}}$ passing the cut-and-choose phase. The probability of passing the second phase (the MPC protocol) is represented by $\frac{1}{N^{k-M+\tau}}$, which is the result of having $k-M$ many dishonest vectors for each of the τ executions. In other words, it represents the probability that the honest verifier V does

Soundness

not pick the dishonest vectors in the opening phase of the protocol. Suppose the dishonest vectors are the hidden vectors of the MPC protocol. In that case, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ with rewindable black-box access to $\tilde{\mathsf{P}}$ that produces a solution $\mathbf{x} \in \{0,1\}^n$ with input $h = (\mathbf{g}, \mathbf{x})$ or a commitment collision. For this, the extractor will make, on average, a certain number of calls to $\tilde{\mathsf{P}}$. This expected number arises because two distinct accepting transcripts are required for successful extraction, meaning the required calls depend on the difference between the malicious prover's success probability $\tilde{\varepsilon}$ and the soundness threshold ε. The expected number of calls is given by:

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 - \tilde{\varepsilon} \cdot \frac{8 \cdot M}{\tilde{\varepsilon} - \varepsilon}\right)$$

The idea behind the extraction of the solution $\mathbf{x}$ is that given three transcripts of the protocol execution, with specific conditions, we can extract the solution using two extractors $\mathcal{E}_1, \mathcal{E}_2$. These two transcripts then need to satisfy the following:

- They are run on the same shares, and their commitments
- The first two transcripts T_1, T_2 have different hidden shares in the second challenge but have the same ones in the first challenge.
- The first two transcripts are success transcripts, meaning they passed all checks of the honest verifier.
- The hidden share of the third transcript in the first challenge is different from the hidden share of T_1 and T_2 of their first challenge.

We assume all the revealed shares between the transcripts are mutually consistent, as we would otherwise have a hash collision. Furthermore, transcript T_1 and T_2 differ in the second challenge but use the same initial shares. Thus, the hidden share of one transcript is not hidden in the other, and the extractor can access all shares and recover the solution candidate. The third transcript differs in the first challenge and allows the second extractor to recover the secret independent vector $\mathbf{r}$, following the same argumentation. This recovery is necessary to prove

Soundness

its binarity. Thus, we have shown that the three extractors can extract the secret $\mathbf{x}$ and the corresponding trusted vector $\mathbf{r}$.

In order to create these transcripts, we generate one successful transcript T_1. After that, we roll back the transcript to the point after the first commit, from where the second transcript is generated in a way that differs in the index of the hidden share in the second challenge. With these two transcripts, we again roll back to the point after the first commit and generate transcripts until we find one that differs from the first challenge to the first two. Thus, we have three transcripts that satisfy the necessary characteristics for the extraction.

For a detailed proof, check Appendix F.4 of [FMRV22]. Note that this proof is close to the proof of Appendix E in [FJR21], which gives a more detailed explanation of the same proof based on the syndrome decoding problem.

5 MPCitH with Hypercube Structure

The Small Integer Sharing optimization tackles the significant communication cost problem of code-based MPCitH protocols. However, there is also the problem of computation costs, which can be improved by the hypercube optimization introduced by Aguilar-Melchor et al. [AGH+22]. For this, we divide MPCitH protocols into two different phases, the online and the offline phase, where the offline phase encapsulates every computation, which is independent of the communication between the prover and the verifier. The online phase corresponds to the remaining calculations, which require communication between the calculations. In this context, the hypercube structure is applied to the offline phase. This optimization reduces computation costs without impacting the soundness of the protocol. It works by rearranging the shares onto a hypercube and executing the MPCitH computations on various combinations of the shares.

5.1 Rearranging shares into a Hypercube

In the MPCitH protocols described in Section 3.8, the share commitments consist of seeds for a PRG either of the $N-1$ first shares or, in the case of the small integer sharing the N shares, while the final share is calculated as the difference between the sum of shares and the secret. The prover then uses these shares to simulate the MPC protocol for all n parties to produce the corresponding communications and transcripts. After that, the verifier asks the prover to open $N-1$ of the commitments and performs the protocol for those $n-1$ parties again to check for consistency. Executing the protocol for each of the n parties is the optimization point of the hypercube technique. Instead of performing n executions, they rearrange the shares such that only $\log_2(n)+1$ many executions are needed while preserving the soundness error by using the same initial commitments.

Before we dive into the rearrangement, note that the additive secret sharing used in the MPCitH computations works independently of the sampling method of the individual shares as long as they add up to the secret of the zero-knowledge proof. This allows us to re-express the number of parties of the MPCitH protocol as $n = N_H^D$ parties, where N_H is the number of shares or parties per hypercube dimension and D is the number of hypercube dimensions. The shares of each of the D dimensions add up to the original secret. Thus, regardless of the additive secret sharing used, each dimension instance will be correct. It is even independent of the underlying MPC protocol.

First, we describe the general hypercube creation. Then, we will give an example of a two-dimensional hypercube, which is used to visualize the performance increase. After that, we will show an example of a three-dimensional hypercube to visualize the index generation, and finally, we give a practical example to provide a better intuition.

Genral Hypercube

The general hypercube is a cube where each side is N_H long and has a dimensionality of D. This allows it to contain $n = N_H^D$ many parties. These n parties are called *leaf-parties* and have a list of indices where each element in the list defines the share it belongs to in the corresponding dimension denoted by the position in the list. More formally the indices are $(i_1, \ldots, i_D) \in [N_H]^D$. For example, given $(0, 1)$ with $N_H = 2$, $D = 2$, and $N = 2^2 = 4$, the first index tells us that the share will be used to calculate the first share of the hypercube in the first dimension. The parties corresponding to the shares of the hypercube are called *main-parties*, and for each dimension $k \in [D]$, there is one MPC run between N_H of these main-parties. To index a main-party, we use the index k, as defined before, to determine the cube's dimension and utilize an additional index $j \in [N_H]$ to index the main-party of a this dimension k. Thus, each main-party is indexed by (k, j), where we select all leaf-party shares, which have the value j at their index k in the index list $(i_1, \ldots, i_D) \in [N_H]^D$. These shares are then added together to form the sharing of the corresponding main-party share. A resulting hypercube with its main-party share indices can be seen in Figure 5.1. In order to keep the amount of revealed information the same as in the original MPC protocol, the verifier will ask to open all but one of the leaf-parties, instead of the main-parties. This preserves the proof size while reducing the number of MPC calculations.

Genral Hypercube

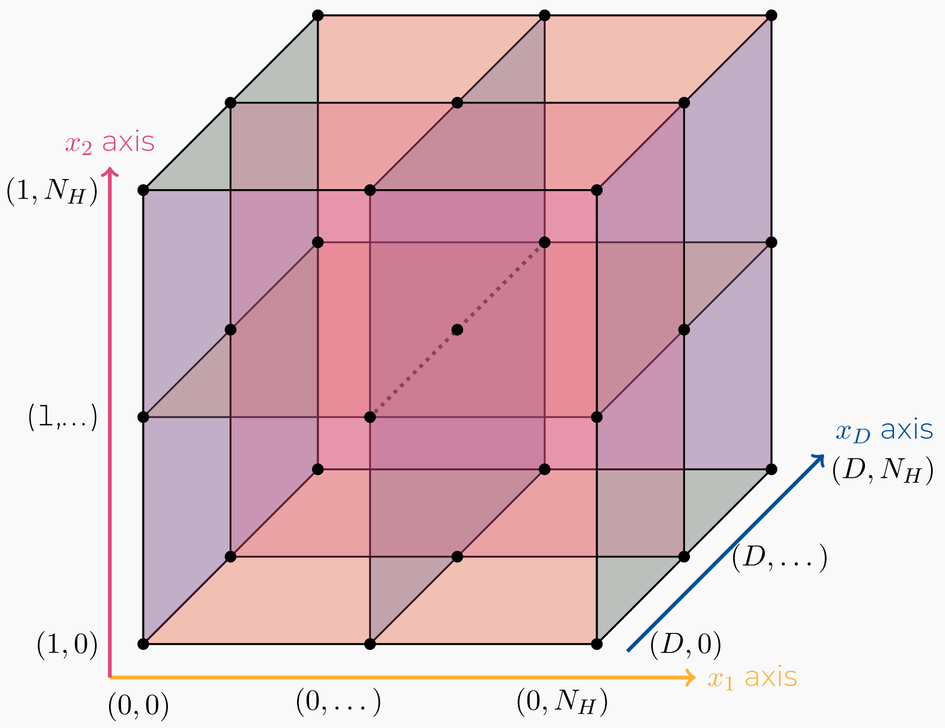


Figure 5.1: This figure shows a general hypercube and its main-party indices, where N_H represents the number of main-party shares per dimension and D is the number of dimensions in the cube. This hypercube consists of $N_H \cdot D$ many main-parties, which contain N_H^D many leaf-parties.

With this view of the general construction of the hypercube architecture, we will dive into a two-dimensional example to visualize the computational improvement and the calculations. Note that Aguilar-Melchor et al. [AGH+22] recommend choosing $N_H = 2$, which will give the highest optimization, as we will see after the example in the performance analysis.

2D Hypercube

Given $N_H = 2$ and $D = 2$, we know that the hypercube can store $N_H \cdot D = 2 \cdot 2 = 4$ many leaf parties, and thus we can assume a 4-party protocol with $n = N = 4$, meaning we have 4 parties with one share per party. For better readability, we will denote the leaf-party shares with ls_1, ls_2, ls_3 and the corresponding auxiliary value with aux, where $ls_0 + ls_1 + ls_2 +$

2D Hypercube

$aux = x$. Furthermore, we denote the main-party shares as m_0 and m_1 for the first dimension (x-axis) and n_0, n_1 for the second dimension (y-axis). As previously mentioned, we assign each leaf-party share a list of indices in $[N_H]^D$ and the main-parties their corresponding index tuple as follows:

$$\begin{aligned} ls_0 &\leftarrow (0,0) & m_0 &\leftarrow (0,0) \\ ls_1 &\leftarrow (0,1) & m_1 &\leftarrow (0,1) \\ ls_2 &\leftarrow (1,0) & n_0 &\leftarrow (1,0) \\ aux &\leftarrow (1,1) & n_1 &\leftarrow (1,1) \end{aligned}$$

From this, we can deduce that the leaf-party shares of ls_0 and ls_1 are used to calculate the first main-party share of m_0. The first element (0) of $m_0 \leftarrow (0,0)$ indicates that we look at the first index of the ls indices (denotes the position in the leaf-party index list). The second element (0) of this tuple denotes that we choose all leaf-shares with the first element equal to 0. Thus, we have the following main-party share calculations:

- $m_0 = ls_0 + ls_1$
- $m_1 = ls_2 + aux$
- $n_0 = ls_0 + ls_2$
- $n_1 = ls_1 + aux$

Figure 5.2 illustrates the arrangement of the main-party shares. In both hypercube dimensions, the shares of the main parties add up to the original secret x due to the associative and commutative nature of the additive secret sharing.

2D Hypercube

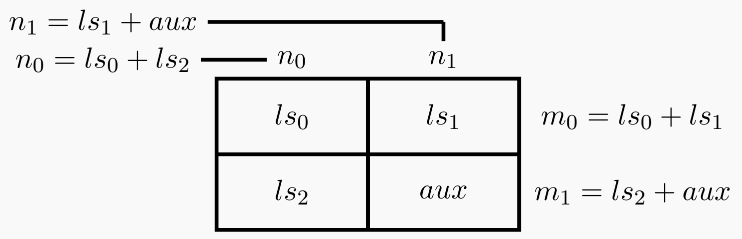


Figure 5.2: This figure adopted from [AGH+22] shows a simple two-dimensional example of the hypercube architecture with 4 main-party shares (m_0, m_1, n_0, n_1) containing 4 leaf-party shares (ls_0, ls_1, ls_2, aux).

This example shows that using a two-dimensional hypercube does not provide a performance improvement, as we have to generate 4 states, commit to 4 states, and perform 4 MPC calculations for both the original leaf-party and the main-party MPC execution. However, let us look at the common number of leaf-parties of $n = 256$. By keeping $N_H = 2$ and increasing the dimension D to 8, we can incorporate $n = 256$ leaf-party shares while performing only 2 MPC executions per dimension. This leads to $2+2+2+2+2+2+2+2 = 16$ MPC calculation using the hypercube technique instead of one MPC execution per leaf-party (256). This shows that the technique preserves the proof size, as the opening depends on the leaf-parties, while reducing the computation costs by a factor greater than 10. It further indicates that the security can be noticeably increased for the same computation cost.

An additional advantage of the hypercube structure is that each of the D executions relates to a specific aggregation of the same hypercube shares. As a result, each dimension sums up to the same plain text in the MPC algorithm. The prover can compute these plain-text values once; for example, by evaluating the first N_H parties, and for the remaining $D-1$ runs, we can derive the final share from the difference to the plain-text value. Thus, only $N_H - 1$ parties must be evaluated per run, resulting in $1 + (N_H - 1) \cdot D$ evaluations. One can, therefore, bypass most of the D execution. For example, in the instance of 256 parties, this optimization reduces the number of MPC computations to $2+1+1+1+1+1+1+1 = 9$, compared to 16 in the previous method and 256 in the original protocol.

This technique allows us to balance the MPC computation and the initial state commitment, where the MPC computation aspect is commonly much more expensive. We can increase the number of shares N (leaf-party shares) of the original protocol until both phases reach comparable cost. This only works because increasing the number of leaf-party shares increases the number of MPC executions on the main-party shares logarithmically (N_H^D). Thus, reaching parameters that were impossible in the original protocol and achieving previously unpractical security parameters is possible. In general, the more complex the functionality, the larger the hypercube improvements. Thus, we close a larger gap between the computation aspect and the initial costs.

For a better intuition of the improvement, we will provide the following practical example of a three-dimensional hypercube computation.

3D-Hypercube

Let us consider the secret $\mathbf{x} = (1,0,1)^T$ of length $m = 3$, the number of leaf-party shares $N = 8$ and the hypercube parameters $N_H = 2, D = \log_2(8) = 3$. At first, we calculate the shares using additive secret sharing as described in Section 3.6, which results in the following sharing:

$$\mathsf{ls}_0 = \begin{pmatrix}1\\0\\0\end{pmatrix} \quad \mathsf{ls}_4 = \begin{pmatrix}1\\0\\0\end{pmatrix}$$
$$\mathsf{ls}_1 = \begin{pmatrix}0\\1\\0\end{pmatrix} \quad \mathsf{ls}_5 = \begin{pmatrix}1\\0\\1\end{pmatrix}$$
$$\mathsf{ls}_2 = \begin{pmatrix}1\\0\\1\end{pmatrix} \quad \mathsf{ls}_6 = \begin{pmatrix}0\\1\\1\end{pmatrix}$$
$$\mathsf{ls}_3 = \begin{pmatrix}1\\1\\1\end{pmatrix} \quad \mathsf{aux} = \begin{pmatrix}-4\\-3\\-3\end{pmatrix}$$

After that, we assign each of these shares a list of indices. This list is $D = 3$ long, and each element is in $\{0,1\}$. This represents the main-party share it belongs to. Because $N_H = 2$, we can start at $(0,0,0)$ and increment the list by counting in binary, which gives us these indices:

3D-Hypercube

$$\begin{array}{ll} \mathsf{ls}_0 \leftarrow (0,0,0) & \mathsf{ls}_4 \leftarrow (1,0,0) \\ \mathsf{ls}_1 \leftarrow (0,0,1) & \mathsf{ls}_5 \leftarrow (1,0,1) \\ \mathsf{ls}_2 \leftarrow (0,1,0) & \mathsf{ls}_6 \leftarrow (1,1,0) \\ \mathsf{ls}_3 \leftarrow (0,1,1) & \mathsf{aux} \leftarrow (1,1,1) \end{array}$$

After this, we need to compute the indices for the main-parties, which are of the form (k, j), where $k \in [D] = \{0, 1, 2\}$ and $j \in [N] = (0, 1)$. In addition, we have $N_H \cdot D = 6$ many main-party shares, where $\mathsf{m}_i, \mathsf{n}_i, \mathsf{p}_i$ with $i \in \{0, 1\}$ represent the main-party shares of the first, second and third dimension respectively. Thus, we have:

$$\begin{array}{lll} \mathsf{m}_0 \leftarrow (0,0) & \mathsf{n}_0 \leftarrow (1,0) & \mathsf{p}_0 \leftarrow (2,0) \\ \mathsf{m}_1 \leftarrow (0,1) & \mathsf{n}_1 \leftarrow (1,1) & \mathsf{p}_1 \leftarrow (2,1) \end{array}$$

These indices indicate which leaf-party shares are needed to calculate the corresponding main-party share. As described earlier, the first element (k) determines the position in the list of indices in the leaf-party share index list, while the second element (j) defines the value it has to match. This means that for $m_0 \leftarrow (0, 0)$, we look at the index list's first element $(k = 0)$ and select every share for which the first element in its index list is equal to $j = 0$. These are marked red.

$$\begin{array}{ll} \mathsf{ls}_0 \leftarrow (0,0,0) & \mathsf{ls}_4 \leftarrow (1,0,0) \\ \mathsf{ls}_1 \leftarrow (0,0,1) & \mathsf{ls}_5 \leftarrow (1,0,1) \\ \mathsf{ls}_2 \leftarrow (0,1,0) & \mathsf{ls}_6 \leftarrow (1,1,0) \\ \mathsf{ls}_3 \leftarrow (0,1,1) & \mathsf{aux} \leftarrow (1,1,1) \end{array}$$

3D-Hypercube

Following this construction, we get the main-party shares as:

$$
\begin{aligned}
\mathsf{m}_0 &= ls_0 + ls_1 + ls_2 + ls_3 \\
&= \begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}0\\1\\0\end{pmatrix} + \begin{pmatrix}1\\0\\1\end{pmatrix} + \begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}3\\2\\2\end{pmatrix} \\
\mathsf{m}_1 &= ls_4 + ls_5 + ls_6 + aux \\
&= \begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}1\\0\\1\end{pmatrix} + \begin{pmatrix}0\\1\\1\end{pmatrix} + \begin{pmatrix}-4\\-3\\-3\end{pmatrix} = \begin{pmatrix}-2\\-2\\-1\end{pmatrix} \\
\mathsf{n}_0 &= ls_0 + ls_1 + ls_4 + ls_5 \\
&= \begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}0\\1\\0\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}1\\0\\1\end{pmatrix} = \begin{pmatrix}3\\1\\1\end{pmatrix} \\
\mathsf{n}_1 &= ls_2 + ls_3 + ls_6 + aux \\
&= \begin{pmatrix}1\\0\\1\end{pmatrix} + \begin{pmatrix}1\\1\\1\end{pmatrix} + \begin{pmatrix}0\\1\\1\end{pmatrix} + \begin{pmatrix}-4\\-3\\-3\end{pmatrix} = \begin{pmatrix}-2\\-1\\0\end{pmatrix} \\
\mathsf{p}_0 &= ls_0 + ls_2 + ls_4 + ls_6 \\
&= \begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}1\\0\\1\end{pmatrix} + \begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}0\\1\\1\end{pmatrix} = \begin{pmatrix}3\\1\\2\end{pmatrix} \\
\mathsf{p}_1 &= ls_1 + ls_3 + ls_5 + aux \\
&= \begin{pmatrix}0\\1\\0\end{pmatrix} + \begin{pmatrix}1\\1\\1\end{pmatrix} + \begin{pmatrix}1\\0\\1\end{pmatrix} + \begin{pmatrix}-4\\-3\\-3\end{pmatrix} = \begin{pmatrix}-2\\-1\\-1\end{pmatrix}
\end{aligned}
$$

If we sum these main-party shares up, we will get the original secret $\mathbf{x} = (1, 0, 1)^T$, which we can see in the subsequent computation.

