UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

# A Comparison of MPCitH and VOLEitH - Optimisations for Post-Quantum Signatures

*Ein Vergleich zwischen MPCitH und VOLEitH - Optimierungen für post-quanten Signaturen*

**Masterarbeit**

im Rahmen des Studiengangs
**IT-Sicherheit**
der Universität zu Lübeck

vorgelegt von
**Eric Landthaler**

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

mit Unterstützung von
**M. Sc. Paula Arnold**

Lübeck, 05. Dezember 2024

## Abstract

Digital signatures are becoming increasingly important and with the advent of quantum computers, attackers are gaining access to more and more resources. Therefore, new signature procedures that are also secure against this type of attack must be developed. Current post-quantum protocols are often based on the Multi-Party Computation in the Head (MPCitH) paradigm, where digital signatures are created using Zero-Knowledge Proofs (ZKP) and applying the Fiat-Shamir Transformation to get non-interactive protocols. A novel approach uses Vector Oblivious Linear Evaluation (VOLE) to generate such proofs, called VOLEitH, and shows significant improvements in complexity and communication overhead. In this work, we describe the VOLEitH approach in detail. We show how the protocol is structured and how digital signatures can be created. We also discuss which countermeasures help to expose a malicious prover. This provides a better understanding of VOLEitH and helps in future work, as VOLEitH has so far been viewed as a black box. Using [KKW18] as a representative for MPCitH approaches, we provide a comparison of the communication costs and computational complexity of VOLEitH to previous MPCitH protocols. We highlight that MPCitH and VOLEitH have the same structure but take different approaches to create ZKPs. We show the limitations of both approaches, such as poor scalability and dependencies on circuit size or number of parties, and develop theoretical improvements for digital signatures.

## Zusammenfassung

Digitale Signaturen werden immer wichtiger und mit dem Aufkommen von Quantencomputern erhalten Angreifer Zugang zu immer mehr Ressourcen. Daher müssen neue Signaturverfahren entwickelt werden, die auch gegen diese Art von Angriffen sicher sind. Derzeitige post-quanten Protokolle basieren häufig auf dem Multi-Party Computation in the Head (MPCitH) Paradigma, bei dem digitale Signaturen unter Verwendung von Zero-Knowledge Beweisen (ZKP) und der Anwendung der Fiat-Shamir-Transformation erstellt werden, um nicht-interaktive Protokolle zu erhalten. Ein neuartiger Ansatz verwendet Vector Oblivious Linear Evaluation (VOLE), um solche Beweise zu generieren, VOLEitH genannt. VOLEitH zeigt signifikante Verbesserungen bei der Komplexität und dem Kommunikations-Overhead. In dieser Arbeit beschreiben wir den VOLEitH-Ansatz im Detail. Wir zeigen, wie das Protokoll aufgebaut ist und wie digitale Signaturen erstellt werden können. Wir diskutieren auch, welche Gegenmaßnahmen dabei helfen, einen böswilligen Prover zu entlarven. Dies trägt zu einem besseren Verständnis von VOLEitH bei und hilft bei zukünftigen Arbeiten, da VOLEitH bisher als Blackbox betrachtet wurde. Unter Verwendung von [KKW18] als Repräsentant für MPCitH-Ansätze bieten wir einen Vergleich der Kommunikationskosten und der Rechenkomplexität von VOLEitH im Vergleich zu früheren MPCitH-Protokollen. Wir heben hervor, dass MPCitH und VOLEitH die gleiche Struktur haben, aber unterschiedliche Ansätze zur Erstellung von ZKPs verwenden. Wir zeigen die Grenzen beider Ansätze auf, wie die schlechte Skalierbarkeit und die Abhängigkeit von der Größe des Schaltkreises oder der Anzahl der Parteien, und entwickeln theoretische Verbesserungen für digitale Signaturen.

## Erklärung

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

Lübeck, 05. Dezember 2024

## Acknowledgements

I would like to thank Sebastian Berndt and Paula Arnold for their extensive support during the work and for answering all of my questions. Thanks also to Thomas Eisenbarth, who made this work possible. Also a special thanks to Jorge Andresen, Nicolas Hawighorst, Florian Marwitz and Jannik Westenfeld for proofreading this thesis.

# Contents

# 1 Introduction

Signatures are an important part of our daily life and they are also becoming more and more significant in the digital space. To create trust in signatures, no attacker must have the ability to forge the signatures of other entities and act on their behalf. With the advent of quantum computers, modern cryptography faces new challenges, as attackers can access significantly more resources and can therefore simply bypass current security mechanisms. An example is Shor's quantum algorithm [Sho94], which is designed to efficiently factorise large numbers or to find the Discrete Logarithm and thus break widely used cryptography, such as RSA or Elliptic Curves. Despite the early stage of development of quantum computers, their existence should be taken into account and considered when developing future protocols to be safe against this type of attacker. Previous approaches use Multi-Party Computation (MPC) to create Zero Knowledge Proofs (ZKP) and establish signatures on top of them, using the Fiat-Shamir Transformation (FS). To build efficient ZKPs, the prover splits his secret and simulates all parties of the MPC protocol "in his head". This framework is called MPC in the Head (MPCitH) and was first introduced by [IKOS07] in 2007. Since then, many optimisations have been developed and the first MPCitH signatures have been standardised. In 2023, [BBdSG+23b] introduced a novel approach that realises digital signatures based on Vector Oblivious Linear Evaluation (VOLE), called VOLEitH. The authors of the paper establish a direct connection between MPCitH and VOLEitH and note that their approach is up to two times faster than previous MPCitH protocols.

Through years of development and improvement of MPCitH protocols, there is a very good understanding of the approach and the associated limitations. Since VOLEitH is still very new, such elaboration is missing and previous optimisations focus primarily on improvements to individual building blocks and cryptographic primitives, but do not attempt to understand or improve the structure of VOLEitH. Therefore, this work aims to explain the VOLEitH approach and classify it into existing post-quantum signatures based on ZKPs. We will look in particular at the structure of the individual rounds of the protocol, highlight the countermeasures for detecting cheating provers and present a comparison of MPCitH and VOLEitH. This comparison serves to build a good understanding of post-quantum signatures and highlight bottlenecks. We then use this understanding to present theoretical improvements to digital signatures. We describe the ideas and their effects on communication overhead, computational complexity and security.

## 1.1 Related Work

In this part, we want to take a closer look at related work and highlight the differences in this elaboration.

**Improved all but one Vector Commitment.** In 2024, [BCdSG24] published an improvement on all but one Vector Commitment, which does not use the Random Oracle assumption, but a random permutation oracle. Such an oracle can be instanced using fixed key AES and thus has hardware support. This leads to smaller and faster signing and verification times. In addition to that, a more efficient way to create all-but-one Vector Commitments called the half-tree technique [GYW+23], is used and leads to better efficiency.
In this work, we are not concerned with improving individual building blocks, such as Vector Commitments, within the protocols, but want to consider whether the basic structure of MPCitH and VOLEitH can be adapted to achieve improvements for signatures.