3D-Hypercube

$$\begin{aligned}\mathbf{x}' &= \mathsf{m}_0 + \mathsf{m}_1 + \mathsf{n}_0 + \mathsf{n}_1 + \mathsf{p}_0 + \mathsf{p}_1 \\ &= \begin{pmatrix}3\\2\\2\end{pmatrix} + \begin{pmatrix}-2\\-2\\-1\end{pmatrix} + \begin{pmatrix}3\\1\\1\end{pmatrix} + \begin{pmatrix}-2\\-1\\0\end{pmatrix} + \begin{pmatrix}3\\1\\2\end{pmatrix} + \begin{pmatrix}-2\\-1\\-1\end{pmatrix} \\ &= \begin{pmatrix}1\\0\\1\end{pmatrix}\end{aligned}$$

Therefore, we can add the hypercube sharings exactly after the additive secret sharings and perform the MPC protocol as before without changing the functionality itself.

With this mode of operation of the hypercube structure in mind, we will analyze the performance of the SDitH protocol with the hypercube structure.

5.1.1 Performance Analysis

In this section, we provide the performance analysis of the hypercube protocol. This analysis is based on the syndrome decoding problem, where $\mathbf{x} \in \mathbb{F}_{SD}^m$ is the secret with a hamming weight of $wt(\mathbf{x}) \leq w$. For this, we analyze the zero-knowledge protocol, which follows the same operation path as the original SDitH protocol but uses the hypercube scheme described in the previous sections. It is important to note that similar to the original protocol, a few points of the protocol are ignored for the analysis, namely the challenges of the verifier because they do not impact the performance significantly. Thus, the communication cost is calculated via:

- **Com** $:= h$ of the N_H^D commitments
- **Resp**$_1$ $:= h'$ of the D hashes output from the MPC simulation
- **Resp**$_2$ $:= (state_{i_1,\ldots,i_D}, p_{i_1,\ldots,i_D}) \forall (i_1,\ldots,i_D) \neq (i_1^*,\ldots,i_D^*)$, $\mathbf{com}_{i_1^*,\ldots,i_D^*}$, $[\![\alpha]\!]_{i_1^*,\ldots,i_D^*}, [\![\beta]\!]_{i_1^*,\ldots,i_D^*}$

We can consider all but the final leaf-party share ($i' \in N_H^D$) with ($i' = (i_1,\ldots,i_D) \in \{1,\ldots,N_H^D\}$) of the hypercube, where each state of these leafs has a cost of the size of the seed of λ bits. For the final leaf, in addition to the seed $seed_{N_H^D}$, we need to take into account the auxiliary value. This auxiliary value consists of three parts, first the plain-text share $[\![x_A]\!]_{N_H^D}$, second the shares for the polynomial sharing, namely the two polynomials of degree $w-1$: $[\![\mathbf{Q}]\!]_{N_H^D}$ and $[\![\mathbf{P}]\!]_{N_H^D}$ and third the share $[\![c]\!]_{N_H^D}$ for the $t \in \mathbb{F}_{points}$.

Note that the pseudorandom generator used in the hypercube protocol follows a tree structure similar to the one used in the small integer sharing paper. We will give a more detailed explanation of this technique in Section 6. This technique affects the communication cost, as the parties are affected by the hypercube optimization, and thus, the component D is the number of seeds and commitment-randomness. Because of the TreePRG structure, we can reduce these random values to a sibling path in the tree of length D that connects $(state_{i_1^*,\dots,i_D^*}, p_{i_1^*,\dots,i_D^*})$ and the root. This path has a communication cost of $D \cdot \lambda \cdot \log_2(N)$ bits. The **com**$_{i_1^*,\dots,i_D^*}$ has the same costs as in SIS with $2 \cdot \lambda$ and $|\mathbb{F}_{points}|$ for all t shares $[\![\alpha]\!]_{i_1^*,\dots,i_D^*}, [\![\beta]\!]_{i_1^*,\dots,i_D^*}$.Combining all of these, we get:

$$
\begin{aligned}
COST = 4 \cdot \lambda & \qquad \textbf{Com} \text{ and } \mathrm{Res}_1 \\
+ D \cdot \lambda \cdot \log_2(N) & \qquad \text{PRG seed} \\
+ k \cdot \log_2(|\mathbb{F}_{SD}|) & \qquad [\![\mathbf{x}_A]\!]_{N_H^D} \\
+ (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) & \qquad [\![\mathbf{Q}]\!]_{N_H^D}, [\![\mathbf{P}]\!]_{N_H^D} \\
+ 2 \cdot t \cdot \log_2(|\mathbb{F}_{Points}|) & \qquad [\![\alpha]\!]_{i_1^*,\dots,i_D^*}, [\![\beta]\!]_{i_1^*,\dots,i_D^*} \\
+ 2 \cdot \lambda & \qquad \textbf{com}_{i_1^*,\dots,i_D^*}
\end{aligned}
$$

By performing the protocol τ times in parallel after transforming it into a non-interactive protocol, Aguilar-Melchor et al. [AGH+22] follow the common construction to achieve a soundness error of $2^{-\lambda}$. Similarly to the optimization in Section 4.1.3, we can further reduce the communication cost by merging h and h' of all τ runs. This results in a total communication cost of:

$$
\begin{aligned}
COST = 4 \cdot \lambda + \tau \cdot (D \cdot \lambda \cdot \log_2(N) + k \cdot \log_2(|\mathbb{F}_{SD}|) \\
+ (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) + 2 \cdot t \cdot \log_2(|\mathbb{F}_{Points}|) + 2 \cdot \lambda) \qquad (5.1)
\end{aligned}
$$

The resulting soundness error, where p is the false positive rate, is

$$\varepsilon = (p + (1-p) \cdot \frac{1}{N_H^D})^\tau.$$

Finally, we will go over the security proof of the SDitH protocol with the hypercube structure in the next section.

5.1.2 Security Proof

Let us give a high-level description of the security proof for the hypercube protocol for the syndrome decoding problem. This description follows the high-level description of the SDitH security proof in Section 3.11.3, as they are based on the same problem and therefore have a similar underlying hardness assumptions. In addition, we will denote a general prover as a prover who does not necessarily know the secret but reads and produces the same types of messages as an honest prover without needing to follow the protocol. We start by showing that an honest prover will be accepted with certainty if she knows the secret and, in turn, show that a malicious prover without the knowledge of the secret will only be accepted by the verifier with a probability of $\varepsilon \approx \frac{1}{N_H^D}$ (completeness and false acceptance rate). After that, we describe the proof that the protocol is a zero-knowledge proof by showing that the protocol can be simulated without the knowledge of the secret. This closely follows the proof in the original syndrome that decodes the zero-knowledge proof in Section 3.11.3 and the small integer that shares ZK in Section 4.2.

(Perfect) Completeness

Given an honest prover P with knowledge of the secret $\mathbf{x}$ who follows the protocol without any deviation, P will be accepted by a verifier V with a probability of 1.

Proof: This follows by design of the protocol. A prover that follows the protocol honestly will acquire computations for any choice of randomness, which will pass all of the verification checks of the verifier V by construction.

Before we continue with the zero-knowledge proof sketch, we will look at the protocol's false positive rate p. We need to show that it relates to the false positive probability 3.9 of the original SDitH protocol.

False Acceptance Rate

For this, we again define a malicious prover $\tilde{\mathsf{P}}$ as a prover without knowledge of the secret $\mathbf{x}$, who intends to create transcripts of the protocol that will be falsely accepted by the verifier. The malicious prover commits to a bad witness with $\mathbf{S}\cdot\mathbf{Q} \neq \mathbf{P}\cdot\mathbf{F}$ and is with probability less or equal $\varepsilon = p + \frac{1-p}{N_H^D}$ accepted by the verifier. By creating these polynomials without knowing the secret, she has a probability of $(1-p)$ that at least one of the challenge points is non-zero, as the verifier's random challenge points are unlikely to align with the false polynomials at every position, revealing the inconsistency and leading to rejection by the verifier. She creates these polynomials by following the technique of Section 3.9 and communicates them using the shares of α, β, v. Without loss of generality, we can assume that the prover cheats on one of the shares of α in one of the D independent SDitH runs, as cheating on either β or v would have the same probability. However, there are N_H many main-party shares in this SDitH run, giving her a success chance of $\frac{1}{N_H}$. In addition, each main party consists of N_H^{D-1} leaf-party shares, where all but one will be opened. Thus, the prover cannot cheat on more than one leaf-party share as the additional cheated share will be opened and identified. Each leaf-party share belongs to a single main-party share of each of the SDitH runs, which forces the prover to also cheat on any of the $[\![\alpha]\!]$ shares used in the other runs. The prover must, therefore, cheat on the share $[\![\alpha]\!]$ of a single leaf-party share cs.
The only way this is possible is that the uniformly random challenge of the verifier selects its hidden share i^* to be precisely the cheated leaf-party share cs. The probability of this happening is $\frac{1}{N_H^D}$, which is equivalent to the probability of cheating in the original SDitH protocol of $\frac{1}{N}$. In a non-false positive scenario, the prover $\tilde{\mathsf{P}}$ has, therefore, a chance of cheating of $\leq \frac{1}{N_H^D}$. This bounds the probability of a successfully cheating prover for the entire protocol without the knowledge of the secret $\mathbf{x}$ to be

$$\varepsilon = p + \frac{1-p}{N_H^D} \tag{5.2}$$

We continue with the security proof and show the honest-verifier zero-knowledge proof.

Honest-Verifier Zero-Knowledge (HVZK)

Assuming the protocol's *pseudorandom generator* (PRG) and the commitment **Com** are indistinguishable from the uniform random distribution, then the protocol is HVZK.

Proof: The proof follows the original honest-verifier zero-knowledge proof of the SDitH [FJR22]. Given a (t, ε_{PRG})-secure PRG and a (t, ε_{Com})-hiding commitment scheme, there exists an efficient simulator $\mathcal{S}$ that produces a $(t, \varepsilon_{PRG} + \varepsilon_{Com})$-secure transcript without the knowledge of the secret $\mathbf{x}$. This transcript is indistinguishable from the transcript produced by an honest protocol execution. This shows that a malicious verifier would not gain any information about the secret. The general mode of operation in this proof is to consider a simulator $\mathcal{S}$ that produces the transcript responses $(\textbf{Com}, \textbf{CH}_1, \textbf{Resp}_1, \textbf{CH}_2, \textbf{Resp}_2)$. After that, we create a so-called *true*-transcript, which is an execution of the protocol with an honest prover and verifier and the knowledge of the secret $\mathbf{x}$. This creates a correct transcript. Using this transcript, we alter the outputs section-by-section until we arrive at the simulator $\mathcal{S}$ and argue why the distribution did not change after each alteration. Thus, we prove that we can get from a *true*-transcript to a simulated one, which is only possible if no information of the secret is included in the transcript.

Soundness

Similar to the completeness and the HVZK proof, the proof of soundness follows the proofs of the original SDitH protocol. We assume an efficient prover $\tilde{\mathsf{P}}$ with knowledge of only (H, y) who can convince an honest verifier V with a probability of

$$\tilde{\varepsilon} = Pr[\langle \tilde{\mathsf{P}}, \mathsf{V} \rangle \rightarrow 1] > \varepsilon = (p + \frac{1-p}{N_H^D}). \quad (5.3)$$

Assuming the false positive rate p is bounded by Equation 3.9, there exists an extraction algorithm $\mathcal{E}$, which either produces a good witness $\mathbf{x}'$ such that $\mathsf{H} \cdot \mathbf{x}' = \mathsf{y}$ and $wt(\mathbf{x}') \leq w$, or a commitment collision, by mak-

Soundness

ing an average number of calls to $\tilde{P}$. This average number of calls arises because two distinct accepting transcripts are required for successful extraction; thus, the number of calls depends on the gap between the malicious prover's success probability $\tilde{\varepsilon}$ and the soundness threshold ε. The expected number of calls is then given by:

$$\frac{4}{\tilde{\varepsilon}-\varepsilon} \cdot \left(\frac{2 \cdot \tilde{\varepsilon} \cdot \ln(2)}{\tilde{\varepsilon}-\varepsilon} \right) \tag{5.4}$$

In the case of a prover cheating with a probability of $p \leq \varepsilon$, this counts as regular cheating and is not an issue for this proof or the security of the protocol in general.

Proof: The proof idea follows the proof of the original syndrome in the head soundness proof from [FJR22] closely. The only difference is related to the secret extraction, more precisely, the argument of why we can extract the secret. Let us recall that we run the SDitH protocol in D parallel executions, where each execution state is secret shared, as defined earlier. These shares are arranged in the hypercube geometry; thus, each secret share is used in D executions of the protocol. Moreover, the prover commits to these shares in her first message. We now need to explain the extraction of the secret $\mathbf{x}$.

The extraction operates similarly to the original extraction algorithm, where one can extract the secret if two accepting transcripts are given. These transcripts use the same initial commitments but differ in their second challenge. This means that both transcripts use the same secret shares but do not open the same shares in the second challenge. This, in turn, means that all shares are opened over the two transcripts, and thus, we can calculate the secret. This technique assumes that the used commitment scheme is binding. In the following paragraph, we will argue that this extraction is sufficient.

First, this argument holds for the original SDitH, but due to the different commitments of the hypercube structure, it does not automatically hold for this technique. However, by rephrasing the extraction condition, we can utilize the original proof for the hypercube. Here, we argue that the extraction is possible if we have access to all opened shares and

Soundness

their communication as long as each share is verified in at least one of the two accepting transcripts.
Because all but one share is opened in the first transcript, we know that all but this hidden share and their communication are verified in this transcript. The second transcript differs only in the second challenge, where all but one share and their communication is opened and verified. This means that the hidden share of the first transcript belongs to a different main party in the second transcript, in which this main-party share must be opened. Thus, the hidden share of the first transcript is verified using the second transcript. This also holds for the hidden share of the second transcript. Following the extraction argument of [FJR22], we now have access to all leaf-party shares, which have all been verified. Consequently, we can reconstruct the secret. After this, we only need to show that the extracted secret is a good witness, meaning that it solves the equation $\mathsf{H} \cdot \mathbf{x} = \mathsf{y}$ with $wt(\mathbf{x}) \leq w$. This precisely follows the SDitH proof.

We refer the reader to Section 3.5 (Security Proof) in [AGH+22] for the detailed proofs.
This concludes the soundness and security analysis for the hypercube optimization in MPCitH protocols. By efficiently structuring computation through the hypercube model, Aguilar-Melchor et al. [AGH+22] achieve substantial improvements in both performance and scalability without compromising security. With this foundation, we now turn to the role of tree-based pseudorandom generators, which further optimize the SDitH protocol regarding its communication costs.

6 Tree-based Pseudorandom Number Generator

In this section, we will describe the tree-based pseudorandom number generation optimization, which reduces the communication costs of secret sharing. We start by looking at *Goldreich, Goldwasser Micali* (GGM) tree-based pseudorandom number generators. After that, we present the One Tree to Rule them All structure, which optimizes the communication of secret shares over τ executions.

GGM Tree-based Pseudorandom Number Generator

The afford mentioned *tree based pseudorandom number generator* (treePRG) is, as the name suggests, a *pseudorandom number generator* (PRG) that utilizes a tree structure to derive its pseudorandom values. These structures are commonly based on the Goldreich, Goldwasser, Micali (GGM) [GGM86] trees and often fall under this name. We initialize the PRG with a random seed of length d, which will be used for the root tr of the tree. After that, the length-doubling method creates a new random number tl of length $2 \cdot d$. This technique can, for example, be a hash function th, with $th : \mathbb{Z}^d \rightarrow \mathbb{Z}^{2 \cdot d}$, such that it takes a number of length d as input and returns a number of length $2 \cdot d$. After that, tl is split in half, where one half is assigned to the root's left child and the other to the right child. This process is repeated until no more nodes are needed, creating a binary tree in which each node represents a random number.

The advantage of such a tree is that depending on the security needed, only the root or a few nodes must be communicated so that the other side can reconstruct the entire tree, reducing communication costs significantly. In the case of the MPCitH protocols, this allows the prover to send $\log_2(N)$ nodes, where N is the number of shares. It is important to note that the root cannot simply be shared, as this would allow the verifier to reconstruct all shares. Thus, the prover must send all nodes needed to construct the tree without revealing the hidden share path, commonly referred to as a sibling path. For this, the prover only uses the tree leafs as random values for the party shares because all but one share commonly needs to be opened, which would not be possible if non-leaf nodes were used. In order to provide a better intuition of this construction in the MPCitH context, we will give an example of using a TreePRG in the context of an MPCitH protocol.

TreePRG for MPCitH

Let us consider an instance of an MPCitH run, with $n = 9$ parties and $N = 9$ shares, utilizing the additive secret sharing from Section 3.6. Therefore, we need $N - 1 = 8$ uniformly random values for the sharing and the auxiliary value, which will be used for the 9th share. These 8 shares can be generated using the treePRG technique as described in Definition 6 until the tree has 8 leafs, one for each share. In this case, we utilize every leaf of the tree below (Figure 6.1) as a random share and commit to it during the initial communication. After receiving the challenge from the verifier, we know that all but the 14th node (hidden share) needs to be opened. In order to prevent the opening of the hidden share, we can look at the sibling path marked in red. This path tells us that the nodes $1, 3, 7$ and 14 cannot be revealed to the verifier as revealing any of them would allow the verifier to reconstruct the 14 node by following the construction described earlier. However, every parent node that is not part of this path can be used, resulting in sending nodes $2, 6$ and 15. In contrast to the original sharing, which would require the prover to send all leaf nodes ($8 - 15$), she only needs to send $\log_2(N) = \log_2(9) = 3$ nodes.

TreePRG for MPCitH

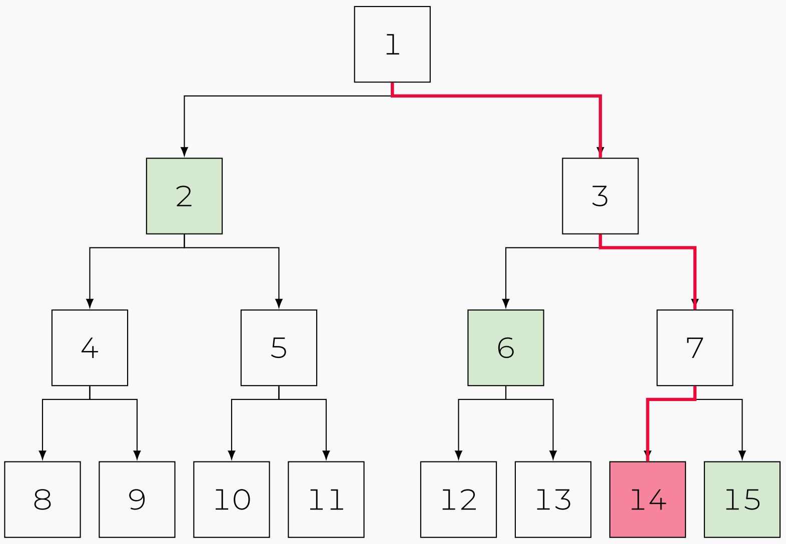


Figure 6.1: This figure shows an example treePRG of 15 total nodes, where nodes 8 to 15 can be used for the additive or small integer secret sharing. The red path $(1, 3, 7, 14)$ represents the nodes that need to be hidden from the verifier to prevent the verifier from obtaining the hidden node 14. Thus, the green nodes $(2, 6, 15)$ represent the nodes that are sent to the verifier and are needed to reconstruct the tree without revealing the hidden nodes.

6.1 One Tree to Rule them All

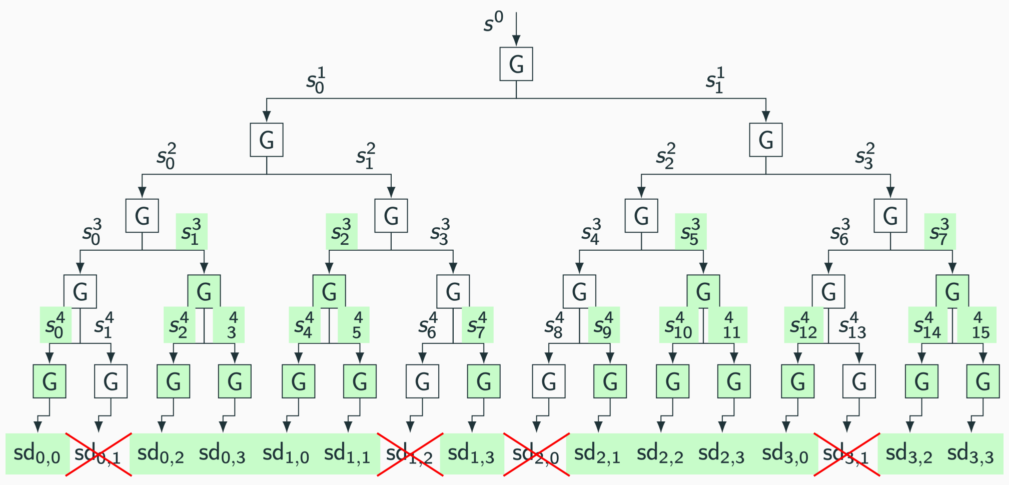
We introduce an optimization of the treePRG for a single run of an MPCitH instance in mind; [BBM+24] describes a technique to utilize the treePRG structure to reduce the communication for all τ executions of the MPCitH protocol. This technique introduces a structure called the *One Tree to Rule them All*, and we shortened the name to *OneTree*.

OneTree

The OneTree combines all τ treePRGs into a single GGM tree. They argue that opening all but τ leafs of the OneTree reduces, on average, the number of nodes that must be revealed to the verifier because some of the original sibling paths of the τ treePRGs merge relatively close to the leafs in the OneTree.

OneTree

For this merging to occur, they map the entries of the single trees into the big tree in an interleaving fashion. This can be seen in Figure 6.3. The first τ leafs of the OneTree correspond to the first element of the τ commitments. This means that the first share of each MPCitH run is used for the tree's first τ leafs. Then, the second share of the τ commits corresponds to the next τ leafs of the OneTree. Following this construction, all leafs/shares of the τ GGM trees are mapped to the OneTree. Simply merging all GGM trees into a single one, as visualized in Figure 6.2, is detrimental to the optimization because none of the sibling paths of the hidden shares would merge close to a leaf of the OneTree. Thus, there would be no reduction in the number of nodes that need to be sent to the verifier. This can also be seen in Figure 6.2.



s^0
G
s^1_0
s^1_1
s^2_0
s^2_1
s^2_2
s^2_3
s^3_0 s^3_1 s^3_2 s^3_3 s^3_4 s^3_5 s^3_6 s^3_7
s^4_0 s^4_1 s^4_2 s^4_3 s^4_4 s^4_5 s^4_6 s^4_7 s^4_8 s^4_9 s^4_{10} s^4_{11} s^4_{12} s^4_{13} s^4_{14} s^4_{15}
$sd_{0,0}$ $sd_{0,1}$ $sd_{0,2}$ $sd_{0,3}$ $sd_{1,0}$ $sd_{1,1}$ $sd_{1,2}$ $sd_{1,3}$ $sd_{2,0}$ $sd_{2,1}$ $sd_{2,2}$ $sd_{2,3}$ $sd_{3,0}$ $sd_{3,1}$ $sd_{3,2}$ $sd_{3,3}$

Figure 6.2: In this figure, we can see τ GGM trees, which have been combined into a single tree without using the interleaving combination described in Section 6.1. Thus, it shows that we need to reveal 8 nodes of the tree (highest green nodes) in order to open all requested shares for the verifier. The ✕ marks the hidden nodes for each of the τ TreePRGs. This figure was taken from [BB24].

OneTree

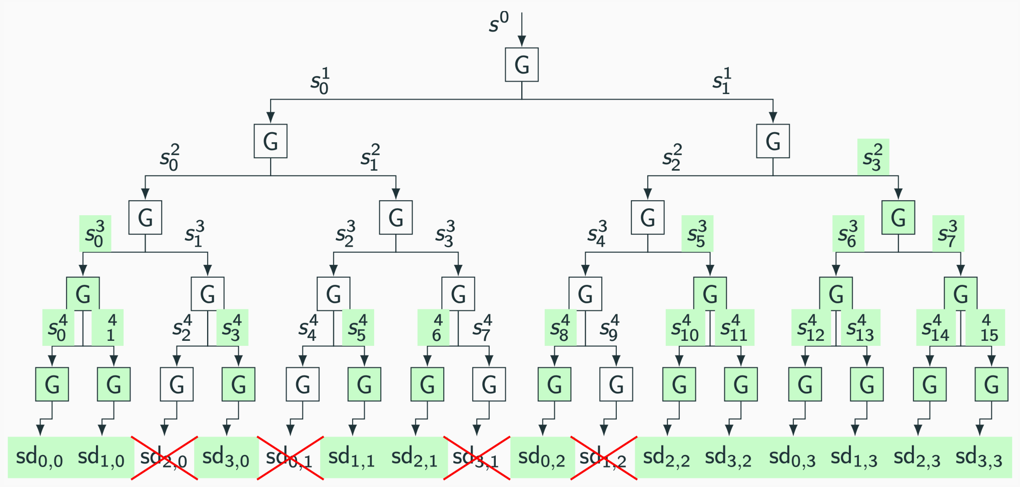

s^0
G
s^1_0
s^1_1
s^2_0
s^2_1
s^2_2
s^2_3
s^3_0 s^3_1 s^3_2 s^3_3 s^3_4 s^3_5 s^3_6 s^3_7
s^4_0 … s^4_{15}
sd0,0 sd1,0 sd2,0 sd3,0 sd0,1 sd1,1 sd2,1 sd3,1 sd0,2 sd1,2 sd2,2 sd3,2 sd0,3 sd1,3 sd2,3 sd3,3

Figure 6.3: This figure shows an example of the improvements of the OneTree structure by interleaving the nodes of the τ TreePRGs, such that the first τ nodes of the OneTree correspond to the first commitment of each of the τ MPCitH executions corresponding to these trees. Compared to the τ treePRGs of Figure 6.2, which were combined without the interleaving technique, we need to reveal 7 instead of 8 nodes (highest green nodes) to the verifier in order for her to reconstruct the necessary path to the opened shares. Again, ✗ marks the hidden nodes for each of the τ TreePRGs. This figure is taken from [BB24].

In [BB24], the average optimization of this technique is not provided, but we can calculate it as follows.

Assuming we have a OneTree that was constructed as described above, we need to calculate the probability that τ leafs have a path that merges close to any other leaf. To do this, we can have a closer look at the problem. The hidden shares, which are in different sub-trees, impact the optimization negatively the further up the tree they force a split. More precisely, we have τ hidden shares, so the worst case is that at depth $\log_2(\tau)$, every share resides in a unique sub-tree, which would result in no optimization. However, for every shared sub-tree, we gain one seed point that does not need to be sent at this height. Consequently, we can calculate the probability for this sharing of sub-trees at this height, where hs is the number of sub-trees that share a hidden share. Firstly, we need to consider the number of ways hs of the τ unique values can be placed on $\tau - hs$ positions. This essentially results from selecting hs hidden shares and placing them into a unique sub-tree of the OneTree, which can be calculated via $\frac{\tau!}{(\tau-hs)!}$. Secondly, we need to consider the selection of the remaining hidden shares in a way that they are in any of the already selected sub-trees, represented by $hs^{\tau-hs}$. We are thirdly dividing this by the total number of permutations of the sub-tree selection τ^{τ} gives us the probability of the OneTree structure optimizing the sharing. In other words, the probability of only needing hs of the τ bigger sub-trees. The final equation is as follows:

$$Pr[\text{sibling path merging}] = \frac{\tau! \cdot hs^{\tau-hs}}{(\tau - hs)! \cdot \tau^{\tau}} \tag{6.1}$$

Using this probability, we can calculate the percentile average of sub-trees needed. On average, sampled over $\tau \in \{2, \ldots, 128\}$ this results in needing $\approx 63\%$ of the sub-tress and thus an optimization of $\approx 37\%$. We, therefore, can assume an upper bound for the communication cost optimization by a factor of 0.4 or using only 0.6 times the number of nodes revealed over τ treePRGs.

7 Methodology

After introducing the different optimization strategies in the previous sections, we will describe how these can be combined to reduce the communication and computational costs of the original syndrome decoding in the head protocol. First, we implement these protocols, then analyze the new protocol's performance, and finally, we conclude with the security proof.

7.1 SIS Hypercube Tree

The protocol of the combined optimization follows the structure of the Small Integer Sharing and hypercube optimization, where we can combine the different optimizations by following the order of the previous sections, starting with SIS, followed by the hypercube structure, ending with the OneTree architecture.

7.1.1 Small Integer Sharing for Syndrome Decoding

Let us start with the Small Integer Sharing (SIS), which we must first convert to the different underlying problem, the syndrome decoding. The SIS was described on the sub-set sum problem, where the huge number space causes the main communication cost, which is not the case for the syndrome decoding problem. However, let us first recall the subset sum problem. Given a vector $\mathbf{g} \in \mathbb{Z}_q^n$ and a number $h \in \mathbb{Z}_q$ one must find a solution vector $\mathbf{x} \in \{0,1\}^n$ such that:

$$\sum_{j=1}^{n} \mathbf{x}_j \cdot \mathbf{g}_j = h \mod q$$

The accommodating additive secret sharing for the secret $\mathbf{x}$ is defined over the number space $\mathbb{Z}_q$, which results in high communication costs, as the q needs to be chosen big enough to ensure the hardness of the problem and thus in the context of the MPCitH protocol the required soundness error. In addition, the large number space increases the collision resistance, which impacts the protocol's security. The size of the q used in the additive secret sharing is tackled by the SIS optimization. It is replaced by a smaller security parameter A in connection with modifying the additive secret sharing. Here, N instead of $N-1$ shares are chosen uniformly random, and an additional auxiliary vector $\Delta\mathbf{x}$ is introduced. As described in Section 4.1.3, this works well but cannot be transferred directly to the syndrome decoding (SD) problem because the communication costs of the syndrome decoding problem stem from the size of the vectors instead of the number space in the sub-set sum problem.

Next, recall the syndrome decoding problem. Given a matrix $\mathsf{H} \in \mathbb{F}_{SD}^{(m-k)\times m}$ and a resulting vector $\mathsf{y} \in \mathbb{F}_q^{m-k}$, find a vector $\mathbf{x} \in \{0,1\}^m$ such that $\mathsf{H} \cdot \mathbf{x} = \mathsf{y}$ and the hamming weight of $\mathbf{x}$ is $wt(\mathbf{x}) \leq w$. Here, $m \in \mathbb{F}_q$ defines the length of the solution vector $\mathbf{x}$ as well as the size of the matrix H in combination with the security parameter k. This k is chosen big enough that the hardness of the problem is preserved by determining the number of solutions for the problem. The security parameter w further impacts the hardness of the problem by reducing the number of solutions.

At first glance, it seems like we can directly apply the Small Integer Sharing to the SDitH protocol. However, for a commonly chosen modulo q for the syndrome decoding problem, of either 2 or $2^8 = 256$, there is little to no improvement to be gained because q is already smaller than the number space of the SIS. However, the communication cost defining parameter in the syndrome decoding problem is the length of the solution vector $\mathbf{x}$ and, in turn, the resulting vector y and the matrix H. To reduce the dimension of the vector, it is impossible to reduce the length in the additive sharing, as $\mathbf{x}$ is commonly a binary vector, and thus, reducing its length significantly is detrimental to the security of the protocol. Nevertheless, by introducing a larger number space, the length restriction no longer binds us. This allows us to reduce the length of each share without allowing an attacker to guess valid shares or, more precisely, the unopened share in the context of the SDitH technique. Although this reduces the communication cost between the prover and the verifier, these shorter shares of length sm do not allow a reconstruction of the secret, as they do not match the original length m and thus do not contain the necessary information for a successful reconstruction. This problem can be solved by introducing a hash function $\mathcal{SH} : \mathbb{Z}^{sm} \rightarrow \{0,1\}^m$, which maps each share to a binary vector of length m. Like the secret $\mathbf{x}$, these shares are too long to be efficiently obtained through a brute force attack and thus do not reduce the protocol's security. In addition, we need to calculate the auxiliary vector $\mathbf{\Delta x}$ from these shares to obtain a valid secret sharing. After that, we can follow the original protocol to implement the Small Integer Sharing.

In contrast to the original protocol, which needs a binarity proof through masking and cut-and-choose, the modified protocol does not. In the original Small Integer, paper [FMRV22], they need to prove that the secret $\mathbf{x}$ is a binary vector, which does not follow from the correctness of the equation $\sum_{j=1}^{n} \mathbf{x}_j \cdot \mathbf{g}_j = h \mod q$. This stems from the SIS mapping the shares into a different number space; thus, their sum does not need to be binary. Furthermore, they only have the sum of the equation to judge the secret from, which does not provide any information about the structure of the underlying vector $\mathbf{x}$. This $\mathbf{x}$ could contain a single number to achieve the needed sum h and fill the rest with zeros without being caught by the verifier. This is because all information is crushed into a single number. However, the problem of missing information about the secret is not contained in the syndrome decoding problem because its result is the vector y. This vector y allows the verifier to check whether the shared secret is within a specific number space by comparing each row to the maximum values one could get. In addition, the secret is bound by its weight constraint. Finally, by utilizing the batch product verification to prove that $wt(\mathbf{x}) \leq w$, we further bind $\mathbf{x}$ to be within the provided number space $\mathbb{F}_{SD}^{m}$.

We now formally define the SIS for the syndrome decoding problem.

SIS for SD

Let (H, y) be an instance of the syndrome decoding problem with $\mathsf{H} \in \mathbb{F}_q^{(m-k)\times m}$ and $\mathsf{y} \in \mathbb{F}_q^{m-k}$ and the secret $\mathbf{x} \in \{0,1\}^m$ such that

$$\mathsf{H} \cdot \mathbf{x} = \mathsf{y} \text{ and } wt(\mathbf{x}) \leq w$$

where $m, k, q \in \mathbb{Z}$ are security parameters. Then $k < m$ and $wt(\cdot)$ is the hamming weight of a given vector. Further, let $sm \in \mathbb{Z}$ be the length of the shorter shares, $A \in \mathbb{Z}$ the security parameter, similar to the one in the original SIS, and N the number of shares. $\mathcal{SH} : \mathbb{F_A}^{sm} \rightarrow \{0,1\}^m$ is a deterministic hash function that maps an input vector of length sm to a binary vector of length m. Thus, we utilize the Small Integer Sharing

SIS for SD

as follows:

$$\begin{cases} [\![\mathbf{x}]\!]_i & \xleftarrow{\$} \{0,\ldots,A-1\}^{sm} \text{ for all } i \in [N] \\ \Delta\mathbf{x} & \leftarrow \mathbf{x} - \mathcal{SH}\left(\sum_{i=1}^{N} [\![\mathbf{x}]\!]_i\right) \end{cases} \tag{7.1}$$

Here, we map the shorter vector of values from $\mathbb{Z}$ to a binary vector to further reduce the size of the resulting vector while not affecting the protocol's security. Thus, the verifier can obtain the secret $\mathbf{x}$ from this sharing precisely as in the original Small Integer Sharing by calculating the sum of the shares, applying the hash function to this sum, and adding the auxiliary vector to the hashed sum:

$$\mathbf{x} = \mathcal{SH}\left(\sum_{i=1}^{N} [\![\mathbf{x}]\!]_i\right) + \Delta\mathbf{x} \tag{7.2}$$

This optimization reduces the communication cost of the original SDitH protocol while slightly increasing the computational cost depending on the efficiency of the hash function. To counter this, we incorporate the hypercube optimization from Section 5.

7.1.2 SIS Hypercube

The hypercube optimization introduced by Aguilar-Melchor et al. [AGH+22] and described in Section 5 focuses on reducing the computational cost of the offline phase in an MPCitH protocol. For this, they introduce a new geometry they call hypercube, which is used to rearrange the original shares to reduce the number of MPC protocol executions needed for the validation check made by the verifier.

Let us recall the hypercube geometry. Given the original shares generated through additive secret sharing, called leaf-shares, they create a hypercube with length N_H in each of the D dimensions, allowing us to contain $N_H^D = N$ many leaf-party shares. After that, we create a new share for each hypercube position, where the shares of one dimension add up to the secret shared through the leaf-party shares. These new shares are called main-party shares and are calculated by adding up the corresponding leaf-party shares to obtain a new share. This works because the additive secret sharing used in the MPC protocol operates independently of the sampling method of the individual shares as long as they add up to the shared secret. These main-party shares are indexed through (k, j), where $k \in [D]$ determines the dimension of the hypercube and $j \in [N_H]$ specifies the main-party share in the selected dimension. In addition, each leaf-party share is given a list of indices $(i_1, \ldots, i_D) \in [N_H]^D$, where the position k in that list denotes the dimension in the hypercube and the value of i_k the main-party share in the dimension. Thus, the hypercube contains N_H^D many leaf-party shares through $N_H \cdot D$ many main-party shares. These main-party shares are each calculated over N_H^{D-1} leaf-party shares. Each share is used for one MPC protocol execution in the verification step, reducing the number of MPC executions by a factor greater than 10.