**FAESTer.** Building on the VOLEitH signature FAEST, which is presented in Section 4.2.2, [BBM+24] showed how to improve the performance by applying a new Vector Commitment scheme, called batch all but one Vector Commitments (BAVC). In addition, a new strategy is presented to reduce the entropy in the proof and thereby improve the signature size known as "grinding" [Sta21]. Using such BAVCs and grinding in the FAEST mechanism results in a more efficient protocol, called FAESTer. Furthermore, the authors consider other OWFs, in addition to AES, namely Rain and Multivariate Quadratic (MQ), which also use the new improvement of BAVC.
The goal of FAESTer is to improve FAEST and apply other OWFs, besides AES. The authors also consider how these changes affect complexity and thus ensure optimisations in digital signatures, which is also the goal of this work. We, on the other hand, want to look at the general VOLEitH and MPCitH protocols and make changes based on the general structure of both frameworks. FAEST and FAESTer are very optimised protocols and therefore less generalisable.

**VOLEitH Signature from Multivariate Quadratic.** Multivariate quadratic (MQ) is a cryptographic assumption and serves as the basis for a variety of protocols, including signatures. In MQ, a system $\mathcal{F} : \mathbb{F}_p^n \to \mathbb{F}_p^m$ of $m$ quadratic polynomials in $n$ variables over some field $\mathbb{F}_p$ is used. For a uniformly random system $\mathcal{F} : \mathbb{F}_p^n \to \mathbb{F}_p^m$ of $m$ and some $x \in F_p^n$, the MQ problem is to find $x$ given $\mathcal{F}$ and $\mathcal{F}(x)$.
Based on the Rainbow signature scheme [DS05], which is also built on MQ, [Bui24] now shows how to adapt the Quicksilver protocol to use such a system of quadratic polynomials and thus realise the adjustment to the VOLEitH approach. To do that, they first present

the publicly verifiable zero-knowledge protocol based on MQ over any finite field $\mathbb{F}_p$ and describe the transformation, using Fiat-Shamir, to build the final signature. Furthermore, they present an improvement in Vector Commitments.
The authors present an improvement of the VOLEitH protocol using a different assumptions, Multivariate Quadratic. In our work, we do not use any additional cryptographic assumptions but rather try to optimise based on the previous protocols and their cryptographic framework.

**Improvement PERK Signature.** PERK [ABB+23] is a digital signature scheme that builds on the MPCitH paradigm and thus provides security, even against post-quantum attackers. In 2024, the PERK team shared ongoing work to improve the underlying Permuted Kernel Problem (PKP) using the VOLEitH proof system [PER24]. This results in smaller and faster signatures at the same security level.
Unfortunately, this is still an ongoing work and at this point, there are no final results presented, which could be useful for this work. But the basic idea is the same, adjustments to Perk's MPCitH approach to be able to calculate more efficient signatures. In contrast, we look at the [KKW18] protocol, which is also an MPCitH protocol but is structured fundamentally differently.

**Threshold Computation in the Head.** An important part of most of the MPCitH implementations is secret sharing. Improving such sharing results in more efficient protocols. The Threshold Computation in the Head (TCitH) framework [FR23] introduces a new approach called "batch secret sharing" that uses GGM Trees [GGM86], resulting in much lower communication overhead and reduction in computational complexity. The most interesting part of this work is the provided comparison between the new TCitH framework and VOLEitH. The authors show, that VOLEitH can be seen as a specific application of TCitH, which suggests that MPCitH and VOLEitH can be combined or could even replace each other.
This paper comes very close to our work because it tries to fundamentally improve the MPCitH approach by using GGM Trees. The first major component of our work is the description of the VOLEitH approach and the comparison with previous MPCitH approaches. TCitH goes beyond this and shows that certain instances of TCitH, i.e. a MPCitH approach, can also be represented as VOLEitH protocols. Therefore, our work is more focused on comparing the frameworks and only deals with optimisations very superficially, while TCitH provides practical approaches to make MPCitH approaches much more efficient.

## 1.2 Contributions

The results of this work are threefold. First, we describe the idea behind VOLE correlations and provide a detailed description of the VOLEitH approach. We then compare VOLEitH with MPCitH to highlight bottlenecks and possible improvements. Finally, we describe modifications to the [KKW18] protocol, a very efficient MPCitH approach, to apply new techniques from VOLEitH and thus achieve theoretical improvements in digital post-quantum signatures.

**Description of the VOLEitH approach.** Vector Oblivious Linear Evaluation is a special form of Oblivious Transfer (OT). Two parties, called sender and receiver, learn a common VOLE correlation $\boldsymbol{Q} = \boldsymbol{V} + \boldsymbol{U} \cdot \Delta$, where the sender knows $\boldsymbol{U}$ and $\boldsymbol{V}$ and the receiver knows $\boldsymbol{Q}$ and $\Delta$. At the end of the protocol, the receiver learned the VOLE correlation without publishing its own and the sender's secret inputs. To gain an understanding of the use of such correlations in Zero-Knowledge Proofs, we describe in detail the path from OT to SoftSpokenOT, which forms the basis for the subspace VOLEs used in VOLEitH.
The problem with previous VOLE-based ZKPs is the designated verifier scenario. The verifier has to keep a secret state, namely his part of the VOLE correlation, that the prover cannot learn to ensure the soundness of the proof. This means that the proof is not publicly verifiable and therefore cannot be used for signatures. To overcome this problem, [BBdSG+23b] introduced VOLEitH. In this work, we describe the VOLEitH protocol in detail. We will look at the individual rounds and their effects within the proof. We also look at the possibilities of a (malicious) prover and show which countermeasures are available in the protocol. This provides a detailed understanding of VOLEitH and thus helps to improve future research, as previous work has mostly used VOLEitH as a black box.

**Comparison of MPCitH and VOLEitH.** One goal of this work is to improve digital post-quantum signatures. Previous protocols used the MPCitH approach, which enables the creation of signatures on Multi-Party Computation protocols. To have a good starting point for optimisations, we describe the previous MPCitH approach and compare it with VOLEitH. We look in particular at the [KKW18] protocol, which implements the MPCitH approach very efficiently through many optimisations, and FAEST, the first VOLEitH-based signature. We describe the computational complexity and the communication used by the underlying protocols and highlight existing bottlenecks. Furthermore, we show that the basic structure of MPCitH and VOLEitH is similar, but both protocols use fundamentally different approaches to create digital signatures. MPCitH is based on the simulation of an MPC protocol "in the head" of the prover, while VOLEitH realises mathematical correlations through VOLEs.

**Modifications.** In the final part of this work, we use the previous descriptions of the protocols and the highlighted bottlenecks to make improvements to digital signatures. To do this, we provide a high-level description of changes to the protocol we deem necessary as well as how these changes effect the computational complexity, communication and security of the protocols.

## 1.3 Overview

In the following Chapter 2 we deal with the basics required for this work. We will focus on cryptographic primitives and methods for creating digital signatures. Building on this, we describe in Chapter 3 the idea of Vector Oblivious Linear Evaluation and the connection between Oblivious Transfer and VOLE correlations. Chapter 4 explains in detail the Multi-Party Computation in the Head and the VOLE in the Head approach. We describe the idea of both protocols and how efficient signatures can be created based on them. These are then compared with each other in Chapter 5 and bottlenecks are highlighted. We particularly consider the communication effort of both approaches and the computational complexity for the prover and verifier. In Chapter 6, we present a few optimisations that can be used to improve the MPCitH approach, which particularly draws on ideas from VOLEitH. In Chapter 7, we summarise our work and describe some open problems.

# 2 Preliminaries

This chapter introduces some basics that are needed to understand this work in more detail. The focus relies on some notations, cryptographic primitives, like Zero-Knowledge Proofs or One-Way Functions, and basic concepts, such as Multi-Party Computation or the Fiat Shamir Transformation.

## 2.1 Notations

**Vectors and Matrices.** In this work we denote a vector by lowercase bold letters $\boldsymbol{x}$ and its elements by $\boldsymbol{x}_i$. The index of a vector can also be a range $[i, ..., j]$ with $i \leq j$. So with $\boldsymbol{x}_{[i,...,j]}$ we address the elements $\boldsymbol{x}_i$, $\boldsymbol{x}_{i+1}$ until $\boldsymbol{x}_j$. For matrices, bold capital letters are used, so that a $m \times n$ matrix $\boldsymbol{U}$ has $m$ rows and $n$ columns. The element $u_{i,j}$ with $i \leq m$ and $j \leq n$ is the value in the $i$-th row and the $j$-th column of $\boldsymbol{U}$. The $i$-th row and the $j$-th column vector of $\boldsymbol{U}$ will be denoted by $\boldsymbol{u}_i$ and $\boldsymbol{u}^j$ respectively. By using $\boldsymbol{U}_{a,...,b}$, we denote the concatenation of the row vectors $\boldsymbol{u}_i$ for $a \leq i \leq b$. For a given vector $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, diag($\boldsymbol{x}$) denotes the diagonal matrix using $\boldsymbol{x}$ on the diagonal, i.e.

$$\text{diag}(\boldsymbol{x}) = \begin{pmatrix} x_1 & 0 & ... & 0 \\ 0 & x_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & x_n \end{pmatrix}.$$

**Ideal Functionalities.** An Ideal Functionality is a theoretical construct used to define the security properties of an cryptographic protocol [DDM+06]. If the real execution of the protocol is indistinguishable from an Ideal Functionality, the attacker cannot learn anything. In this work, Ideal Functionalities are used to interact with parties and calculate their secret inputs. In contrast to Trusted Third Parties, an Ideal Functionality can also perform further calculations on the secrets, which is especially needed to build correlations between the inputs. We will denote an Ideal Functionality as $\mathcal{F}$. In order to distinguish the individual Functionalities, the respective index indicates the cryptographic primitive to which the functionality relates. The exponent indicates all the parameters that the Ideal Functionality needs for the calculations, such as field size or matrix dimensions.

**Linear Codes.** A $[n_\mathcal{C},k_\mathcal{C},d_\mathcal{C}]_p$ linear Code $\mathcal{C}$ is a $k$-dimensional subspace of $\mathbb{F}_p^{n_\mathcal{C}}$, where $n_\mathcal{C}$ denotes the length and $d_\mathcal{C}$ the minimal distance of the code. The distance between two codewords is calculated by the Hamming Distance. Note that we use the subscript $C$ to show that these elements belong to the linear code $\mathcal{C}$ and thus can be distinguished from other variables and matrices. Each linear code has a generator matrix, $\boldsymbol{G}_\mathcal{C} \in \mathbb{F}_p^{n_\mathcal{C} \times k_\mathcal{C}}$, i.e. its rows are a basis for $\mathcal{C}$ as a linear subspace. We define $\boldsymbol{T}_\mathcal{C} \in \mathbb{F}_p^{n_\mathcal{C} \times n_\mathcal{C}}$ as an extension of $\boldsymbol{G}_\mathcal{C}$, such that the first $k_\mathcal{C}$ rows are from $\boldsymbol{G}_\mathcal{C}$ and the remaining rows are chosen so that $\boldsymbol{T}_\mathcal{C}$ is invertible and forms a basis of $\mathbb{F}_p^{n_\mathcal{C} \times n_\mathcal{C}}$. This will be used later to extract the information embedded in the code. Furthermore, given a matrix $\boldsymbol{A} \in \mathbb{F}_p^{n \times k_\mathcal{C}}$ and an $[n_\mathcal{C},k_\mathcal{C},d_\mathcal{C}]_p$ linear Code $\mathcal{C}$, the embedding of $\boldsymbol{A}$ inside the code $\mathcal{C}$ will be denoted by the $n \times n_\mathcal{C}$ matrix $\mathcal{C}(\boldsymbol{A})$, such that $\mathcal{C}(\boldsymbol{A}) = \boldsymbol{G}_\mathcal{C} \cdot \boldsymbol{A}$.

**Repetition Codes.** Repetition Codes are simple linear error-correcting codes which repeat the message several times and thus realise a correct transmission over a noisy channel. Repetition codes can also be parameterised with $[n, k, d]$, similar to linear codes. For a given repetition parameter $\tau$, each bit in the original message is repeated $\tau$ times. Thus, a repetition code $[\tau, 1, \tau]$ consists of one information bit which is repeated $\tau$ times.

**Negligible probability.** A function *negl* is called *negligible* if it grows slower than the inverse of every polynomial function. More formally, negl($\lambda$) $\leq 1/\lambda^c$ for all $c$ and all sufficient large $\lambda$. Negligible functions are mostly used in security games for cryptographic primitives where they are used to upper bound the advantage of adversaries. Furthermore, if the advantage of a given adversary against some protocol is negligible, this means that the protocol is secure against the adversaries attack strategy.

**Probabilistic Polynomial Time.** An Probabilistic Polynomial Time (PPT) algorithm is an algorithm that runs in polynomial time, w.r.t. the input size, and uses randomness. Such an algorithm can produce different outputs on the same input. In addition, the error for an incorrect result is bounded. In this work, PPT algorithms are needed to model participants in protocols, such as sender and receiver, and to be able to show the security.