To utilize the hypercube geometry in connection with the Small Integer Sharing, we need to look at the underlying additive secret sharing used in the SIS scheme. The sharing differs slightly from the commonly used additive secret sharing described in Section 3.6 by uniformly random sampling N instead of $N - 1$ shares. However, this does not pose a problem in the implementation, as we can utilize the hypercube geometry and send the auxiliary value $\Delta\mathbf{x}$ separately. Furthermore, we do not need to modify the calculation of the auxiliary value because the sum of the main-party shares per dimension is equivalent to the sum of all leaf-party shares. Thus, if we calculate the sum of the main-party shares and add the auxiliary value calculated over the leaf-party shares, we get the secret $\mathbf{x}$. This means we create the hypercube using the small integer shares $[\![\mathbf{x}]\!]_{(i_1,\ldots,i_D)} \in \{0, \ldots, A-1\}^{sm}$ to calculate the main-party shares.

Similarly to the original SIS implementation, we can obtain the secret $\mathbf{x}$ by summing up the main-party shares of the corresponding dimension, extending them using the hash function, and adding the auxiliary value.

With this in mind, we can give a formal definition of the Small Integer Sharing with the hypercube geometry:

SIS Hypercube

Given an instance of the syndrome decoding problem (H, y) with $\mathsf{H} \in \mathbb{F}_{SD}^{(m-k)\times m}$ and $\mathsf{y} \in \mathbb{F}_{SD}^{m-k}$. Further, let there be a secret solution $\mathbf{x} \in \{0,1\}^m$, such that $\mathsf{H} \cdot \mathbf{x} = \mathsf{y}$ and $wt(\mathbf{x}) \leq w$. Here $m, k \in F_{SD}$ and $m > k$, we consider n many parties with N many shares. Then there N many leaf-party shares uniformly sampled as described in Equation 7.1. Each share is given a list of indexes $(i_1, \ldots, i_D) \in [N_H]^D$ as described in Section 5

SIS Hypercube

and an index of the main-party shares through (k, j) with $k \in [D]$ and $j \in [N_H]$. From these, the main party shares $[\![ms]\!]_{(k,j)}$ are calculated by selecting every leaf party share that has at position k in its index list $(i_1, \ldots, i_D)$ the value j. Thus, the prover P obtains $N \cdot D$ many main-party shares positioned by (k, j) in the hypercube structure. Given these main-party shares, P calculates the auxiliary value for the Small Integer Sharing by selecting a random dimension D' of the hypercube, determining their sum, and applying the deterministic hash-function $\mathcal{SH} : \{0, \ldots, A-1\}^{sm} \to \mathbb{F}_{SD}^m$ to the sum. Note that A is chosen as described in Section 4. This hashed sum is then subtracted from $\mathbf{x}$ to obtain the auxiliary value $\Delta\mathbf{x}$, as described in the following equation. Calculating the auxiliary value this way is the same as calculating it over the leaf-party shares because both add up to the same sum.

$$\Delta\mathbf{x} = \mathbf{x} - \mathcal{SH}(\sum_{j=1}^{N_H} [\![\mathbf{x}]\!]_j)$$

Now, the prover can share the leaf-party shares. To verify the obtained shares, the verifier can calculate the main-party shares from the leaf-party shares as described above and run the MPC protocol for each of the main-parties.

With these two optimizations, we continue to the final optimization. This follows the standard optimization of using TreePRGs from Section 6 but goes one step further and utilizes the OneTree technique from [BBM+24], which again reduces the communication cost of the entire protocol.

7.1.3 SIS Hypercube OneTree

The OneTree technique introduced by [BBM+24] and described in Section 6.1 can be applied to the Small Integer Sharing with the hypercube geometry because it only affects the generation of the random shares. Let us first recall the TreePRG before talking about the OneTree.

The TreePRG is a random number generator commonly based on the *Goldreich, Goldwasser, and Micali*(GGM) tree structure. This tree is initialized with a random length d, representing the tree's root. After that, one utilizes the

length-doubling method, which takes an input of length d and generates a random number of length $2 \cdot d$. After that, the new random number is split in half and assigned to either the current node's left or right child. This process is repeated until we obtain N leafs, which can be used as shares. This structure allows the prover to send only the tree nodes that are needed for the verifier to calculate all other shares. In the context of the MPC in the head protocol, the prover sends all nodes needed for the reconstruction without sending any node on the path to the hidden share. Thus, she needs to communicate $\log_2(N)$ many nodes, as shown in Figure 6.1.
The OneTree technique comes into play when we look at the τ execution of an MPCitH protocol in order to achieve a given soundness error ε. For this, we generate τ TreePRGs, one for each execution, and rearrange them such that the first τ leafs of the OneTree correspond to the first element of each of the τ commitments. This rearrangement can be seen by comparing the leaf order in Figure 6.2 and the leaf order in Figure 6.3. This rearrangement allows the prover to reduce further the number of sent nodes by a factor of 0.6.
In order for us to incorporate this structure, we change the leaf-party share generation to utilize a TreePRG for each of the τ executions and rearrange them accordingly to form the OneTree structure. After that, the protocol is executed without any modifications, using the new random shares as described in the previous section.
These optimizations form our approach to the MPCitH protocol based on the syndrome decoding problem. The entire protocol is described in the following Protocol 7.1.

SIS hypercube OneTree SDitH protocol part 1

Prover		**Verifier**
$\mathsf{H} = (\mathsf{H}' \vert \mathsf{I}_{m-k}) \in \mathbb{F}_{SD}^{(m-k)\times m}$ $\mathsf{y} \in \mathbb{F}_{SD}^{m-k}$ $\mathbf{x} = (\mathbf{x}_A, \mathbf{x}_B) \in \mathbb{F}_{SD}^{m}$ such that $\mathsf{y} = \mathsf{H} \cdot \mathbf{x}$ and $wt(x) \leq w$		$\mathsf{H} = (\mathsf{H}' \vert \mathsf{I}_{m-k}) \in \mathbb{F}_{SD}^{(m-k)\times m}$ $\mathsf{y} \in \mathbb{F}_{SD}^{m-k}$
# Construct the MPC inputs for each # of the τ executions $\mathbf{S}, \mathbf{Q}, \mathbf{P}, \mathbf{F}, tree =$ Proto. 7.2 # Compute the shares $(\llbracket \mathbf{x}_A \rrbracket, \llbracket \mathbf{Q} \rrbracket, \llbracket \mathbf{P} \rrbracket, \llbracket \mathbf{a} \rrbracket, \llbracket \mathbf{b} \rrbracket, \llbracket \mathbf{c} \rrbracket)$, $(\Delta \mathbf{x}_A, \Delta \mathbf{Q}, \Delta \mathbf{P}, \Delta \mathbf{a}, \Delta \mathbf{b}, \Delta \mathbf{c}) =$ Proto. 7.3 # Compute the commitment hash For each execution $e \in [\tau]$: $\quad \mathbf{com}_{aux} = Com(\Delta \mathbf{x}_A \vert\vert \Delta \mathbf{Q} \vert\vert \Delta \mathbf{P} \vert\vert \Delta \mathbf{a} \vert\vert \Delta \mathbf{b} \vert\vert \Delta \mathbf{c})$ $\quad h^{[e]} = Hash_1(\mathbf{com}_1^{[e]}, \ldots, \mathbf{com}_{N_H^D}^{[e]}, \mathbf{com}_{aux}^{[e]})$ Rearrange the trees of the τ treePRGs to form a OneTree		
	$\xrightarrow{h^{[e]} \forall e \in [\tau]}$	
		# Sample t challenge points for # polynomial verification # for all τ executions For each execution $e \in [\tau]$: $\quad (\mathsf{z}, \epsilon)^{[e]} \in \mathbb{F}_{points}^{t} \times \mathbb{F}_{points}^{t}$
	$\xleftarrow{(\mathsf{z}, \epsilon)}$	
# Execute MPC protocol on (z, ϵ) Build hash h'_k by running Algo. 7.4. $h' = Hash_4(h'_1, \ldots, h'_D)$		
	$\xrightarrow{h'}$	
		Pick uniformly for each $e \in [\tau]$: $\quad (i_1^*, \ldots, i_D^*)^{[e]} \to \{1, \ldots, N_H\}^D$
	$\xleftarrow{((i_1^*, \ldots, i_D^*)^{[e]})_{\forall e \in [\tau]}}$	

SIS hypercube OneTree SDitH protocol part 1

Prover		**Verifier**
# Open the corresponding elements If there exists a $j \in [N_H^D]$ such that: · either $[\![\mathbf{x}_j]\!]_{(i_1^*,\dots,i_D^*)} = 0$ with $\mathbf{x}_j = 1$ · or $[\![\mathbf{x}_j]\!]_{(i_1^*,\dots,i_D^*)} = A-1$ with $\mathbf{x}_j = 0$ then abort else: Send answer using OneTree sibling path and calculate $[\![\alpha]\!]_{(i_1^*,\dots,i_D^*)}, [\![\beta]\!]_{(i_1^*,\dots,i_D^*)}$ using Proto. 7.4 on related leaf-parties		
	$((seed_{(i_1,\dots,i_D)}, p_{(i_1,\dots,i_D)})$ $\forall (i_1,\dots,i_D) \neq (i_1^*,\dots,i_D^*)$ $\textbf{com}_{(i_1^*,\dots,i_D^*)},$ $[\![\alpha]\!]_{(i_1^*,\dots,i_D^*)},$ $[\![\beta]\!]_{(i_1^*,\dots,i_D^*)})^{\forall e \in [\tau]}$ $\longrightarrow$	
		The verifier accepts if and only if: **1.** For each $i' \neq i^*$: · Expand seeds to get leaf-party shares (they have $0.6 \cdot (D \cdot \log(N_H))^\tau$ seeds in the sibling paths), each is expanded to the leaf-party level giving $(N_H^D - 1)^\tau$ leaves · Calculate h' using the computed seeds, auxiliary values and $\textbf{com}_{i^*}$ · Compare h' and h, where $h = Hash_2(\textbf{com}_1, \dots, \textbf{com}_{N_H^D}, \textbf{com}_{aux})$ **2.** Compute the hash h'_v and $\alpha, \beta, \mathsf{v}$ using Proto. 7.5 and check: · $\alpha, \beta, \mathsf{v}$ are the same in each of the t executions · $h'_v = Hash_4(h_1^{v'}, \dots, h_D^{v'})$ is the same as h' from the prover

Protocol 7.1: This protocol describes our approach to optimize the MPCitH protocol by utilizing the Small Integer Sharing from [FMRV22], the hypercube geometry from [AGH+22] and the OneTree pseudorandom generator structure introduced in [BBM+24].

Construction of the polynomials and tree expansion

Input: The secret $\mathbf{x}$, its length m, the security parameter λ and τ
Output: $\mathbf{S}, \mathbf{Q}, \mathbf{P}, \mathbf{F}, tree$

```
# the $\tau$ executions
For each e $\in [\tau]$ :
  # Generate root seed
  $mseed^{[e]} \xleftarrow{\$} \{0,1\}^\lambda$
  # Compute $N_H^D$ leaf seeds through
  # TreePRG (expand root into $N_H^D$ leafs)
  $(seed_i^{[e]}, p_i^{[e]})_{i \in N_H^D} \leftarrow TreePRG(mseed^{[e]})$        $seed_i^{[e]} \in \{0,1\}^\lambda, p_i^{[e]} \in \mathbb{Z}$

# Create polynomials for batch product
# verification ($wt(\mathbf{x}) \leq w$)
Choose v $\subset [m]$ such that $|v| = m$ and $\{i \in [m] : \mathbf{x}_i \neq 0\} \subset$ v)
$\mathbf{S} \in \mathbb{F}_{poly}[X]$, such that $\forall i \in [m]\ \mathbf{S}(\gamma_i) = \phi(\mathbf{x}_i)$ and $deg(\mathbf{S}) \leq m-1$        $\mathbf{S} \in \mathbb{F}_{poly}^m$
$\mathbf{Q} \in \mathbb{F}_{poly}[X]$, such that $\mathbf{Q}(X) = \prod_{i \in v}(X - \gamma_i)$        $\mathbf{Q} \in \mathbb{F}_{poly}^q$ with $q = deq(\mathbf{Q}) \leq w-1$
$\mathbf{F} \in \mathbb{F}_{poly}[X]$, such that $\mathbf{F}(X) = \prod_{i \in [m]}(X - \gamma_i)$        $\mathbf{F} \in \mathbb{F}_{poly}^m$
$\mathbf{P} \in \mathbb{F}_{poly}[X]$, such that $\mathbf{P} = (\mathbf{Q} \cdot \mathbf{S}) \cdot \mathbf{F}$        $\mathbf{P} \in \mathbb{F}_{poly}^{\leq m}$
```

Protocol 7.2: In this protocol, we construct the treePRG and the polynomials for the SDitH protocol. Note that $\mathbf{S}$, in contrast to the other polynomials, is obtained by interpolation over the coordinates of $\mathbf{x}$. Furthermore, γ is the bijective mapping between $\{1, \ldots, \mathbb{F}_{poly}\}$ and $\mathbb{F}_{poly}$ and ϕ is the canonical inclusion of $\mathbb{F}_{SD}$ in $\mathbb{F}_{poly}$.

Construction of the secret shares

```
Input: The polynomials: $\mathbf{S}, \mathbf{Q}, \mathbf{P}, \mathbf{F}, tree$
Output: $([\![\mathbf{x}]\!], [\![\mathbf{Q}]\!], [\![\mathbf{P}]\!], [\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\![\mathbf{c}]\!])$,
$(\Delta\mathbf{x}, \Delta\mathbf{Q}, \Delta\mathbf{P}, \Delta\mathbf{a}, \Delta\mathbf{b}, \Delta\mathbf{c})$, com
# Initialize main party shares as zero
For each share $(k, j) \in [D] \times [N_H]$ :
    # Secret share the information containing $\mathbf{x}$
    $[\![x]\!]_{(k,j)} = 0$
    # Polynomial shares
    $[\![\mathbf{Q}]\!]_{(k,j)} = 0, [\![\mathbf{P}]\!]_{(k,j)} = 0$
    # Beaver triple shares
    $[\![\mathbf{a}]\!]_{(k,j)} = 0, [\![\mathbf{b}]\!]_{(k,j)} = 0, [\![\mathbf{c}]\!]_{(k,j)} = 0$

# Generate polynomial share at leaf level
For each leaf $i' \in [N_H^D]$ :
    $\{[\![\mathbf{a}]\!]_{i'}, [\![\mathbf{b}]\!]_{i'}, [\![\mathbf{c}]\!]_{i'}\} \leftarrow PRG(seed_{i'})$        $[\![\mathbf{a}]\!]_{i'}, [\![\mathbf{b}]\!]_{i'}, [\![\mathbf{c}]\!]_{i'} \in \mathbb{F}_{points} \times \mathbb{F}_{points}, \times\mathbb{F}_{points}$
    $[\![\mathbf{x}]\!]_{i'} \leftarrow PRG(seed_{i'})$        $[\![\mathbf{x}]\!]_{i'} \in \mathbb{F}_A^{sm}$
    $([\![\mathbf{Q}]\!]_{i'}, [\![\mathbf{P}]\!]_{i'}) \leftarrow PRG(seed_{i'})$        $([\![\mathbf{Q}]\!]_{i'}, [\![\mathbf{P}]\!]_{i'}) \in \mathbb{F}_A^{|\mathbf{Q}|} \times \mathbb{F}_A^{|\mathbf{P}|}$
    com$_{i'} = Com(seed_{i'}, p_{i'})$

Index leaf party shares $[\![\mathbf{x}]\!]_i$ on hypercube $(i'_1, \dots, i'_D)$,
where $i'_k \in \{0, \dots, N_H - 1\}$
# Compute main party shares
For each main party share index
$p \in \{(1, i'_1), (2, i'_2), \dots, (D, i'_D)\}$ :
    $[\![\mathbf{x}]\!]_p += [\![\mathbf{x}]\!]_{i'}$        $[\![\mathbf{x}]\!]_p \in \mathbb{F}_A^{sm}$
    $[\![\mathbf{Q}]\!]_p += [\![\mathbf{Q}]\!]_{i'}, [\![\mathbf{P}]\!]_p += [\![\mathbf{P}]\!]_{i'}$        $[\![\mathbf{Q}]\!]_p, [\![\mathbf{P}]\!]_p \in \mathbb{F}_A^{|\mathbf{Q}|} \times \mathbb{F}_A^{|\mathbf{P}|}$
    $[\![\mathbf{a}]\!]_p += [\![\mathbf{a}]\!]_{i'}, [\![\mathbf{b}]\!]_p += [\![\mathbf{b}]\!]_{i'}, [\![\mathbf{c}]\!]_p += [\![\mathbf{c}]\!]_{i'}$        $[\![\mathbf{a}]\!]_{i'}, [\![\mathbf{b}]\!]_{i'}, [\![\mathbf{c}]\!]_{i'} \in \mathbb{F}_{points} \times \mathbb{F}_{points}, \times\mathbb{F}_{points}$

# Calculate the auxiliary values
Choose $d \in [D]$
$\Delta\mathbf{x} = \mathbf{x} - \mathcal{SH}\left(\sum_{i\in\{0,\dots,N_H-1\}} [\![\mathbf{x}_A]\!]_i^{[d]}\right)$        $\Delta\mathbf{x} \in \mathbb{F}_{SD}^m$
$\Delta\mathbf{Q} = \mathbf{Q} - \mathcal{SH}\left(\sum_{i\in\{0,\dots,N_H-1\}} [\![\mathbf{Q}]\!]_i^{[d]}\right), \Delta\mathbf{P} = \mathbf{P} - \mathcal{SH}\left(\sum_{i\in\{0,\dots,N_H-1\}} [\![\mathbf{P}]\!]_i^{[d]}\right)$        $\Delta\mathbf{Q}, \Delta\mathbf{P} \in \mathbb{F}_{poly}^{|\mathbf{Q}|} \times \mathbb{F}_{poly}^{|\mathbf{P}|}$
$\Delta\mathbf{a} = \mathsf{a} - \mathcal{SH}\left(\sum_{i\in\{0,\dots,N_H-1\}} [\![\mathbf{a}]\!]_i^{[d]}\right), \Delta\mathbf{b} = \mathsf{b} - \mathcal{SH}\left(\sum_{i\in\{0,\dots,N_H-1\}} [\![\mathbf{b}]\!]_i^{[d]}\right)$        $\Delta\mathbf{a}, \Delta\mathbf{b} \in \mathbb{F}_{points} \times \mathbb{F}_{points}$
$\Delta\mathbf{c} = \langle \mathsf{a}, \mathsf{b} \rangle - \mathcal{SH}\left(\sum_{i\in\{0,\dots,N_H-1\}} [\![\mathbf{c}]\!]_i^{[d]}\right)$        $\Delta\mathbf{c} \in \mathbb{F}_{points}$
```

Protocol 7.3: This protocol generates the secret shares, as well as the polynomial shares with their beaver triple, needed for the verification process at the end of the main Protocol 7.1. Note that each share of a, b, c is a single polynomial. In addition, the indices on the hypercube for each leaf-party share are a list or tuple, where the position in the list represents the dimension of the hypercube and the value at that position the main-party share it belongs to. This means that each main-party share is calculated using N_H^{D-1} many leaf-party shares.