## 2.2 Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKP) are two-party protocols between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The idea is quite simple: For a given language $\mathscr{L}$, a public input $x$ and an NP-relation $\mathcal{R}$, the prover wants to convince the verifier that he has a witness $w$ that proves that $x \in \mathscr{L}$ with $(x, w) \in \mathcal{R}$, without revealing $w$ to the verifier. The concept of ZKP was first intro-

duced by Goldwasser, Micali, and Rackoff in 1985 and is nowadays a fundamental building block in cryptography [GMR85]. For such protocols, we consider three properties:

- **Correctness** - if $\mathcal{P}$ really knows a witness $w$, then $\mathcal{V}$ should always be convinced.
- **Soundness** - if a malicious prover $\mathcal{P}^*$ does not know a witness $w$, then $\mathcal{V}$ accepts only with a small probability.
- **Zero-Knowledge** - if a malicious verifier $\mathcal{V}^*$ tries to learn anything from the interaction with $\mathcal{P}$, he is not able to extract additional information.

Notice, that the soundness error only has to be small, but not necessarily negligible. To achieve a negligible soundness error, one can just run the protocol multiple times and thus reduce the error. To prove that a system ensures zero knowledge, a simulator *Sim*, who does not know the real witness $w$, can be built. The idea is that the output of *Sim* is indistinguishable from the distribution of the transcript of the protocol run by $\mathcal{P}$ and $\mathcal{V}^*$. In other words, if the attacker can not distinguish whether he is interacting with the simulator *Sim* or the real protocol, he can not learn anything from the protocol. The transcript denotes the interaction between the prover and the verifier.

## 2.3 Hash Functions

A hash function $H$ describes a function that compresses a string of arbitrary input length to a string of fixed length, i.e. $H$: $\{0,1\}^* \rightarrow \{0,1\}^n$. A set of hash functions is also called a family of hash functions. To realise secure protocols and to apply such functions to concrete cryptography, cryptographic hash functions with the following properties are necessary [SG12]:

- **Collision resistance** means that finding any two inputs that generate the same output is hard.
- **One-way-ness**, also called Pre-Image resistance, denotes that knowing $y$, finding $x$ such that $H(x) = y$ is hard.
- **Second Pre-Image resistance**, also called weak collision resistance, signifies that knowing a pair $x$ and $y$, with $H(x) = y$, finding $x'$ such that $x \neq x'$ and $H(x') = y$ is hard.

In this work, Collision Resistant Hash Functions (CRHF) are needed to understand the construction of Vector Commitments by [BBdSG+23b]. Such hash functions fulfil the collision resistance property.

Another special form of non-cryptographic hash functions are universal hash functions [CW77]. A family of hash functions is universal if, for a hash function $H$, which was picked at random from the family, the probability that two distinct keys refer to the same hash value is bounded. More mathematically, for a given family of hash-functions $\mathcal{H}$ with $H \in \mathcal{H} : U \to R$ and all $x, x'$ such that $x \neq x'$, the probability Pr[$H(x) = H(x')$] is bounded by $1/|R|$ [BKST15]. Furthermore, if the probability is bounded by some $\epsilon$, i.e. $1/|R| \leq \epsilon < 1$, then $H$ is an $\epsilon$-almost-universal family of hash functions.

## 2.4 Security Models

To prove the security of protocols, the Common-Reference-String (CRS) and the Random Oracle Model (ROM) are often used. In the CRS model, a random string is chosen and shared with all participants in the protocol. All the parties can use this string inside the protocol. Thus, the CRS model facilitates protocols where parties can prove knowledge of certain information without revealing the information itself.
The Random Oracle Model is a theoretical framework and assumes the existence of an idealised function that responds to every unique query truly random. This allows proving the security of a protocol under ideal conditions [Ble11]. In reality, hash functions are used to replace the ROM. In this case, the hash functions respond to the requests to the ROM by producing and returning completely random hashes. Due to the use of cryptographic hash functions, this output cannot be distinguished from true randomness.
The CRS+RO model now combines both approaches to be able to make useful security assumptions and provide security evidence. For this, it is assumed that all parties have access to a CRS and a random oracle.
To build Zero Knowledge Proofs in the combined CRS+RO model, Baum et al. present a two-phase protocol consisting of a *setup* phase and the proof itself concerning a Random Oracle $H$ [BBdSG+23b]. An interactive Zero-Knowledge Proof system $\Pi$ for NP relation $\mathcal{R}$ is a tuple $\Pi$ = (Setup$_H$, $\mathcal{P}_H$, $\mathcal{V}_H$) of PPT algorithms:

- **Setup**$_H$ gets the security parameter $\lambda$ as an input and returns a common reference string (crs).
- $\boldsymbol{\mathcal{P}}_H$ now interacts with $\boldsymbol{\mathcal{V}}_H$ and both parties receive the crs and a common input $x$. The private input of $\mathcal{P}_H$, the witness $w$, is chosen such that ($x$, $w$) $\in \mathcal{R}$. The verifier outputs a bit $b$ = 1, if he accepts, or $b$ = 0, if he rejects.

Later on, it will be assumed that the prover and the verifier got access to the common reference string and the random oracle $H$ and it is not explicitly mentioned.

## 2.5 Commitment Schemes

A Commitment Scheme (CS) is a cryptographic primitive and was first introduced by Brassard et al. in 1988 [BCC88]. The idea is to allow one to commit to a specific value and send it to other parties. These parties are unable to extract the original value until the sender of the commitment opens it, this is called the *hiding* property. Additionally, a CS satisfies the *binding* property, which means that after sending the commitment to the other parties, one can not change the value which was committed. Such protocols can be formalised with two phases, the commit phase and the reveal phase.
A simple example is a coin-toss scheme, where Alice flips a coin and Bob wants to guess the result. Bob sends his guess $b$ hashed with a cryptographic hash function $H$ and a random string $r$, i.e. $H(b \mid\mid r)$, to Alice. Alice now flips the coin and sends the result to Bob. Bob now "opens" his commitment by sending $H$ and $r$ to Alice, who now sees the guess $b$. The Commitment Scheme is hiding, since a cryptographic hash function is used. This hash function fulfils the pre-image resistance property and thus is hiding the input. The commitment is binding, since such hash functions are collision resilient, so there is just one possible hash for each input. Commitment Schemes can be used in many applications, for example, Zero-Knowledge Proofs [GMW91] or signature schemes [Lam79].

### 2.5.1 Vector Commitment Schemes

Vector Commitments (VC) are Commitment Schemes on vectors instead of single values. They can be formalised as a two-phase protocol between two PPT machines, called sender and receiver. In the commitment phase, the sender commits to a vector of messages while keeping them secret. In the revealing (decommitment) phase, the receiver opens a subset of indices of the commitment. Such Vector Commitments also satisfy the binding and hiding property but with an extended hiding property, since the VC has to ensure that unopened indices are hidden, even after opening a subset of indices.
Baum et al. define their Vector Commitment with respect to the Common Reference String and Random Oracle Model [BBdSG+23b]. For a given Random Oracle $H$ and a message space $\mathcal{M}$, a (non interactive) Vector Commitment can be described by the following PPT algorithms:

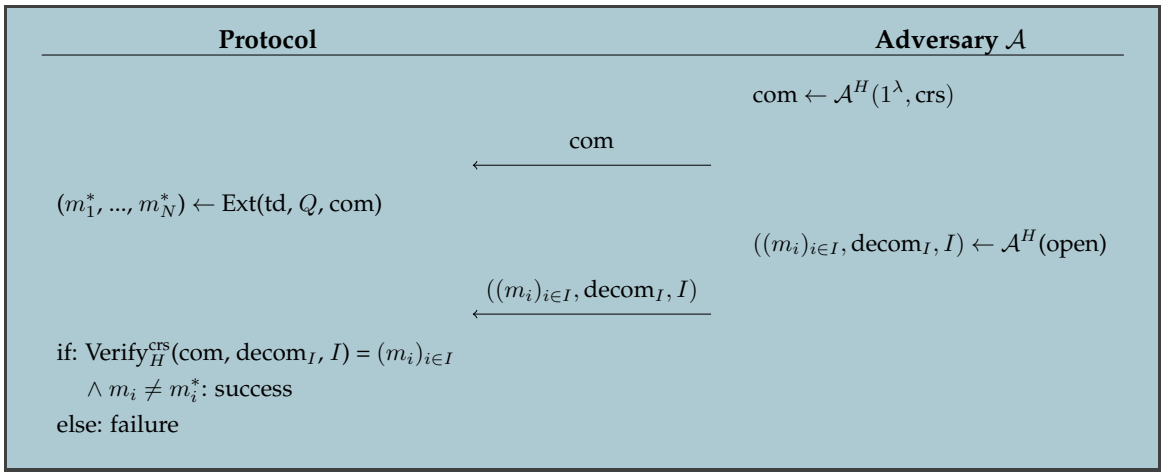
- **Setup**$_H$ receives a security parameter $\lambda$ and a vector length $N$ = poly($\lambda$) as input. It outputs a common reference string, which can be used by the sender and receiver.
- **Commit**$_{H,\text{crs}}$ gets the crs as input and outputs a commitment *com* and its opening information *decom* for messages $(m_1, ..., m_N) \in \mathcal{M}^N$.

- **Open**$_{H,\text{crs}}$ receives the indices $I \subseteq [N]$ of commitments the receiver wants to open, the crs and the opening information provided by **Commit**$_{H,\text{crs}}$. It outputs the openings $decom_I$.
- **Verify**$_{H,\text{crs}}$ finally checks if the openings $decom_I$ match the commitments provided by *com*. For this it gets access to the crs, *com*, $decom_I$ and $I$ itself. **Verify**$_{H,\text{crs}}$ either output the messages $(m_i)_{i\in I}$ if the opening was correct or $\perp$ if it was not.

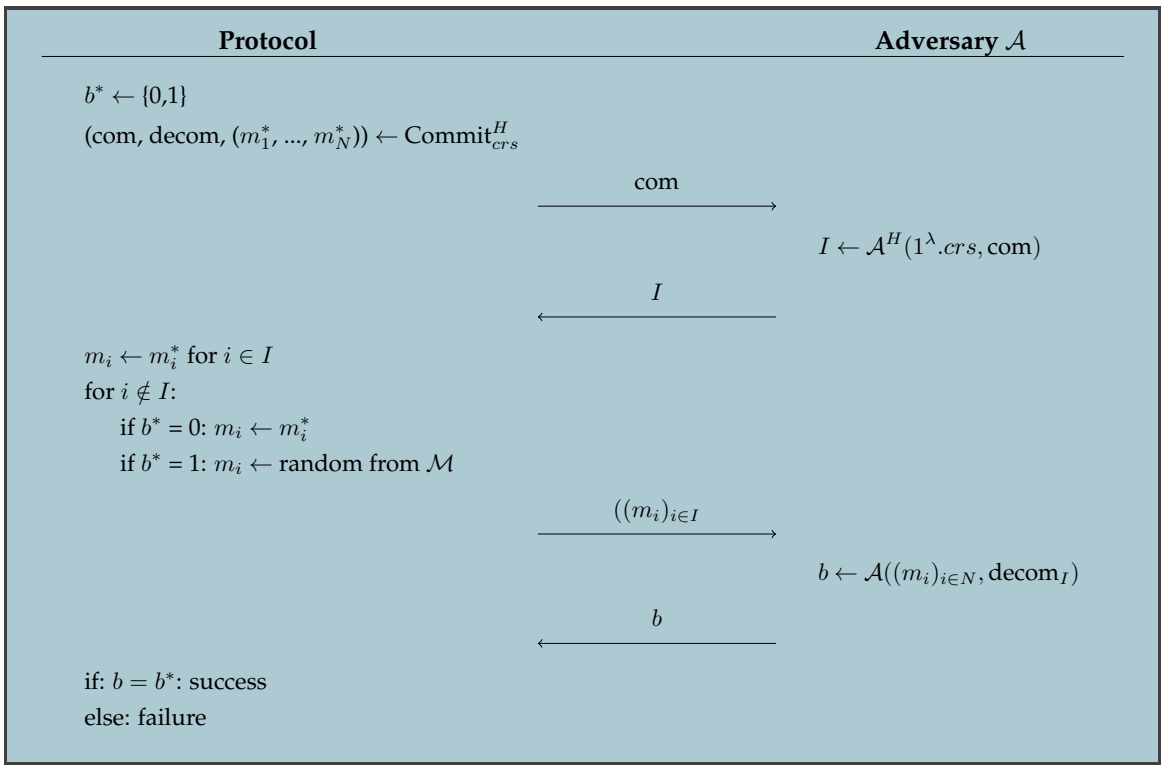
Such VCS can be constructed and are secure in the CRS+RO model. The Vector Commitment is (perfectly) correct if for all random oracles $H$, all security parameters $\lambda \in \mathbb{N}$ and for all $N$ = poly($\lambda$) the verify algorithm outputs the messages $(m_i)_{i\in I}$ for $I \in [N]$ if *com* received by *Commit* and $decom_I$ obtained by *Open* are consistent. To show that such a construction is binding, Baum et al. present a security game where the adversary chooses the commitments and the protocol gets an extract functionality for a given trapdoor function and the queries that were made by the adversary. The attacker is successful if he sends correct decommitments that pass the *Verify* algorithm, but that are not equal to the extracted messages by the protocol. In other words, the attacker wins if he can produce a commitment which has more than one decommitment. Such constructions are hiding if the attacker loses the real or random game for this VC. For this, the protocol chooses random commitments *com* for messages ($m_1^*$, ..., $m_N^*$). The attacker receives the crs and the commitment. Now he chooses $I \subseteq [N]$ and sends it to the protocol. The protocol sets all $m_i$ for i $\in I$ to the real value $m_i^*$. If the choice bit $b$ = 0, all other values $m_i$, i $\notin I$, also get the real value $m_i^*$, but if If $b$ = 1 all other values $m_i$, i $\notin I$, are random elements from $\mathcal{M}$. The attacker now gets all messages $(m_i)_{i\in[N]}$ and decom$_I$. The attacker wins if his guess $b^* = b$, otherwise, he loses. Both security games are depicted in Figure 2.1. Notice, that the protocol is hiding and binding if the advantage of the attacker is negligible. Such Vector Commitments are later used in the VOLE in the Head construction in Section 4.2.