Execution of $\prod$ on a given set of parties

Input: The secret shares: $(\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{Q} \rrbracket, \llbracket \mathbf{P} \rrbracket, \llbracket \mathbf{a} \rrbracket, \llbracket \mathbf{b} \rrbracket, \llbracket \mathbf{c} \rrbracket)$,
$(\Delta \mathbf{x}_A, \Delta \mathbf{Q}, \Delta \mathbf{P}, \Delta \mathbf{a}, \Delta \mathbf{b}, \Delta \mathbf{c})$
and the t evaluations points (z, ϵ)

Output: $\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket \mathbf{v} \rrbracket, h'_k$

```
For each axis $k \in [D]$ between main parties $(k,1)\dots(k,N_H)$ :
  Main-Parties locally compute $\llbracket \mathbf{S} \rrbracket$                                   $\llbracket \mathbf{S} \rrbracket \in \mathbb{F}_A^{sm}$
    by interpolating $\llbracket \mathbf{x} \rrbracket$
  Main-parties interpolate $\Delta \mathbf{S}$ over $\Delta \mathbf{x}$                            $\Delta \mathbf{S} \in \mathbb{F}_{SD}^{m}$
  # Compute Beaver triple elements
  For each $l \in [t]$ :
    Main-Parties locally evaluate $\llbracket \mathbf{S} \rrbracket(\mathsf{z}_l), \llbracket \mathbf{Q} \rrbracket(\mathsf{z}_l), \llbracket \mathbf{P} \rrbracket(\mathsf{z}_l)$      $\llbracket \mathbf{S} \rrbracket(\mathsf{z}_l), \llbracket \mathbf{Q} \rrbracket(\mathsf{z}_l), \llbracket \mathbf{P} \rrbracket(\mathsf{z}_l) \in \mathbb{F}_A \times \mathbb{F}_A \times \mathbb{F}_A$
    Main-Parties $\llbracket \alpha_l \rrbracket = \epsilon_l \cdot \llbracket \mathbf{Q} \rrbracket(\mathsf{z}_l) + \llbracket \mathbf{a}_l \rrbracket$      $\llbracket \alpha_l \rrbracket, \epsilon_l, \llbracket \mathbf{a}_l \rrbracket \in \mathbb{F}_A \times \mathbb{F}_A \times \mathbb{F}_A$
    Main-Parties $\llbracket \beta_l \rrbracket = \llbracket \mathbf{S} \rrbracket(\mathsf{z}_l) + \llbracket \mathbf{b}_l \rrbracket$      $\llbracket \beta_l \rrbracket, \llbracket \mathbf{b}_l \rrbracket \in \mathbb{F}_A \times \mathbb{F}_A$
    Main-Parties $\Delta \alpha_l = \epsilon_l \cdot \Delta \mathbf{Q}(\mathsf{z}_l) + \Delta \mathbf{a}_l$      $\Delta \alpha_l, \Delta \mathbf{Q}, \Delta \mathbf{a}_l \in \mathbb{F}_{points} \times \mathbb{F}_{points}^{|\mathbf{Q}|} \times \mathbb{F}_{points}$
    Main-Parties $\Delta \beta_l = \Delta \mathbf{S}(\mathsf{z}_l) + \Delta \mathbf{b}_l$      $\Delta \beta_l, \Delta \mathbf{b}_l \in F_{points} \times \mathbb{F}_{points}$
    open $\llbracket \alpha_l \rrbracket$ and $\llbracket \beta_l \rrbracket$ to get $\alpha_l, \beta_l$      $\alpha_l, \beta_l \in \mathbb{F}_{points} \times \mathbb{F}_{points}$
    Main-Parties locally:
      $\llbracket v_l \rrbracket = -\llbracket \mathbf{c}_l \rrbracket + \langle \epsilon, \mathbf{F}(\mathsf{z}_l) \cdot \llbracket \mathbf{P} \rrbracket(\mathsf{z}_l) \rangle + \langle \alpha_l, \llbracket \mathbf{b}_l \rrbracket \rangle + \langle \beta_l, a_l \rangle - \langle \alpha_l, \beta_l \rangle$      $\llbracket v_l \rrbracket, \mathbf{F}(\mathsf{z}_l) \in \mathbb{F}_A \times \mathbb{F}_{poly}^{m}$
      $\Delta \mathbf{v}_l = -\Delta \mathbf{c}_l + \langle \epsilon, \mathbf{F}(\mathsf{z}_l) \cdot \Delta \mathbf{P}(\mathsf{z}_l)) \rangle + \langle \alpha_l, \Delta \mathbf{b}_l) \rangle + \langle \beta_l, \Delta \mathbf{a}_l) \rangle - \langle \alpha_l, \beta_l \rangle$      $\Delta \mathbf{v}_e, \Delta \mathbf{c}_l, \Delta \mathbf{P}(\mathsf{z}_l) \in \mathbb{F}_{points} \times \mathbb{F}_{points} \times \mathbb{F}_{points}$

  $h'_k = Hash_3(\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket \mathbf{v} \rrbracket, \Delta \mathbf{v})$
```

Protocol 7.4: This protocol describes the execution of the MPC protocol for given z, ϵ points. Here, the polynomials are evaluated on the given points to calculate the beaver triple and verify the weight constraint of the secret without revealing any information about it. Recall that opening the shares $\llbracket \alpha_l \rrbracket, \llbracket \beta_l \rrbracket$ means we calculate the sum of their shares and add the auxiliary value to obtain the secret shares α and β. The auxiliary values must be calculated separately because of the Small Integer Sharing.

Verify a partition of parties

```
Input: The secret shares: $([\![\mathbf{x}]\!], [\![\mathbf{Q}]\!], [\![\mathbf{P}]\!], [\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\![\mathbf{c}]\!])$,
  $(\Delta\mathbf{x}, \Delta\mathbf{Q}, \Delta\mathbf{P}, \Delta\mathbf{a}, \Delta\mathbf{b}, \Delta\mathbf{c})$
  and the $t$ evaluations points $(\mathsf{z}, \epsilon)$. For the leaf-party with hidden
  share $i^*$ we use partial aggregations from its disclosed leaf-parties.
  And get the index $i^*$ with the communications $[\![\alpha]\!]_{i^*}, [\![\beta]\!]_{i^*}, [\![\mathbf{v}]\!]_{i^*}$
Output: $[\![\alpha]\!], [\![\beta]\!], [\![\mathbf{v}]\!], h_k^{v'}$

For each $h \in [D]$ :
  Main-Parties locally compute $[\![\mathbf{S}]\!]$ by interpolating $[\![\mathbf{x}]\!]$      $[\![\mathbf{S}]\!] \in \mathbb{F}_A^{sm}$
  Main-parties interpolate $\Delta\mathbf{S}$ over $\Delta\mathbf{x}$      $\Delta\mathbf{S} \in \mathbb{F}_{SD}^{m}$
  # Compute Beaver triple elements
  For each $l \in [t]$ :
    Main-Parties locally evaluate $[\![\mathbf{S}]\!](\mathsf{z}_l), [\![\mathbf{Q}]\!](\mathsf{z}_l), [\![\mathbf{P}]\!](\mathsf{z}_l)$      $[\![\mathbf{S}]\!](\mathsf{z}_l), [\![\mathbf{Q}]\!](\mathsf{z}_l), [\![\mathbf{P}]\!](\mathsf{z}_l) \in \mathbb{F}_A \times \mathbb{F}_A \times \mathbb{F}_A$
    Main-Parties $[\![\alpha_l]\!] = \epsilon_l \cdot ([\![\mathbf{Q}]\!](\mathsf{z}_l) + [\![\mathbf{a}_l]\!])$      $[\![\alpha_l]\!], \epsilon_l, [\![\mathbf{a}_l]\!] \in \mathbb{F}_A \times \mathbb{F}_A \times \mathbb{F}_A$
    Main-Parties $[\![\beta_l]\!] = [\![\mathbf{S}]\!](\mathsf{z}_l) + [\![\mathbf{b}_l]\!]$      $[\![\beta_l]\!], [\![\mathbf{b}_l]\!] \in \mathbb{F}_A \times \mathbb{F}_A$
    The partially disclosed parties calculate their shares $[\![\alpha_l]\!], [\![\beta_l]\!]$
    as above and add the communicated $[\![\alpha_l]\!]_{i^*}, [\![\beta_l]\!]_{i^*}$
    to $[\![\alpha_l]\!]$ and $[\![\beta_l]\!]$
    Main-Parties $\Delta\alpha_l = \epsilon_l \cdot \Delta\mathbf{Q}(\mathsf{z}_l) + \Delta\mathbf{a}_l$      $\Delta\alpha_l, \Delta\mathbf{Q}, \Delta\mathbf{a}_l \in \mathbb{F}_{points} \times \mathbb{F}_{points}^{|\mathbf{Q}|} \times \mathbb{F}_{points}$
    Main-Parties $\Delta\beta_l = \Delta\mathbf{S}(\mathsf{z}_l) + \Delta\mathbf{b}_l$      $\Delta\beta_l, \Delta\mathbf{b}_l \in F_{points} \times \mathbb{F}_{points}$
  Open $[\![\alpha_l]\!]$ and $[\![\beta_l]\!]$ to get $\alpha_l, \beta_l$      $\alpha_l, \beta_l \in \mathbb{F}_{points} \times \mathbb{F}_{points}$
  Main-Parties locally:
      $[\![v_l]\!] = -[\![\mathbf{c}_l]\!] + \langle \epsilon, \mathbf{F}(\mathsf{z}_l) \cdot [\![\mathbf{P}]\!](\mathsf{z}_l)\rangle + \langle \alpha_l, [\![\mathbf{b}_l]\!]\rangle + \langle \beta_l, a_l\rangle - \langle \alpha_l, \beta_l\rangle$      $[\![v_l]\!], \mathbf{F}(\mathsf{z}_l) \in \mathbb{F}_A \times \mathbb{F}_{poly}^{m}$
      $\Delta\mathbf{v}_l = -\Delta\mathbf{c}_l + \langle \epsilon, \mathbf{F}(\mathsf{z}_l) \cdot \Delta\mathbf{P}(\mathsf{z}_l))\rangle + \langle \alpha_l, \Delta\mathbf{b}_l)\rangle + \langle \beta_l, \Delta\mathbf{a}_l)\rangle - \langle \alpha_l, \beta_l\rangle$      $\Delta\mathbf{v}_e, \Delta\mathbf{c}_l, \Delta\mathbf{P}(\mathsf{z}_l) \in \mathbb{F}_{points} \times \mathbb{F}_{points} \times \mathbb{F}_{points}$
  $[\![\mathbf{v}_l]\!]_{i^*}$ is set such that $\mathsf{v}_l = 0 \rightarrow [\![\mathbf{v}_l]\!]_{i^*} = -(\Delta\mathbf{v}_l + \sum_{i \neq i^*} [\![\mathbf{v}_l]\!]_i)$      $[\![\mathbf{v}_l]\!]_{i^*} \in \mathbb{F}_A$
  $h_k^{v'} = Hash_3([\![\alpha]\!], [\![\beta]\!], [\![\mathbf{v}]\!])$
```

Protocol 7.5: In this protocol, the verifier executes the MPC protocol to verify a partition of parties. The partially disclosed main-party shares are calculated similarly to the non-disclosed main-party shares, where a zero vector replaces the hidden leaf-party share. This allows us to calculate each α share independent of the hidden index and add the corresponding communicated $[\![\alpha]\!]_{i^*}$ if needed to obtain the entire share of α. Calculating the β and v share is done equivalently. This follows the hypercube structure of [AGH+22], where this procedure is not explicitly mentioned. Furthermore, note that this process needs to be done for $\frac{D}{2}$ many main-party shares, which is the number of partially disclosed main-parties.

Before we dive into the performance analysis, we will look at the non-interactive transformation of our SDitH protocol with the hypercube and OneTree structure to obtain a signature scheme. For this, we utilize the Fiat-Shamir transformation and follow the methodology described by [FJR22]. More precisely, the first and second challenges are generated non-interactively by hashing the transcript up to that point and expanding the hashes to form their corresponding challenge. For the first challenge, we generate the hash from the initial commitments across all τ repetitions to get:

$$h_2 = Hash_2(m, salt, \mathbf{com}^{[1]}, \ldots, \mathbf{com}^{[\tau]})$$

Then we expand the challenge through the pseudorandom generator $PRG(h_2)$ to obtain the challenge points $\{\mathsf{z}^{[e]}, \epsilon^{[e]}\}$. In order to acquire the second challenge, we again hash the transcript that is generated up to the point of the second challenge, which results in:

$$h_3 = Hash_4(m, salt, h_2, \{h_1'^{[e]}, \ldots, h_D'^{[e]}\}_{e\in[\tau]})$$

Similarly to the first challenge, this hash is expanded using the $PRG(h_3)$ to obtain the leaf party shares for the second challenge, namely $\{i^{*[e]}\}_{e\in[\tau]}$.

Furthermore, note that because of this transformation, the security of the protocol can no longer be purely considered based on a brute force attack, as the attack on the Fiat-Shamir transformed scheme gives the attacker an advantage, which was described by [AABN02] and [FS06]. In essence, this tells us that for protocols with more than one challenge from a verifier, the prover often needs to correctly guess only one of the challenges for her to cheat successfully. Thus, the challenges of protocols with five or more passes can no longer be considered independently.

With this in mind, we can derive the forgery attack cost. Following the attack presented by [KZ20], which generates an additive attack cost instead of a multiplicative one by breaking the two rounds of the protocol separately. This cost relies on finding an optimal τ' that allows us to correctly guess the first challenge with the lowest possible cost. This cost can be calculated via:

$$\begin{aligned} cost_{forge} &:= \min_{0\leq\tau'\leq\tau}\left\{\frac{1}{\sum_{i=\tau'}^{\tau}\binom{\tau}{i}\cdot p^i\cdot(1-p)^{\tau-i}}+(N_H^D)^{\tau-\tau'}\right\} \\ &\leq \min_{0\leq\tau'\leq\tau}\left\{\frac{1}{p}^{\tau'}+(N_H^D)^{\tau-\tau'}\right\} \end{aligned}$$

Here, p is the false positive probability, which is the same as in the syndrome decoding (Equation 3.10). This forgery cost is much lower than the brute-force cost of $\frac{1}{\epsilon^\tau}$, which in turn means that the security parameter needs to be modified accordingly. This strategy and forgery cost follows the one described in the paper by [AGH+22].

In this section, we detailed the non-interactive transformation of our SDitH protocol using the Fiat-Shamir methodology, specifying how challenges are generated through hashing and pseudorandom expansion. By applying this transformation, we achieve a signature scheme that balances efficiency with enhanced resistance to forgery, accounting for the adversarial advantage outlined by [AABN02] and [FS06]. With this structure in place, we can proceed to analyze the performance implications, including the effectiveness of SIS for syndrome decoding with the hypercube and the OneTree optimization.

7.2 Performance Analysis

in this section we analyze our protocol's performance (7.1) concerning the communication costs. We will provide the cost of the zero-knowledge protocol in order for us to compare it to other protocols based on the syndrome decoding problem. More precisely, we will compare our approach to the original SDitH protocol of [FJR22], as well as the version utilizing the hypercube geometry [AGH+22] and the protocol that introduced the OneTree pseudorandom number generator based on the tree structure [BB24]. Before we dive into the entire protocol, we will examine the individual improvements of the optimizations we utilized. Again, we start with the Small Integer Sharing, then include the hypercube geometry, and finally, add the OneTree structure. Note that we will examine these optimizations in isolation without any surrounding communication.

For the Small Integer Sharing, we need to sample N uniformly random vectors and calculate a corresponding auxiliary vector of length m. However, instead of sampling random vectors of length m, we reduce their length to be sm, where $sm << m$. Furthermore, we need to consider the new number space of A, from which the N uniformly random vectors are sampled, resulting in a sampling from $\{0, \ldots, A-1\}$. The auxiliary value is the difference between the hashed sum of the shares and the secret $\mathbf{x}$ in the corresponding number space $\mathbb{F}_{SD}$. Thus, the resulting communication costs are:

$$costs_{comm} = N \cdot \log_2(A-1) \cdot sm + m \cdot \mathbb{F}_{SD} \tag{7.3}$$

After applying the Small Integer Sharing, we add the hypercube geometry to the sharing and verification process. This geometry does not impact the communication cost but reduces the computational costs. This is important because replacing the additive secret sharing with the Small Integer Sharing adds computational cost in the form of the hash functions, which are needed to expand the short shares to the longer ones needed to obtain the secret. The hypercube geometry reduces the n evaluations (one for each party) to $1 + (N_H - 1) \cdot D$. On average, this comes down to about 10 times fewer calculations.