### 2.5.2 Tree-PRG Vector Commitments

Tree-PRG Vector Commitments can be used to realise all but one opening. That means, for a commitment of length $N$, exactly $N-1$ indices can be selected to be opened by the sender without losing security. To realise such protocols, [BdSGK$^+$21] uses a so-called GGM Tree of length-doubling PRGs [GGM86]. The idea behind the GGM construction is, that there is a PRG which takes a binary input of length $n$ and outputs a binary string of length $2n$, which can be split up into two new seeds of length $n$. Repeating these algorithms allows one to create many seeds based on one string of length $n$. More technically, a given PRG: {0,1}$^n$ $\rightarrow$ {0,1}$^{2n}$ takes a seed $k$ and outputs PRG($k$) = PRG$_0(k)$ || PRG$_1(k)$, where PRG$_0(k)$ denotes the first $n$ bits and PRG$_1(k)$ the second $n$ bits of PRG($k$). This can be used




(a) (Extractable) Binding security game for a given trapdoor function *td* and its extractable function *Ext* using the set of queries $Q$, which were placed by $\mathcal{A}$ to $H$. At the beginning, there is a setup phase which creates the crs and the *td*, i.e. (crs, td) $\leftarrow$ TSetup$^H(1^\lambda, N)$.




(b) Hiding security game for a given set of message $\mathcal{M}$. At the beginning, there is a setup phase which creates the crs, i.e. crs $\leftarrow$ Setup$^H(1^\lambda, N)$

Figure 2.1: Depiction of the (extractable) binding and hiding security game for Vector Commitments used in [BBdSG+23b]. In both cases, $N$ denotes the amount of messages that are committed and $N$ = poly($\lambda$).

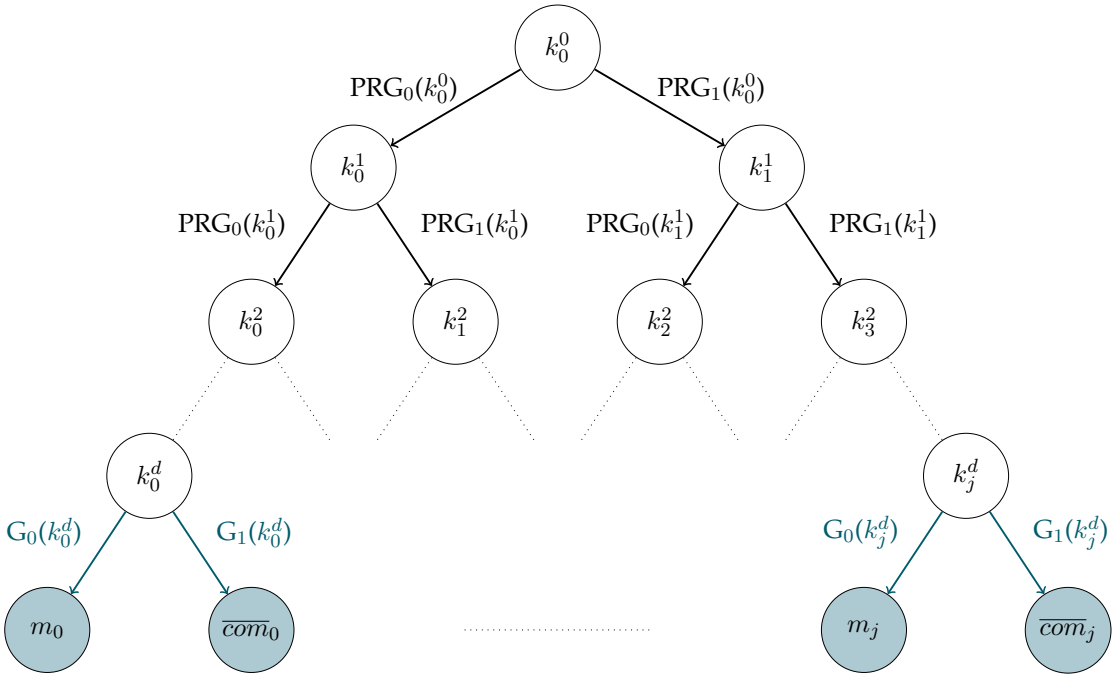



Figure 2.2: The GGM construction for a given seed $k \in \{0,1\}^n$ a PRG: {0,1}$^n \to$ {0,1}$^{2n}$ and a collision-resistant hash function G: $\{0,1\}^\lambda \to \{0,1\}^\lambda \times \{0,1\}^{2\lambda}$. The $j$-th element at depth $d$ is denoted by $k_j^d$. The GGM extension for Vector Commitments is highlighted in colour.

to build Vector Commitments as described in [BBdSG+23a], by extending the tree leafs $k_j^d$ with a collision-resistant hash function $G$ into two values, the message $m_j$ that is used for the commitment and the commitment $\overline{com}_j$. A visualisation of the GGM construction and this extension is shown in Figure 2.2.

To publish a commitment it suffices to publish the hash of $\overline{com}_0$, $\overline{com}_1$, ..., $\overline{com}_{n-1}$. The tree keys $k_j^i$ and all $\overline{com}_j$ are stored as decommitment information. To open $N-1$ commitments, *open* generates a partial decommitment consisting of all siblings of GGM keys on the path between $k_0^0$ and $k_{j^*}^d$, with $j^*$ denoting the message that is not opened. Notice, that all messages can be reconstructed except the seed for message $j^*$, i.e. $sd_{j^*}$. Furthermore, the partial decommitment contains $\overline{com}_{j^*}$ which can later be used to verify that all elements between $k_0^0$ and $k_{j^*}^d$ are well formed. To now verify the openings, an algorithm *reconstruct* is used. *Reconstruct* takes the message $j^*$ and the partial decommitment received by *open*. It now reconstructs all tree leafs $k_j^d$ for $j \in [0, ..., 2^d)$ except for $j^*$, which were stored in the commitment phase, recomputes all $\overline{com}_j$ and hashes them. The output then contains all hashes and the leaf keys. Note that verify only has to call *reconstruct* and compare the hash received by *reconstruct* with the commitment received before.

## 2.6 One-Way Functions

In today's cryptography, the existence of One-Way Functions (OWF) is a fundamental assumption, as they enable the construction of various cryptographic primitives, such as Signatures, Symmetric Encryption, or Pseudo-Random Generators. The basic idea is that for a given One-Way Function $f$ and a value $x$, it is easy to compute $f(x) = y$ but the other way around is hard, so it is hard to find a $x'$ such that $f(x') = y$. *Easy* to compute describes in this context that there is a polynomial-time algorithm that outputs $f(x)$ for a given input $x$. In contrast, *hard* means that the probability that for a given function $f$ and its result $y$ finding an input $x'$ such that $f(x') = y$ is negligible [Gol01].
OWFs can be used to build efficient zero-knowledge protocols. The prover wants to convince the verifier, that he knows the value $x$, such that $f(x) = y$. Based on the above properties, the verifier can easily validate that this is a valid instance. Finding efficient and secure OWFs is a way to optimise Zero-Knowledge Proofs.

## 2.7 Multi Party Computation

Secure Multi-Party Computation (MPC) describes a subfield in cryptography, which deals with the computation of functions based on secret inputs held by different parties, without revealing their inputs themselves. The concept of MPC was introduced by Goldwasser, Ben-Or and Widgerson in 1988 [BGW88] and independently Chaum, Crépeau and Damgard presented their idea of secure computation [CCD88].

**General Structure.** A simple example for Multi-Party Computation is the millionaires problem: Three millionaires want to compute the sum of their total assets without revealing their assets. To achieve this, the first millionaire adds a random value to his wealth and sends it to the next one who adds his value. After sending it to the third who does the same, the first millionaire receives the result, subtracts his random value and publishes the final result. More generally speaking, in a Multi-Party Computation protocol there are $N$ parties $P_1$, ..., $P_N$ with individual secrets $s_i$ that each party wants to keep private. All parties got access to a shared functionality $f$, which takes $N$ inputs and returns a single output to all participating parties. The parties now want to compute $y = f(s_1, ..., s_N)$ without revealing $s_i$, besides what can be learned from $y$.
To realise meaningful Multi-Party Computation protocols, *correctness* and *input privacy* is needed. To define those properties, we assume there is an adversary $\mathcal{A}$, who controls a subset of parties. In the first step, these parties participate semi-honestly (honest but curious) in the protocol, which means that they follow the protocol but try to obtain additional

information about the other parties' secrets. So for a given Multi-Party Computation protocol:

- **Correctness** ensures that any subset of controlled parties by $\mathcal{A}$ should not be able to force honest parties to output an incorrect result.
- **Input privacy** formalises that no information about the secrets held by the honest parties can be inferred by the execution of the protocol.

This can be extended, such that the parties also can act maliciously, so they can arbitrarily deviate from the protocol. Correctness can be achieved in two ways, either the protocol can guarantee that the honest parties always output the correct result or there is a way to observe if something went wrong. Then the honest parties could abort the current run and restart the protocol.
Notice, that input privacy just ensures that the adversary can not obtain information by the execution of the protocol, which means communication between the parties, but the attacker could get additional information by the result. Since all parties got access to the result, this is not an advantage for the attacker to disturb the privacy of the input. We call a protocol $t$-private if it preserves input privacy against $t$ adversarial colluding parties, $t < N$. Back to the millionaire problem, this protocol ensures $t = 1$ privacy, since one adversarial party can not find out the assets of the other two parties caused by the random value. Such privacy proofs can be formalised using a simulator, which simulates the honest parties without knowing their input. The simulator gets the final result of the function $f$ and the input of the adversarial parties. Then he carries out the protocol: if the adversary is not able to distinguish between the real protocol execution and the simulator execution, he can not learn anything about the input of the honest parties. In other words, an MPC protocol ensures input privacy, if an adversary $\mathcal{A}$ can not distinguish between the real and ideal world [BPW04].

**MPC with Circuits.** Most of today's Multi-Party Computation protocols consider the function $f$ as an arithmetic circuit operating over some finite field $\mathbb{F}$, using only addition ($\oplus$) and multiplication ($\odot$) gadgets. For each gadget, there is a unary and binary version, the unary one takes one input by a party and a fixed constant $\alpha$ as the second input. In the binary setting, two parties want to calculate a function based on their inputs. Each party $P_1$, ..., $P_N$ now gets a copy of the circuit. The problem is, that the party $P_i$ just know its input $x_i$, but not the other inputs. To overcome this problem, $P_i$ has to share its input $x_i$ into $N$ values denoted by $x_i^{(1)}$, ..., $x_i^{(N)}$. To ensure the privacy of $x_i$ itself, it has to be ensured that $N - 1$ shares do not reveal any information about $x_i$, which can be realised through a *additive* or *polynomial* sharing. To share a value in the additive sharing,

the elements $x_i^{(1)}$, ..., $x_i^{(N-1)}$ are drawn completely random from $\mathbb{F}$. The final share $x_i^{(N)}$ is chosen, such that the addition of all shares $x_i^{(j)}$ equals $x_i$. In the polynomial sharing, coefficients $a_1$ to $a_n$ are chosen completely random from $\mathbb{F}$ and the coefficient $a_0$ contains the secret $x_i$. Based on these coefficients, the following polynomial can be constructed:

$$p(y) = \sum_{i=0}^{n} a_i = x_i + a_1 y + a_2 y^2 + ... + a_n y^n$$

A sharing for party $j$ can then be computed by evaluating the polynomial on input $j$, i. e. $p(j)$. Later on, polynomial interpolation can be used to reconstruct $p(0) = x_i$. One example of interpolating polynomials is the Lagrange interpolation.
The idea for realising MPC protocols can be summarised as follows: Each party shares its secret $x_i$ with additive or polynomial sharing and sends the jth share $x_i^{(j)}$ to party j. After that, all parties perform a gate-wise computation on their local copy of the circuit. Each party now holds a valid sharing of the result of $f$, so all parties communicate their result-sharings and can reconstruct the final output.

**BGW.** The BGW protocol, presented by Ben-Or, Goldwasser and Widgerson, uses this approach based on a polynomial sharing for t $\leq \lfloor (N-1/2) \rfloor$ [BGW88]. Each party computes a sharing of its secret $x_i$ of degree $t$, denoted as $[\![x]\!]_t$, and sends $x_i^{(j)}$ to party $j$. Now it just needs to be specified, how the different gadgets are realised:

- The **unary addition** gadget takes a sharing $x^{(i)}$ and a constant $\alpha$ as input and returns $z^{(i)} = x^{(i)} \oplus \alpha$ for the values $x^{(1)}$, ..., $x^{(n)}$ that are hold by the parties.
- The **unary multiplication** gadget takes a sharing $x^{(i)}$ and a constant $\alpha$ as input and returns $z^{(i)} = x^{(i)} \odot \alpha$ for the values $x^{(1)}$, ..., $x^{(n)}$ that are hold by the parties.
- The **binary addition** gadget takes a sharing $x^{(i)}$ and a sharing $y^{(i)}$ from another party as input and returns $z^{(i)} = x^{(i)} \oplus y^{(i)}$ for the values $x^{(1)}$, ..., $x^{(n)}$ and $y^{(1)}$, ..., $y^{(n)}$ that are hold by the parties.
- The **binary multiplication** gadget takes a sharing $x^{(i)}$ and a sharing $y^{(i)}$ from another party as input and returns $z^{(i)} = x^{(i)} \odot y^{(i)}$ for the values $x^{(1)}$, ..., $x^{(n)}$ and $y^{(1)}$, ..., $y^{(n)}$ that are hold by the parties.