The final optimization is adding the One Tree to Rule them All (OneTree), which rearranges the shares of the τ protocol executions into a single tree in a way that, on average, only 60% of the sibling path nodes in the treePRG need to be send. For this, we will firstly modify the Equation 7.3 for the SIS sharing to incorporate τ executions, leading to:

$$costs_{comm} = \tau \cdot (N \cdot \log_2(A-1) \cdot sm + m \cdot \mathbb{F}_{SD}))$$

The OneTree now affects, as mentioned, the number of shares that need to be sent over the τ executions. Furthermore, utilizing the GGM pseudorandom number generator allows us to send the sibling path to the hidden share. This sibling path contains $\log_2(N)$ many nodes. It also means that we can send the seeds of the sibling path, which are bit strings of length λ. This reduces the communication costs significantly but also shows that the SIS optimization does not make any difference in the communication costs at this point. The resulting communication costs for the sharing are as follows:

$$costs_{comm} \approx 0.6 \cdot \tau(\cdot \log_2(N) \cdot \lambda + m \cdot \mathbb{F}_{SD}))$$

With these costs in mind, we can dive into the entire protocol. To simplify the performance analysis of the entire protocol, we will leave out some of the communication that can be seen in Protocol 7.1. More precisely, we ignore the communication cost for the challenges sent by the verifier because they add an arbitrarily small amount compared to the main communication cost. Thus, we need to consider the communication costs of the following aspects:

- **Com**: The hash h resulting from the N_H^D commitments
- **Res**$_1$: The hash h', which contains the D hashes from the MPC simulations
- **Res**$_2$: The final communication from the prover containing $(seed_{(i_1,\ldots,i_D)}, p_{(i_1,\ldots,i_D)}) \forall (i_1, \ldots, i_D) \neq (i_1^*, \ldots, i_D^*)$, $\mathbf{com}_{(i_1^*,\ldots,i_D^*)}, [\![\alpha]\!]_{(i_1^*,\ldots,i_D^*)}, [\![\beta]\!]_{(i_1^*,\ldots,i_D^*)})^{\forall e \in [\tau]}$

First we consider each of the leaf-party shares ($i' = (i_1, \ldots, i_D) \in \{1, \ldots, N_H^D\}$), which need to be shared via their treePRG seed of λ bits. In addition, we need to send the auxiliary value for the party shares ($\Delta\mathbf{x}$) and the corresponding auxiliary values for the batch product verification. This includes the polynomial auxiliary values $(\Delta\mathbf{Q}, \Delta\mathbf{P}) \in \mathbb{F}_{poly}$ and the values for the beaver triple ($[\![\alpha]\!], [\![\beta]\!], \Delta\mathbf{c} \in \mathbb{F}_{points}$). We also need to consider the impact of the hypercube geometry on the commitments and responses of the prover and verifier. This geometry allows us to formulate the costs of sending the leaf-party shares (tree seeds) as the sibling path to the hidden share of length D. This will cost us $D \cdot \lambda \cdot \log_2(N)$. Finally, we need to consider the costs of sending the commitments $\mathbf{com}_{(i_1^*,\ldots,i_D^*)}$, which cost $2 \cdot \lambda$ bits, as well as the two hashes h and h' each of length λ bits. The resulting communication costs for one execution are:

$$\begin{aligned}
costs_{comm} &= 4 \cdot \lambda && \textbf{Com} \text{ and } \textbf{Res}_1 \\
&+ D \cdot \lambda \cdot \log_2(N) && \text{PRG seeds} \\
&+ m \cdot \log_2(|\mathbb{F}_{SD}|) && \Delta\mathbf{x} \\
&+ (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) && \Delta\mathbf{Q}, \Delta\mathbf{P} \\
&+ 2 \cdot t \cdot \log_2(|\mathbb{F}_A|) && [\![\alpha]\!]_{(i_1^*,\ldots,i_D^*)}, [\![\beta]\!]_{(i_1^*,\ldots,i_D^*)} \\
&+ t \cdot \log_2(|F_points|) && \Delta\mathbf{c} \\
&+ 2 \cdot \lambda && \mathbf{com}_{i_1^*,\ldots,i_D^*}
\end{aligned}$$

Here, we can also see that the SIS optimization no longer affects the communication cost regarding the sharing because we do not send explicit shares but rather the seeds needed to reconstruct them. However, the α and β shares are in the smaller number space, which reduces the communication cost slightly. Similarly to the original SDitH, completing one execution of this process does not provide the target security level of $2^{-\lambda}$. We, therefore, need to perform the protocol τ times in parallel. However, we do not need to send every aspect τ times, as we can compute all commitments of the τ executions and the D hashes for each execution and produce a single hash. Thus, the final communication costs for our SDitH protocol with the hypercube and the OneTree structure are:

$$\begin{aligned} costs_{comm} = {} & 4 \cdot \lambda && \textbf{Com} \text{ and } \textbf{Res}_1 \\ & + \tau \cdot \big(D \cdot \lambda \cdot \log_2(N) && \text{PRG seeds} \\ & + m \cdot \log_2(|\mathbb{F}_{SD}|) && \Delta\mathbf{x} \\ & + (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) && \Delta\mathbf{Q}, \Delta\mathbf{P} \\ & + 2 \cdot t \cdot \log_2(|\mathbb{F}_A|) && [\![\alpha]\!]_{(i_1^*,\dots,i_D^*)}, [\![\beta]\!]_{(i_1^*,\dots,i_D^*)} \\ & + t \cdot \log_2(|F_{points}|) && \Delta\mathbf{c} \\ & + 2 \cdot \lambda\big) && \textbf{com}_{i_1^*,\dots,i_D^*} \end{aligned}$$

Finally, we analyze the SIS's computational impact on the protocol. Here, the most significant aspect is reducing the vector length and, thus, the elements of the MPC protocol execution. By reducing the vector length to sm, the prover and the verifier must perform the MPC protocol over smaller polynomials, significantly reducing the computation time. On the one hand, this is because the effort to interpolate the polynomial $\mathbf{S}$ depends on the number of interpolation points. On the other hand, calculations with polynomials are less optimized on modern hardware. Note that every operation of the SDitH protocol is done on polynomials as the number spaces are defined over prime fields (2^a, where a is some number in $\mathbb{Z}$), which means that each number must be represented by a polynomial in order to satisfy all needed characteristics, such as multiplicative inverse. Thus, we significantly reduce the computational costs of the SDitH protocol, which we will visualize in Section 8.2.

We will now dive into the security proof of our protocol in the following section.

7.3 Security Proof

The security proof of our SDitH protocol closely follows the proof from [AGH+22] due to the protocol's similarity and the underlying hardness assumptions. In addition, Protocol 7.1 follows a five-round structure, in which an honest prover P follows the protocol in the odd rounds $(1, 3, 5)$ and an honest verifier follows the even rounds $(2, 4)$. The rounds can be seen as the algorithms between sent messages.

For the security proof, we consider a general or malicious prover $\tilde{\mathsf{P}}$ to be someone who does not necessarily know the secret or follows the protocol correctly but produces messages of the same type as messages from an honest prover. The proof structure follows the security proofs of previous sections, where we show that a prover who knows a correct secret will always be accepted (Completeness) and provide an honest-verifier zero-knowledge proof. Lastly, we will show that a malicious prover who commits to a bad witness, which means that it does not encode a syndrome decoding solution, will only be accepted with a probability lower than or equal to $\varepsilon \approx \frac{1}{N_H^D}$ (soundness). However, before we start with the proof, we first need to establish the abort event probability, which will serve as a key parameter later in quantifying the likelihood that a malicious provers attempt results in protocol rejection, reinforcing the soundness guarantee. This proof follows closely the proof in [FMRV22].

Abort Events

To formally address the probability of protocol termination due to an abort event, we first define the relevant events associated with such occurrences. These events follow the conditions described in Section 4.1 to prevent information leakage of the shared secret. The abort events are defined as follows:

1. $A_j^0 := \{\mathbf{x}_j = 0, [\![\mathbf{x}_j]\!]_{i*} = A - 1\}$
2. $A_j^1 := \{\mathbf{x}_j = 1, [\![\mathbf{x}_j]\!]_{i^*} = 0\}$
3. $A_j = A_j^0 \cup A_j^1$

If we look at our SDitH protocol, we have, by construction, an abort prob-

Abort Events

ability of

$$Pr[abort] := Pr\left[\bigcup_{j=1}^{m} A_j\right] \tag{7.4}$$

If we now consider a random variable X, which is modeling the secret vector $\mathbf{x} \in \mathbb{F}_{SD}^m$, we have:

$$\begin{aligned}
Pr[abort|X = \mathbf{x}] &= Pr\left[\bigcup_{j=1}^{m} A_j|X = \mathbf{x}\right] \\
&= 1 - Pr\left[\bigcap_{j=1}^{m}(\neg A_j^0 \cap A_j^1)|X = \mathbf{x}\right] \\
&= 1 - Pr\left[\bigcap_{j=1}^{m}(\neg A_j^{\mathbf{x}_j}|X = \mathbf{x}\right] \\
&= 1 - Pr\left[\bigcap_{j=1}^{m}[\![\mathbf{x}_j]\!]_{i^*} \neq (1 - \mathbf{x}_j) \cdot (A - 1)\right] \\
&= 1 - \prod_{j=1}^{m} Pr\left[[\![\mathbf{x}_j]\!]_{i^*} \neq (1 - \mathbf{x}_j) \cdot (A - 1)\right] \\
&= 1 - \left(1 - \frac{1}{A}\right)^m
\end{aligned}$$

We can perform the second transformation because $\neg A_j^{1-\mathbf{x}_j}$ is true in the case of $X_j = \mathbf{x}_j$, which is assumed. Further, we can substitute the section operator inside the probability with the product of the probabilities because the coordinates of the secret share $[\![\mathbf{x}]\!]_{i^*}$ are independent of each other. Thus, we get the resulting probability of the abort event as:

$$Pr[abort] = 1 - \left(1 - \frac{1}{A}\right)^m \tag{7.5}$$

Furthermore, we know that it is independent of X.

With this in mind, we can go over the security proof of the Protocol 7.1, where we start with the completeness property.

(Perfect) Completeness

The protocol is perfectly complete, meaning that a prover P with the knowledge of a witness w, who performs her protocol rounds correctly and there is no abort event, will be accepted by an honest verifier V with probability 1. Thus, the completeness probability of the protocol is:

$$1 - Pr[abort]$$

Proof: For any choice of randomness for P, V, the computation of P passes all the verification checks performed by the verifier by construction. The completeness probability of $1 - Pr[abort]$ from Equation 7.5 implies the rest of the statement.

Second Aspect: Given a malicious prover $\tilde{\mathsf{P}}$ with a bad witness, meaning that $\mathbf{S} \cdot \mathbf{Q} \neq \mathbf{P} \cdot \mathbf{F}$ in the first round of the protocol, who is unable to find a hash or commitment collision has a probability of lower or equal to $\varepsilon = (p + \frac{(1-p)}{N_H^D})$ of cheating successfully. This means an honest verifier V falsely accepts her bad witness.

Proof: We consider the two scenarios in which V would accept, given a bad witness $\mathbf{S} \cdot \mathbf{Q} \neq \mathbf{P} \cdot \mathbf{F}$:

1. The random value that is encoded by $[\![\mathbf{v}]\!]$ is zero, which has a probability of p
2. $\tilde{\mathsf{P}}$ must cheat on the communications they send, corresponding to the MPCitH protocols on the main-parties, so the resulting vector v appears as the zero vector.

Because the first scenario is captured by the false positive probability p, we focus on the second scenario. After the first commitments the verifier sends the corresponding challenge points z, ϵ. Thus, looking at the first scenario, which occurs with a probability of p, the plain-text vector v, generated by the MPC protocol, is the zero vector. This is, for example, the case when on all t points $\delta = (\mathbf{S} \cdot \mathbf{Q} - \mathbf{P} \cdot \mathbf{F})(\mathsf{z}) = 0$ and/or the beaver triple of the first round, satisfies $\mathsf{c} - \langle \mathsf{a}, \mathsf{b} \rangle = \epsilon \cdot \delta$.

However, if at least one of the challenge points results in $\mathbf{S} \cdot \mathbf{Q}(\mathsf{z}_i) \neq \mathbf{P} \cdot \mathbf{F}(\mathsf{z}_i)$ than the malicious prover needs to cheat on the communication. This happens when at least one of the coordinates of the plain-text vector $\mathsf{v} = \mathsf{c} - \langle \mathsf{a}, \mathsf{b} \rangle - \epsilon \cdot \delta$ is non-zero, which has a probability of $(1 - p)$.

(Perfect) Completeness

The malicious prover needs to cheat on the communication because, in this case, the verifier would not accept the communication $[\![\alpha]\!], [\![\beta]\!], [\![\mathbf{v}]\!]$ resulting from an honest execution of the protocol.

The malicious prover commits to one independent SDitH run per hypercube dimension, resulting in D commitments in round 3. We can assume that $\tilde{\mathsf{P}}$ cheats on one of the SDitH runs' communications. Thus, without loss of generality, we can assume that she will cheat on the share of α. It is equally valid to assume that $\tilde{\mathsf{P}}$ cheats on either β or v.

Here, the share $[\![\alpha]\!]$ consists of N_H main-party shares $[\![\alpha]\!]_i$, meaning the malicious prover must pick one to cheat on. This has a success probability of $\frac{1}{N_H}$. However, each of the main-party shares consists of the sum of N_H^{D-1} leaf-party shares, from which all but one are opened. Thus, $\tilde{\mathsf{P}}$ must cheat on the share $[\![\alpha]\!]$ of a particular leaf-party lp by shifting its value by $\delta \neq 0$. Furthermore, because all but one leaf-party share are opened, the malicious prover cannot cheat on more than one of those shares, as one of the modified shares would be opened and immediately spotted. Cheating on none of the leaf-party shares would mean that v would not be the zero-vector, which the verifier would not accept.

Furthermore, the leaf-party ls is contained in one of the N_H main-party shares of each SDitH run. Thus, $\tilde{\mathsf{P}}$ must shift the value of $[\![\alpha]\!]$ by the same δ for each of the other main-party shares, which cannot be achieved by computing an offset using leaf-party shares. This follows the same reasoning of cheating on multiple leaf-party shares, which is always revealed by opening all but one leaf-party share. Any other cheating pattern is not possible by the same logic, which means that each main-party share containing ls must be cheated on by δ in their respective SDitH run.

This leads to only one way for $\tilde{\mathsf{P}}$ to avoid detection, where the uniformly random challenge i^* in round 4 reveals the exact coordinates of lp. Thus, the main-party containing lp is the hidden main-party share, which has a probability of $\frac{1}{N_H^D}$. This is equivalent to the challenge of hiding the exact leaf-party share ls out of the N_H^D leaf-party shares. Thus, in case of no abort event and a non-false positive scenario, the malicious prover has a chance of cheating of $\leq \frac{1}{N_H^D}$. Here, we can ignore the abort event

(Perfect) Completeness

probability because rejecting does not affect the verifier checks, shown in the abort event 7.3. Therefore, the resulting probability of a prover with a bad witness being accepted is:

$$\varepsilon = p + \frac{1-p}{N_H^D} \tag{7.6}$$

Now we dive into the *honest-verifier zero-knowledge* (HVZK) proof, where we construct a simulator replicating the protocol's transcript without knowledge of the secret. For this, we rely on the indistinguishability of the *pseudorandom number generator* (PRG) and commitment scheme from true randomness to ensure that the simulator's output is computationally indistinguishable from a real protocol execution. We will begin with a 'true transcript', representing an honest execution of the protocol, and then iteratively modify it to reach the simulator's transcript. By showing that each step maintains indistinguishability, we conclude that the protocol's distribution is unaffected by these transformations, which shows that our SDitH protocol is HVZK.

Honest-Verifier Zero-Knowledge

If the pseudorandom number generator of the protocol in combination with the commitment **Com** are indistinguishable from the uniform random distribution, then the protocol is honest-verifier zero knowledge.
Proof: Before we describe the simulator construction needed for this proof, it is important to note that the abort event is independent of the secret and, thus, does not leak any information about the secret. This was shown in Proof 7.3.
With this in mind, we need to construct a simulator $\mathcal{S}$ that creates a transcript of the Protocol 7.1, which is computationally indistinguishable from the real transcript. For this, we assume that the PRG of the protocol is (t, ϵ_{PRG})-secure, as well as the commitment **Com** is (t, ϵ_{com})-hiding. Furthermore, we will shorten the writing of the leaf-party indices $(i_{k_1}, \ldots, i_{k_D})$ to i' and the challenge indices $(i_1^*, \ldots, i_D^*)$ to i^*, for better readability.
After this, we can construct the simulator described in 7.6, which produces the transcript responses (**Com**, CH_1, RSP_1, CH_2, RSP_2). Next, we

Honest-Verifier Zero-Knowledge

need to show that the transcripts produced are computationally indistinguishable from a real transcript. We start by constructing a so-called true transcript, which is a transcript from the successfully executed protocol, and modify it step by step. After each modification, we need to argue why the indistinguishability still holds. We do this until the simulator $\mathcal{S}$ transcript is reached, which concludes the proof. Let us start with the true transcript.

TRUE TRANSCRIPT (V0): For this, we take as input a witness $\mathbf{x}$ and the honest verifier challenges $(\mathbf{z}, \epsilon, i^*)$, on which the Protocol 7.1 is executed correctly. Thus, we obtain a 'correct' output distribution.

SIMULATOR (V1): The first modification of the true transcript is replacing the randomness of the leaf-party i^* with true randomness. Thus, the auxiliary values $\Delta\mathbf{x}, \Delta\mathbf{Q}$, $\Delta\mathbf{P}, \Delta\mathbf{a}$ and $\Delta\mathbf{b}$ are calculated as stated in the protocol. So, the witness shares of all leaf-party shares still sum up to the corresponding sum and, in connection with the auxiliary values, give the input witness. By extension, this applies to every share in each of the MPCitH runs. Thus, the difficulty of distinguishing between the output of the simulator $\mathcal{S}_{v_1}$ and the real distribution is the same as distinguishing PRG from true randomness.

SIMULATOR (V2): For this modification, we alter the computation of the auxiliary values, which still depend on the input witness. For this we replace the auxiliary values $\Delta\mathbf{x}_A, \Delta\mathbf{Q}, \Delta\mathbf{P}, \Delta\mathbf{a}$ and $\Delta\mathbf{b}$ with true randomness. This does not change the distribution between the simulator $\mathcal{S}_{v_1}$ and $\mathcal{S}_{v_2}$ because they appear as true randomness through their original calculation as well. The only affected elements of the protocol are the auxiliary values $\Delta\alpha$ and $\Delta\beta$. However, the distribution of the simulator $\mathcal{S}_{v1}$ is not changed by this because they are calculated as described in Protocol 7.1, which is done on seemingly true randomness as stated before. Furthermore, we can reduce the input to the simulator to the challenges CH_1 and CH_2.

SIMULATOR (V3): Finally, we also draw $\Delta\alpha$ and $\Delta\beta$ via true randomness, which already appears to be uniformly distributed in the simulator $\mathcal{S}_{v_2}$ and thus do not change the distribution between $\mathcal{S}_{v_2}$ and $\mathcal{S}_{v_3}$. The outputs of this simulator (RSP_1, RSP_2) are indistinguishable from those of an honest execution of our Protocol 7.1. The resulting simulator, utilizes

Honest-Verifier Zero-Knowledge

the simulator described in Protocol 7.6 and applies the hiding property to $\mathbf{com}_{i^*}, \mathbf{com}_{aux}$. After that, it performs the following steps:

1. Generate the two challenges CH_1, CH_2
2. Run the simulator $\mathcal{S}_{v_3}$ in order to get RSP_1, RSP_2
3. Set the initial leaf-party commitments $i' \neq i^*$ as $\mathbf{com}'_i = \mathbf{Com}(seed_{i'}, p_{i'})$
4. Draw $\mathbf{com}_{i^*}$ at random for the hidden leaf-party i^*
5. Draw $\mathbf{com}_{aux}$ at random
6. Compute the initial commitments via
   $\mathbf{Com} = Hash(\mathbf{com}_1, \ldots, \mathbf{com}_{N_H^D}, \mathbf{com}_{aux})$

From this, we obtain the output of the global HVZK simulator, that is, $(t, \epsilon_{PRG} + \epsilon_{Com})$-indistinguishable from the real distribution.

HVZK Simulator

```
Sample seed $\overset{\$}{\leftarrow} \{0,1\}^\lambda$
Generate $(seed_{i'}, p_{i'})$ for all leaf-parties via $TreePRG(seed)$
**Step 1: Sample Challenges**
  $CH_1 = \{\mathsf{z}, \epsilon\} \leftarrow \mathbb{F}^t_{points} \times \mathbb{F}^t_{points}$
  $CH_2 = i^* \leftarrow [N_H^D]$

**Step 2: Generate $N_H^D$ leaf-party shares**
  Expand root $seed_{i'}$ via TreePRG to get $N_H^D$ leaf party seeds

**Step 3: Generate leaf-party commitments and witness shares**
  For each $i' \neq i^*$ :
    Compute **com**$_{i'} = Hash(seed_{i'}, p_{i'})$
    Expand leaf-party seeds to get secret shares
  Generate auxiliary values: $\Delta\mathbf{x}, \Delta\mathbf{Q}, \Delta\mathbf{P}, \Delta\mathbf{c}$
  Calcualte the auxiliary commitment **com**$_{aux} = Hash(\Delta\mathbf{x}, \Delta\mathbf{Q}, \Delta\mathbf{P}, \Delta\mathbf{c})$
  For each $i' = i^*$ :
    Draw **com**$_{i^*}$ at random
Compute initial commitments **Com** $= Hash($**com**$_1, \ldots,$ **com**$_{i^*}, \ldots,$ **com**$_{N_H^D},$ **com**$_{aux})$

**Step 4: Generate Party Communications**
  Draw $[\![\alpha]\!]_{i^*}, [\![\beta]\!]_{i^*}$ uniformly at random from their respective domains
  For each $k \in [D]$ :
    if $i_k \neq i_k^*$ :
      Get communications $\{[\![\alpha]\!]_{i_k}, [\![\beta]\!]_{i_k}, [\![\mathbf{v}]\!]_{i_k}\}$ as stated in Protocol 7.1 and 7.4
    if $i_k^*$ :
      Compute party communication shares $[\![\alpha]\!]_{i_k^*}, [\![\beta]\!]_{i_k^*}, [\![\mathbf{v}]\!]_{i_k^*}$ by running the MPC protocol
      on the sum of the witnesses of the $N-1$ revealed leaf-party shares, as described
      in Protocol 7.1 and add $[\![\alpha]\!]_{i^*}, [\![\beta]\!]_{i^*}$
      Set $[\![\mathbf{v}]\!]_{i^*} = -\sum_{i' \neq i^*} [\![\mathbf{v}]\!]_{i^*}$

**Step 5: Output the transcript**
  $RSP_1 = h' = Hash(H_1, \ldots, H_D)$ where $H_k \leftarrow$ Protocol 7.4$([\![\mathbf{x}]\!], [\![\mathbf{Q}]\!], [\![\mathbf{P}]\!], [\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\![\mathbf{c}]\!], \mathsf{z}, \epsilon)$
  $RSP_2 =$ **com**$_{i^*}, [\![\alpha]\!]_{i^*}, [\![\beta]\!]_{i^*}, \{(seed_{i_1,\ldots,i_D}, p_{i_1,\ldots,i_D}) \forall (i_1, \ldots, i_D) \neq i_1^*, \ldots, i_D^*)\}$
  **Return** (**Com**$, CH_1, RSP_1, CH_2, RSP_2)$
```

Protocol 7.6: This protocol shows the mode of operation of the simulator $\mathcal{S}$ described in the honest-verifier zero-knowledge proof. This simulator creates a transcript of our SDitH protocol without knowing the secret $\mathbf{x}$, which is indistinguishable from the transcript of an honest execution of our SDitH protocol.

To establish the soundness of our SDitH protocol, we must show that a malicious prover, without the knowledge of a solution $\mathbf{x}$, is unlikely to convince an honest verifier.

Proof sketch: Similar to the soundness proof of Section 5.1.2, we follow the soundness proof of the original SDitH paper [FJR22], but also orient ourselves on the soundness proof of [AGH$^+$22]. Our main differences from both papers lie in the details of how we extract the witness. For this extraction, we run D instances of the SDitH in parallel, where the party instance is shared using the Small Integer Sharing. These shares are then rearranged into the hypercube geometry and committed as part of the first message. The main difference between the approach from [AGH$^+$22] and our approach are the auxiliary values.

The proof for the extraction follows the general idea that we can extract a witness $\mathbf{x}'$ that satisfies $\mathsf{H} \cdot \mathbf{x}' = \mathsf{y}$ and $wt(\mathbf{x}') \leq w$ after seeing two accepting transcripts that agree on the first commitments, but disagree on the second challenge. This is because we always open all but one share after the second challenge. By committing to the same secret shares and changing the hidden share (second challenge), we ensure that all secret shares are opened through both transcripts. This allows us to reconstruct the original witness. The remaining argument is to show that this is sufficient for an extraction.

Instead of following the original soundness proof of the SDitH protocol, we will follow the soundness proof of the hypercube geometry, which states that an extraction is possible as long as the state and the communication of all parties are verified in at least one accepting transcript. This follows from the general hypercube geometry, which optimizes the verification process for the verifier by reducing the number of MPC runs needed to prove that the prover knows a good witness. Thus, by opening all but one commitment in each of the accepting transcripts, where the hidden party is different between the two transcripts, we make sure that the hidden party is part of at least one main-party that is verified in the other transcript. Here, we argue that the smaller vectors of the leaf-parties used to generate the main-party shares still allow us to utilize the extraction argument. The general idea is that the auxiliary values are calculated on the extended sum of the main-party shares, allowing us to utilize the main-party shares to extract the witness. Thus, by using the injective deterministic hash function $\mathcal{SH}$ for the sum extension, we need to know the opening of all shares to calculate the correct sum of main-party shares. After that, we can extract the original witness by calculating the sum of the main-party shares of one dimension, extending it using the aforementioned hash function, and adding the auxiliary value to it.

Soundness

Given the false positive probability p bound by Equation 3.10, we can construct an extraction function $\mathcal{E}$, which produces a good witness $\mathbf{x}'$, if a malicious efficient prover $\tilde{\mathsf{P}}$ with knowledge of the public parameters of the SDitH protocol (H, y) can convince an honest verifier V with probability:

$$\tilde{\varepsilon} = Pr\left[\langle \tilde{\mathsf{P}}, \mathsf{V} \rangle \to 1\right] > \varepsilon = \left(p + (1-p) \cdot \frac{1}{N_H^D}\right) \tag{7.7}$$

The extracted good witness must satisfy $\mathsf{H} \cdot \mathbf{x}' = \mathsf{y}$ as well as $wt(\mathbf{x}') \leq w$ and must be obtained by making an average number of calls to the malicious prover of:

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \frac{2 \cdot \tilde{\varepsilon} \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}\right) \tag{7.8}$$

Note that if a malicious prover cheats successfully with a probability lower than ε, it is considered normal cheating, equivalent to randomly guessing a correct solution.

Proof: Let us assume that the commitment scheme is perfectly binding. Then we have two transcripts with the same initial commitments $h = Hash(\textbf{com}_1, \ldots, \textbf{com}_{N_H^D}, \textbf{com}_{aux})$, but different hidden leaf-party shares $i^* \neq j^*$. This leaves us with two options:

- $[\![\mathbf{x}]\!], [\![\mathbf{Q}]\!], [\![\mathbf{P}]\!]$ differ between the transcripts and, thus, one finds a collision in the commitment hash
- In both transcripts, the openings are the same, and thus $[\![\mathbf{x}]\!], [\![\mathbf{Q}]\!], [\![\mathbf{P}]\!]$ are equal in both transcripts as well

If we look at the second case, we can see that the witness can be recovered from the two transcripts with $i^* \neq j^*$ and $i^*, j^* \in [N_H^D]$ because they were generated with different challenge points from the verifier. This is because, in the first transcript, the verifier receives $N_H^D - 1$ opened leaf-party shares where the i^* leaf-party share remains hidden. In the second transcript, the verifier also receives $N_H^D - 1$ opened leaf-party shares, but

Soundness

the j^* leaf-party share stays hidden this time. As the hidden shares differ in both transcripts while the initial commits are the same, one receives $N_H^D - 1$ opened shares and can extract the hidden leaf-party share i^* from the second transcript. Thus, one can access all opened leaf-party shares and reconstruct the witness following the reconstruction described in Section 7.1.2.

This implies that the extractor function $\mathcal{E}$ can obtain a good witness. First of all, let us consider the hypercube geometry in which $i^* \neq j^*$ means that the corresponding list of indices in the hypercube differ in at least one position, such that $(i_1^*, \ldots, i_D^*) \neq (j_1^*, \ldots, j_D^*)$. Thus, let us assume that they differ in the first coordinate denoted as $i_k^* \neq j_k^*$. We, therefore, have two transcripts with different hidden main-party shares, where both sums have been successfully verified. Furthermore, we know that the verification is only successful if the sum of the main-party shares has been successfully extended. This scenario closely resembles the protocol of [FJR22, AGH$^+$22], and we can follow their structure of the soundness proof.

We first need to show that in order to generate these two accepting transcripts that contain the same initial commitments but with different challenge points in the second challenge, the witness must be good. Furthermore, we need to define what makes a witness a good witness. Recall that $[\![\mathbf{x}]\!]$ is the sharing of $\mathbf{x}$ and each share of $\mathbf{S}$ is interpolated over $[\![\mathbf{x}]\!]$. Thus, a witness $\mathbf{x}$ is good if for $[\![\mathbf{x}]\!], [\![\mathbf{Q}]\!], [\![\mathbf{P}]\!]$ the following holds:

$$\mathbf{S} \cdot \mathbf{Q} = \mathbf{P} \cdot \mathbf{F}$$

After this, we denote the random variable for the randomness of the initial commits as R_h, and one value of this variable is denoted as r_h. Additionally, we call the execution of the entire protocol $\tilde{\mathsf{P}}$ to obtain a first successful transcript T_1 the *outer loop* and the execution of the protocol on the same randomness to acquire a second successful transcript T_2 with a different hidden leaf-part but same initial commitments the *inner loop*.

Now, we can examine how the extractor obtains the first successful transcripts. For this, the extraction algorithm $\mathcal{E}$ follows the forking Lemma 5.

Soundness

By running the Protocol 7.1 with $\tilde{\mathsf{P}}$ and the honest verifier V until a successful transcript T_1 is found. This transcript utilized the randomness r_h for their commitments and has i^* as its second challenge. After that, the process is repeated using the same randomness for the commitments r_h until a second successful transcript is found, with a different second challenge j^*. Finally, the extractor derives the corresponding witness and starts over if the witness is not a good one.
We further need to estimate the number of calls the extractor $\mathcal{E}$ makes to the prover. For this, let $\alpha \in (0, 1)$ be chosen such that $(1 - \alpha) \cdot \tilde{\varepsilon} \geq \varepsilon$ and we consider the randomness r_h as 'good' if it satisfies $Pr[succ_{\tilde{\mathsf{P}}}|r_h] \geq (1-\alpha)\cdot\tilde{\varepsilon}$. The splitting Lemma 4 implies that we can transform this into $Pr[r_h \text{ is good}|succ_{\tilde{\mathsf{P}}}] \geq \alpha$. This entails that we can find a good witness within about $\frac{1}{\alpha}$ many transcripts.
Moreover, by the converse of Lemma 4, when r_h is classified as good meaning the probability $(1 - \alpha)\tilde{\varepsilon}$ exceeds ε the initial commitment within the transcript must necessarily encode a valid witness. Consequently, this witness may be reliably extracted from any other successful transcript generated with the same randomness r_h.
Let us assume we have a good transcript T_1. We can now determine a lower bound of the number of runs of the inner loop to obtain another good transcript T_2 that differs in the second challenge. For this, our SDitH protocol is run on the same initial r_h until a second good transcript T_2 with $i^* \neq j^*$ is found. Thus, we can establish the lower bound to obtain T_2 as:

$$\begin{aligned}
Pr[succ_{\tilde{\mathsf{P}}} \cap i^* \neq j^* | r_h \text{ good}] &= Pr[succ_{\tilde{\mathsf{P}}}|r_h \text{ good}] - Pr[succ_{\tilde{\mathsf{P}}} \cap i^* = j^*|r_h \text{ good}] \\
&\geq Pr[succ_{\tilde{\mathsf{P}}}|r_h \text{ good}] - \frac{1}{N_H^D} \\
&\geq (1-\alpha)\cdot\tilde{\varepsilon} - \frac{1}{N_H^D} \\
&\geq (1-\alpha)\cdot\tilde{\varepsilon} - \varepsilon
\end{aligned}$$

Using this probability, we can identify how often $\tilde{\mathsf{P}}$ needs to be run (outer loop) to obtain a second success transcript T_2 with probability greater than $\frac{1}{2}$. This transcript T_2 must have a different challenge leaf-party from

Soundness

T_1 but is generated with the same randomness as T_1. The resulting number of executions L of $\tilde{\mathsf{P}}$ is:

$$L > \frac{\ln(2)}{\ln\left(\frac{1}{1-((1-\alpha)\cdot\tilde{\varepsilon}-\varepsilon)}\right)} \simeq \frac{\ln(2)}{(1-\alpha)\cdot\tilde{\varepsilon}-\varepsilon} \tag{7.9}$$

Let us denote the number of expected calls from $\mathcal{E}$ to $\tilde{\mathsf{P}}$ as $E_{\tilde{\mathsf{P}}}$. This number of calls can be written as a recursive formula, in which firstly, we have the probability of successfully obtaining the first transcript T_1 (outer loop) as $1-Pr[succ_{\tilde{\mathsf{P}}}]$ and secondly, the probability of acquiring the second successful transcript T_2 (inner loop) as $\frac{1}{2}$ after L calls. We, therefore, have a mode of operation of the extraction algorithm $\mathcal{E}$ as follows:

1. Perform the initial call to $\tilde{\mathsf{P}}$
2. Repeat step 1 if one does not find a successful transcript
3. After finding a successful transcript T_1, we know that r_h is a good randomness with a probability of α due to the splitting Lemma 4. Thus, one makes L many calls to the prover $\tilde{\mathsf{P}}$ to obtain the second transcript with probability $\frac{1}{2}$. If one does not find a second successful transcript, return to step 1; otherwise, terminate.
4. Because one does not know that the randomness r_h is bad (executing $\tilde{\mathsf{P}}$ on r_h cannot yield a successful transcript) with a probability of $1-\alpha$ after the first transcript, she must perform L calls to $\tilde{\mathsf{P}}$. This shows whether she can obtain a successful transcript T_2 or not and prevents her from running into an endless recursion. If no successful transcript was found after L calls, return to step 1.

However, this mode of operation does not provide an upper bound on the number of calls $\mathcal{E}$ makes to $\tilde{\mathsf{P}}$ (step 1). This is because if r_h is a bad randomness with probability $1-\alpha$, there is no guarantee that one finds a successful transcript T_2. We, therefore, denote this scenario as always unsuccessful and thus have the probability of finding no successful transcript T_2 with a successful T_1 of:

$$Pr[\text{no } T_2|succ_{\tilde{\mathsf{P}}}] = Pr[\text{no } T_2|r_h \text{ good}] + Pr[\text{no } T_2|r_h \text{ bad}] = \frac{\alpha}{2} + (1-\alpha) \tag{7.10}$$

Soundness

In this case, $\mathcal{E}$ returns to step 1, and we note the expected number of calls $E_{\tilde{\mathsf{P}}}$ to $\tilde{\mathsf{P}}$ as:

$$E_{\tilde{\mathsf{P}}} \leq 1 + \underbrace{(1 - Pr[succ_{\tilde{\mathsf{P}}}]) \cdot E_{\tilde{\mathsf{P}}}}_{\text{Not finding } T_1} + \underbrace{Pr[succ_{\tilde{\mathsf{P}}}] \cdot \left(L + \left(1 - \frac{\alpha}{2}\right) \cdot E_{\tilde{\mathsf{P}}}\right)}_{\text{Finding } T_1} \tag{7.11}$$

This can be reduced to

$$E_{\tilde{\mathsf{P}}} \leq \frac{2}{\alpha \cdot \tilde{\varepsilon}} \cdot (1 + L \cdot \tilde{\varepsilon}) = \frac{2}{\alpha \cdot \tilde{\varepsilon}} \cdot \left(1 + \frac{\tilde{\varepsilon} \cdot \ln(2)}{(1-\alpha) \cdot \tilde{\varepsilon} - \varepsilon}\right) \tag{7.12}$$

As the final step, we want to simplify the expected number of calls $E_{\tilde{\mathsf{P}}}$ in terms of only two key parameters: the soundness ε of the protocol and the success probability $\tilde{\varepsilon}$ of a malicious prover convincing an honest verifier. For this we define the probability $(1-\alpha) \cdot \tilde{\varepsilon}$ as the midpoint between ε and $\tilde{\varepsilon}$. This midpoint is a valid intermediate value that captures a balanced approximation between the worst-case success rate of a malicious prover ($\tilde{\varepsilon}$) and the base soundness of the protocol (ε). Thus we get $(1-\alpha) \cdot \tilde{\varepsilon} = \frac{1}{2} \cdot (\tilde{\varepsilon} + \varepsilon)$ and transform Equation 7.12 to get the following upper bound for the expected number of calls from $\mathcal{E}$ to $\tilde{\mathsf{P}}$ as:

$$E_{\tilde{\mathsf{P}}} \leq \frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \frac{2 \cdot \tilde{\varepsilon} \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}\right) \tag{7.13}$$

This concludes the security proof of our protocol, where we demonstrated the security properties of our SDitH protocol by analyzing its completeness, HVZK, and soundness characteristics. We first defined the abort events in Definition 7.3 and quantified their probability, ensuring that protocol termination does not reveal information about the secret. We then proved the protocol's perfect completeness, showing that an honest prover will be accepted by an honest verifier with probability $1 - Pr[\text{abort}]$. Additionally, we constructed a simulator to confirm the protocol's honest-verifier zero-knowledge (HVZK) property, ensuring no witness information leakage under an honest verifier. For soundness, we established that a malicious prover without a valid witness has a success probability bounded by $\varepsilon = p + \frac{1-p}{N_H^D}$. Finally, we analyzed the expected calls to a malicious prover, showing that extraction is efficient within the protocol's security limits. Together, these results affirm the protocol's robustness against adversaries while preserving zero-knowledge guarantees. From this, we dive into the communication and computational cost discussion, where we compare our SDitH protocol to the original SDitH protocol [FJR21] and the optimized implementation of [AGH$^+$22]. After that, we give an overview of possible future work.

8 Discussion

In this section, we compare our optimization to current state-of-the-art implementations regarding communication and computational costs. To visualize the improvements of the different optimizations, we first provide a communication costs baseline in the next paragraph and add the modifications described in Section 7 until we reach our final Protocol 7.1. Then, we compare our protocol to the implementation from [AGH+22], which uses the hypercube geometry. Here, it is important to note that the paper [BB24] does mention the SDitH with the hypercube geometry and the OneTree structure but focuses on VolE-based implementations. It does not provide the communication costs for the SDitH implementation. Furthermore, they do not give the effectiveness of the OneTree structure, which we showed in Equation 6.1. This is why we use the communication costs from [AGH+22] instead. Additionally, we will assign the values from the hypercube paper [AGH+22] to the parameters marked **Original** for the communication costs, and utilize the values marked **Comp** for the computational. These parameters are denoted in the tables 8.1, 8.2. We also implemented a proof of concept in Python that could not run for the original parameter values in a reasonable time. This is the reason for the difference between the communication and computational variants. Furthermore, as we discuss later in this section, we do not need to utilize larger values to see the difference between the implementations.

Table 8.1: The SD and MPC parameters for our protocol, where the values for **Original** are mostly taken from [AGH+22] and the **Comp** are smaller parameters for reasonable execution times.

Scheme	**SD Parameters**				**MPC Parameters**			
	SD	m	k	w	$\lvert\mathbb{F}_{\text{poly}}\rvert$	$\lvert\mathbb{F}_{\text{points}}\rvert$	t	p
Original	2	$1\,280$	640	132	2^{13}	2^{26}	6	$\approx 2^{-69}$
Comp	2	32	16	8	2^{13}	2^{26}	6	$\approx 2^{-69}$

Table 8.2: The rest of the SDitH parameters based on [AGH+22].

Scheme	Parameters						
	N	N_H	D	τ	λ	A	sm
Original	256	2	8	17	128	2^{13}	10
Comp	32	2	5	1	128	2^{13}	10

8.1 Communication Cost Analysis

For the baseline of communication costs, we take the additive secret sharing from Section 3.6, which has the biggest impact on the communication costs of MPCitH protocols. This AddSS samples $N-1$ uniformly random vectors $[\![\mathbf{x}]\!]_i \in \mathbb{F}_{SD}^m$ and calculates the final share via $[\![\mathbf{x}]\!]_N = \mathbf{x} - \sum_{i=1}^{N} -1 \in \mathbb{F}_{SD}^m$. After that, the N shares are communicated, where each share is a m long vector that contains elements from $\mathbb{F}_{SD}$. Each of these elements is encoded into bits, leading to $\log_2(SD)$. Thus, we need a vector of length m, where each element in $\mathbb{F}_{SD}$ is in its bit representation costing $\log_2(SD)$. The resulting communication cost for the secret sharing is as follows:

$$costs_{sharing} = N \cdot \log_2(SD) \cdot m \text{ bits}$$

In addition to the communication costs of the secret sharing, we need to account for sending the auxiliary values of the polynomials $[\![\mathbf{Q}]\!]_N, [\![\mathbf{P}]\!]_N = (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|)$, the beaver triples $\{[\![\alpha_j]\!]_{i^*}, [\![\beta_j]\!]_{i^*}, [\![\mathbf{v}_j]\!]_N\} = (2 \cdot t) \cdot \log_2(|\mathbb{F}_{poly}|)$, as well as the hidden commitment **com**$_{i^*} = 2 \cdot \lambda$. We also need the two sent hashes h and h', which are each $2 \cdot \lambda$. Furthermore, we need to send all this τ times for each execution of the SDitH protocol, resulting in the following baseline for the communication costs of the original SDitH protocol from [FJR22]:

$$\begin{aligned} costs_{sharing} = 4 \cdot \lambda + \tau \cdot (N \cdot \log_2(SD) \cdot m + (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) \\ + (3 \cdot t) \cdot \log_2(|\mathbb{F}_{points}|) + 2 \cdot \lambda) \text{ bits} \end{aligned}$$

From this, we get the baseline communication costs for secret sharing by plugging in the **Original** values for the parameters of:

$$\begin{aligned} costs_{sharing} &= 4 \cdot 128 + \tau \cdot (256 \cdot \log_2(2) \cdot 1\,280 + (2 \cdot 132) \cdot \log_2(2^{11}) \\ &+ (3 \cdot 6) \cdot \log_2(2^{22}) + 2 \cdot 128) \text{ bits} \\ &= 3\,943\,941 \text{ bits} \end{aligned}$$

If we look at the additive aspect of secret sharing in the cost function, we can see that it uses $256 \cdot \log_2(2) \cdot 1\,280 = 227\,130$ bits. We can improve this by exchanging it with our first optimization, the small integer secret sharing for syndrome decoding as described in 7.1.1. This uses $N \cdot sm \cdot \log_2(A-1) + m$, where sm is the length of the shorter shares and A is the security parameter, defining the size of the number space used to sample the shares uniformly. In this case, A is chosen as 2^{13} because to map between the different number spaces, we need to make sure that they are multiples of each other. Thus, we need to increase the number space of the challenge points from 2^{22} to 2^{26} and the number space of the polynomials to match A. Thus, increasing A to achieve a lower rejection rate than 0.02 (for $A = 2^{13}$) would significantly increase the cost of communication. Larger A affects the length of the sharings, which must ensure a low probability of random guessing. We define the length sm of a share as $sm = \frac{\lambda}{q} = \frac{128}{13} \approx 10$, which ensures that each share has a probability of being correctly guessed by an attacker of $A^{-sm} = 2^{13 \cdot (-10)} \geq 2^{-128}$. The communication costs of the Small Integer Sharing on the syndrome decoding problem are therefore $256 \cdot \frac{128}{13} \cdot \log_2(2^{13}) + 1\,280 = 34\,048$ bits. This is 6.67 times smaller than the additive secret sharing baseline. In addition, we utilize the structure of the syndrome decoding matrix H, which can be represented by $\mathsf{H} = (\mathsf{H}'|\mathsf{I}_{m-k})$, which in turn allows us to carry out the protocol on the smaller part of $\mathbf{x}$, namely $\mathbf{x}_A$ as described in Section 3.10. The resulting communication costs using SIS are the following:

$$\begin{aligned} costs_{comm} &= 4 \cdot \lambda + \tau \cdot (N \cdot sm \cdot \log_2(A-1) + k \cdot \log_2(|\mathbb{F}_{SD}|) + (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) \\ &+ (3 \cdot t) \cdot \log_2(|\mathbb{F}_{points}|) + 2 \cdot \lambda) \text{ bits} \\ &= 4 \cdot 128 + \tau \cdot (256 \cdot \frac{128}{13} \cdot \log_2(2^{13}) + 640 \cdot \log_2(2) + (2 \cdot 132) \cdot \log_2(2^{13}) \\ &+ (3 \cdot 6) \cdot \log_2(2^{26}) + 2 \cdot 128) \text{ bits} \\ &= 454\,795 \text{ bits} \end{aligned}$$

Looking at this communication cost, we can see that SIS optimization reduces the communication cost of the baseline ($3\,943\,941$ bits) by a factor of approximately 8.7. The second optimization of our protocol, namely the rearrangement of the shares into the hypercube structure, does not impact the communication costs. However, by combining it with the Small Integer Sharing, we send the shares for α and β ($[\![\alpha]\!]_{(i_1^*,\ldots,i_D^*)}, [\![\beta]\!]_{(i_1^*,\ldots,i_D^*)}$) in the smaller $\mathbb{F}_A$ space instead of $\mathbb{F}_{points}$. Nevertheless, in order for us to utilize the SIS in combination with the hypercube scheme, we need to perform the protocol over $\mathbf{x}$ again. Furthermore, note that by using the hypercube structure, shares are called leaf-party shares, and the number of leaf-party shares is denoted as N_H^D instead of N, where $N_H^D = N$. The communication costs, therefore, change to:

$$\begin{aligned}
costs_{comm} &= 4 \cdot \lambda + \tau \cdot (N_H^D \cdot sm \cdot \log_2(A-1) + m + (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) \\
&+ (2 \cdot t) \cdot \log_2(|\mathbb{F}_A|) + t \cdot \log_2(|F_{points}|) + 2 \cdot \lambda) \\
&= 4 \cdot 128 + \tau \cdot (256 \cdot \frac{128}{16} \cdot \log_2(2^{13}) + 1\,280 + (2 \cdot 132) \cdot \log_2(2^{13}) \\
&+ (2 \cdot 6) \cdot \log_2(2^{13}) + 6 \cdot \log_2(2^{26}) + 2 \cdot 128) \\
&= 452\,551 \text{ bits}
\end{aligned}$$

After this, we can integrate our final optimization, which we split into two parts to visualize the impact of the optimization better. The first aspect is utilizing the GGM tree-based pseudo-random generators. These allow us to send only the sibling paths, which reduces the number of shares sent to $\log_2(N)$ instead of N. In addition, this significantly reduces the effectiveness of the Small Integer Sharing on the communication costs because no individual shares are sent to the verifier but only the seeds of the GGM tree. These seeds are of length λ ($|seed| = \lambda$). This is not considerably cheaper than our sharing with $sm \cdot \log_2(A) = 10 \cdot \log_2(2^{13}) = 130$ on a sharing level ($\lambda = 128$), but allows us to send fewer shares and add the last optimization in the next paragraph. The resulting communication costs can be seen in the next equation.

$$\begin{aligned} costs_{comm} &= 4 \cdot \lambda + \tau \cdot (\log_2(N_H^D) \cdot \lambda + m \cdot \log_2(\mathbb{F}_{SD}) + (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) \\ &+ (2 \cdot t) \cdot \log_2(|\mathbb{F}_{poly}|) + 2 \cdot \lambda) \\ &= 4 \cdot 128 + \tau \cdot (\log_2(256) \cdot \log_2(2^{13}) + 1\,280 \cdot \log_2(2) + (2 \cdot 132) \cdot \log_2(2^{13}) \\ &+ (2 \cdot 6) \cdot \log_2(2^{26}) + 2 \cdot 128) \\ &= 127\,338 \text{ bits} \end{aligned}$$

As we can see, by sending the sibling path of the GGM tree, we reduce the number of shares that need to be sent drastically and also reduce the cost of sending each share, which results in a significant improvement. Next, we further reduce the number of nodes that need to be sent using the OneTree strategy, which allows the prover to send about 60% of the sibling path nodes overall τ executions. Our final communication costs are, therefore:

$$\begin{aligned} costs_{comm} &= 4 \cdot \lambda + 0.6 \cdot (\tau \cdot (\log_2(N_H^D) \cdot \lambda) + \tau \cdot (m \cdot \log_2(\mathbb{F}_{SD}) + (2 \cdot w) \cdot \log_2(|\mathbb{F}_{poly}|) \\ &+ (2 \cdot t) \cdot \log_2(|\mathbb{F}_{poly}|) + 2 \cdot \lambda) \\ &= 4 \cdot 128 + 0.6 \cdot (17 \cdot (\log_2(256) \cdot \log_2(2^{13})) + 1\,280 \cdot \log_2(2) + (2 \cdot 132) \cdot \log_2(2^{13}) \\ &+ (2 \cdot 6) \cdot \log_2(2^{26}) + 2 \cdot 128) \\ &= 99\,190 \text{ bits} \end{aligned}$$

Let us compare the final communication costs to the original syndrome decoding in the head protocol with additive secret sharing (baseline). We get an improvement by a factor of approximately 40, which brings the effectiveness of the entire protocol to a competitive area. From here, we will dive into the computational costs, which will show the improvements of our protocol compared to the original SDitH protocol with additive secret sharing and the SDitH with the hypercube scheme, where the latter is close to our protocol in terms of communication costs ($99\,190$ bits to $90\,214$ bits).

8.2 Computational Cost Analysis

For the computational cost analysis, we implemented a version of the original SDitH with additive secret sharing, the hypercube-based SDitH with the same

Table 8.3: This table shows the execution times of the three different protocols: $1)$ SDitH with AddSS, $2)$ SDitH with AddSS and hypercube structure, $3)$ our SDitH, for our Python implementation in seconds, for different runs as well as on average. We utilized the parameter setting **Comp** provided in Table 8.2 for this.

Protocol	**Runs**					**Average**
	1	2	3	4	5	
Original SDitH	47.60	48.66	48.69	48.05	47.08	48.02
Hypercube SDitH	4.84	3.72	3.85	4.90	4.25	4.31
SIS Hypercube SDitH	2.89	1.89	2.44	1.67	1.69	2.12

sharing, and our protocol in Python, using the mathematical software system Sagemath for Python [Sag]. Sagemath allows us to execute the protocol on prime finite fields, which is needed to ensure the correct mode of operation. However, these prime fields significantly impact the execution speed, as every number is represented by a polynomial, which is less optimized, especially in Python. This slowdown is the reason for the small values of the parameters used in the computational cost analysis, shown in Table 8.1 under **Comp**. At this point, it is important to note that the implementation is not optimized, and we do not recommend a Python implementation of any of these protocols for practical usage. The implementation is merely a means to demonstrate the different computation times and highlight the improvements they provide.
We ran each protocol five times to have statistically relevant results and recorded the execution time. After that, we calculated their averages for better comparison. These results can be seen in Table 8.3.
Looking at the average execution times, we can see that the integration of the hypercube structure reduces the computational costs by a factor of approximately 10, which is on par with the findings of the corresponding paper by Aguilar-Melchor et al. [AGH$^+$22]. Furthermore, we can see that our approach is about 2 times faster than the SDitH protocol with the hypercube structure and AddSS. This is because the shares used for the hypercube construction and, consequently, for executing the MPC protocol in round 2 of the prover and the verification round of the verifier are $sm = \lambda/q$ long. Here q is the power of the number space for the shares $A = 2^q$, instead of k long with $k = |\mathbf{x}_A|$. The only values that are dependent on the length of the secret $\mathbf{x}$ are the initial polynomials $\mathbf{S}, \mathbf{Q}, \mathbf{P}, \mathbf{F}$, and the auxiliary values. This, in turn, means that in contrast

Table 8.4: This table shows the execution times of the SDitH protocol with AddSS and the hypercube structure and our protocol on a larger parameter setting in seconds, namely $m = 256, k = 128$ and $w = 32$, in order to show the slower rising computational times regarding the length of the secret $\mathbf{x}$.

Protocol	**Runs**					**Average**
	1	2	3	4	5	
Hypercube SDitH	112.38	115.17	120.47	114.22	117.56	115.96
SIS Hypercube SDitH	31.96	38.19	40.63	35.69	34.14	36.12

to the hypercube SDitH protocol, ours scales better with the size of $\mathbf{x}$ and allows us to achieve reasonable execution times even with a more realistic parameter setting.

We show this by running both implementations with $m = 256, k = 128$, and $w = 32$, which can be seen in Table 8.4. These results show that the improvement of our protocol increases from a factor 2 to a factor 3.25 reduction in computational costs. Furthermore, using the Python c profiler, we see that the initial interpolation of $\mathbf{x}$ to obtain the polynomial $\mathbf{S}$ has the highest impact on execution time because we use the Lagrange interpolation, which is prone to be slow. Note that any interpolation algorithm creates significant overhead in terms of computational costs.

In this section, we analyzed our protocol's communication and computational costs compared to different SDitH implementations. To illustrate the impact of each optimization, we began by establishing a baseline for communication costs and then incrementally applied the enhancements introduced in Section 7, until we reach our final Protocol 7.1. For this, we use the parameters from [AGH+22] to ensure a consistent basis for comparison. After that, we compare our protocol to the original SDitH protocol and the SDitH protocol with the hypercube structure regarding their computational costs. For this, we implemented a proof-of-concept in Python and executed each protocol multiple times on a smaller parameter setting.

Our findings demonstrate that our protocol achieves an approximately 40-fold reduction in communication costs compared to the original SDitH protocol while also outperforming previous implementations in computational efficiency, particularly for larger parameter sets by a factor of 3.25. This reduction is achieved by combining the SDitH protocol with the SIS and the hypercube structure, allowing us to handle larger secrets with minimal additional cost. With all this in mind, we will dive into further possible improvements to our protocol.

8.3 Future Work

Firstly, communication costs could be further reduced using half-tree pseudo-number generators introduced by [GYW+22] to further minimize the number of sent nodes. Another interesting question would be whether the Small Integer Sharing is compatible with the VOLE problem, which is another alternative to the syndrome decoding problem. And lastly, one could try to reduce the rejection rate of our protocol or even get rid of it to allow for a smaller A. This would also scale down the size of $\mathbb{F}_{poly}$ and $\mathbb{F}_{points}$, thus improving the communication and computational cost.

9 Conclusion

In this thesis, we introduced a novel approach to optimize the performance of syndrome decoding in a zero-knowledge proof system, focusing on using *Small Integer Sharing* (SIS) and hypercube structures. Through careful performance analysis, we demonstrated that the proposed method significantly reduces computational overhead through our proof-of-concept implementation, achieving a doubled improvement in efficiency for small parameters and up to factor 3.25 for more realistic parameter settings compared to the state-of-the-art SDitH with the hypercube structure by [AGH+22]. In addition, we showed that our protocol achieves a factor 40 improvement in communication costs compared to the original SDitH protocol from [FJR22] and is on par with the state-of-the-art SDitH protocol of [AGH+22].

To achieve this, we presented and combined several enhancements to the original SDitH protocol, such as integrating the hypercube structure to reorganize share distribution, utilizing the OneTree architecture, and employing an SIS strategy to minimize the length of the secret shares. These optimizations reduced the required operations, particularly in larger instances, making the protocol more scalable.

Future work should investigate additional strategies to reduce communication costs, such as incorporating pseudo-number generators like half-tree PRG or exploring compatibility with other cryptographic problems like VOLE. These improvements would further enhance the protocol's practical applicability in real-world settings.

In conclusion, this thesis contributes to the field of efficient zero-knowledge proofs by introducing both theoretical advancements and practical techniques for syndrome decoding optimization, opening avenues for future research and implementation.

Bibliography

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security, 2002.

[AGH$^+$22] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The Return of the SDitH, 2022.

[BB24] Carsten Baum and Ward Beullens. One Tree to Rule Them All - Optimizing GGM Trees and OWFs for Post-Quantum Signatures. *Cryptology ePrint Archive*, 2024.

[BBC$^+$19] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. A Finite Regime Analysis of Information Set Decoding Algorithms. *Algorithms*, 12(10):209, October 2019.

[BBM$^+$24] Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures, 2024.

[Bea92] Donald Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, Lecture Notes in Computer Science, pages 420–432, Berlin, Heidelberg, 1992. Springer.

[BMVT78] E. Berlekamp, R. McEliece, and H. Van Tilborg. On the inherent intractability of certain coding problems (Corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, May 1978.

[BN19] Carsten Baum and Ariel Nof. Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography, 2019.

[Che24] Yilei Chen. Quantum Algorithms for Lattice Problems, 2024.

[CTS16] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of Information Set Decoding for a Sub-linear Error Weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography*, volume 9606, pages 144–161. Springer International Publishing, Cham, 2016.

[DGV$^+$16] Özgür Dagdelen, David Galindo, Pascal Véron, Sidi Mohamed El Yousfi Alaoui, and Pierre-Louis Cayrel. Extended security arguments for signature schemes. *Designs, Codes and Cryptography*, 78(2):441–461, February 2016.

[FJR21] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Shared Permutation for Syndrome Decoding: New Zero-Knowledge Protocol and Code-Based Signature, 2021.

[FJR22] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs, 2022.

[FMRV22] Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection, 2022.

[FS06] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, volume 263, pages 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, August 1986.

[Gou01] Louis Goubin. A sound method for switching between boolean and arithmetic masking. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems CHES 2001*, volume 2162, pages 3–15. Springer Berlin Heidelberg, 2001. Series Title: Lecture Notes in Computer Science.

[GYW$^+$22] 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, 2022.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, page 2130, New York, NY, USA, 2007. Association for Computing Machinery.

[KZ20] Daniel Kales and Greg Zaverucha. An Attack on Some Signature Schemes Constructed From Five-Pass Identification Schemes, 2020.

[KZ22] Daniel Kales and Greg Zaverucha. Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures, 2022.

[LN17] Yehuda Lindell and Ariel Nof. A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority, 2017.

[PS00] David Pointcheval and Jacques Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

[Sag] Sage. SageMath Mathematical Software System - Sage. https://www.sagemath.org/.

[VG23] Madhu Sudan Venkatesan Guruswami, Atri Rudra. Essential coding theory, 2023.