For the first three gadgets it is easily to see that they produce the correct output, since $[\![z^{(i)}]\!]_t = [\![x^{(i)} \oplus \alpha]\!]_t$ respectively $[\![z^{(i)}]\!]_t = [\![x^{(i)} \odot \alpha]\!]_t$ and $[\![z^{(i)}]\!]_t = [\![x^{(i)} \oplus y^{(i)}]\!]_t$ holds. For the binary multiplication, this is not trivially the case, since a multiplication of two polynomials of degree $t$ results in a polynomial of degree $2t$. If these polynomials are then used in

further binary multiplications, they can also achieve much higher degrees. If the degree of the polynomial is greater or equal then $N$, the polynomial sharing can not be interpolated and we lose information meaning the results can not be retrieved from the shares. The rough idea to solve this is that each party creates a sharing of its result share $z^{(i)}$ of degree $t$. So, each party $i$ now has a new sharing $[\![z^{(i)}]\!]_t = (z^{(i,1)}, z^{(i,2)}, ..., z^{(i,n)})$ and sends $z^{(i,j)}$ to party j. They now compute:

$$\zeta^{(i)} = \sum_{j=1}^{n} \lambda_j z^{(j,i)},$$

where $\lambda_j$ are just the elements of the first row of the inverse Vandermonde Matrix. The elements ($\zeta^{(1)}$, ..., $\zeta^{(n)}$) now form a valid sharing of degree $t$ and $[\![\zeta]\!]_t$ = $[\![x \cdot y]\!]_t$
This construction can be proven to ensure $t$-privacy by building a simulator *Sim*. The rough idea of the simulator is the following: *Sim* generates random values to be the secrets of the honest parties that he simulates. Then the simulator follows the protocol until the shares of the result $y$ have to be exchanged. The simulator then gets access to the real result $y$ and the shares held by the corrupted parties. *Sim* builds a valid sharing that generates $y$ and contains the shares of the adversary. The simulator then broadcasts the final shares he has chosen for the honest parties.
An adversary $\mathcal{A}$ can not distinguish between the real execution of the protocol and the execution simulated by *Sim*, since he only controls $t$ parties and the secrets $s_i$ are shared with polynomials of degree $t$. As $t + 1$ shares are necessary to reconstruct a polynomial of degree $t$, such an attacker is not able to reconstruct the secrets based on the shares and can not distinguish between the real secrets and the random values. Even inside the protocol, the attacker can learn nothing by the communication with the parties, based on the fact that the unary gadgets respectively the binary addition can be computed locally and the binary multiplication uses a fresh sharing. So until the final broadcast step, the attacker learns nothing. In the final step, it just has to be ensured that *Sim* constructs a valid sharing of $y$. This is possible since the simulator has access to $t$ shares held by the adversary and the real result $y$. Based on this he can create a valid sharing of $y$ with degree $t$.
Notice, that throughout the above described protocol execution the $t$ parties controlled by $\mathcal{A}$ are semi-honest. It can be shown that this construction can be adapted to fit in the fully malicious scenario, where those parties can deviate arbitrarily from the protocol. This is not considered in this work as the semi-honest construction suffices for our use-case. We will use BGW later on to show, how Zero-Knowledge Proofs can be built using MPC. This is described in more detail in Section 4.1.

## 2.8 Signature Schemes

A signature scheme can be used to validate the sender of a given message. To realise that, most of today's signatures work with asymmetric cryptography, meaning the sender owns a key pair consisting of a public and a private key. He now uses the private key to produce a signature for a given message and sends it to the receiver, who can then use the public key to verify the signature. Such schemes can be described by the following three algorithms:

- **Key generation.** This algorithm outputs a public and private key pair. To do this, a private key is selected at random from a set of possible private keys and outputs the key itself and the corresponding public key.
- **Signing.** *Sign* takes the message $m$, which has to be authenticated, and the secret key and outputs the signature for $m$.
- **Verification.** *Verify* receives the signature, the public key and the message that produced the signature, and accepts or rejects the authentication.

The concept of signature schemes was originally introduced by Diffie and Hellman in 1976 [DH76]. Since then a lot more signatures have been invented, like RSA [RSA78], Lamport signatures [Lam79] or Merkle signatures [Mer79]. In this work, the focus relies on understanding how to build signatures based on MPCitH and VOLEitH and how to improve signature schemes.

## 2.9 Fiat-Shamir Transformation

In 1986, Fiat and Shamir introduced the idea of Random Oracle Models and showed how to make interactive protocols non-interactive using such a ROM [FS86]. This approach is called Fiat-Shamir Transformation. For the transformation to be applied, the protocol must satisfy the public coin property, meaning that one party only sends random coins. In the context of Zero-Knowledge Proofs, public coin means that the verifier only sends random values to the prover. These values are independent of each other and the prover's messages. In practice, the Fiat-Shamir Transformation is used to build signature schemes based on interactive, public coin zero-knowledge protocols and cryptographic hash functions. This is discussed in more detail in Section 4.1.2.

# 3 From OT to VOLE

In this chapter, we briefly introduce the idea of Oblivious Transfer and then show how VOLE correlations can be constructed from it. We also present the general structure of VOLEs, as they will later be used in VOLEitH.

## 3.1 Oblivious Transfer

The concept of Oblivious Transfer (OT) was first presented by Michael O. Rabin in 1981 and is based on the RSA cryptosystem [Rab05]. Today's usage of OTs is attributed to Even et al., who introduced the first 1-2 Oblivious Transfer which can be used in MPC protocols [EGL85]. The idea is that there is a sender who holds two messages, $m_0$ and $m_1$. The receiver now can sample a bit $b \in \{0, 1\}$ and based on his bit he chooses the message $m_b$. At the end of the protocol, the receiver should have learned $m_b$ and nothing else. The sender is supposed to learn nothing. This can be generalised, such that both parties execute the transfer of a subset of $k$ messages out of a total set of $n$ messages. We will denote these protocols as $k$-$n$ or $\binom{n}{k}$ OT. The most common Oblivious Transfer protocols can be summarised in three subgroups, 1-2 OT respectively 1-$n$ OT and $k$-$n$ OT, with $k, n \in \mathbb{N}$ and $k < n$. To define useful and secure OT protocols, the following two properties are needed:

- **Sender security** formally means that the receiver will only learn the content of the messages he has chosen from the sender.
- **Receiver security** ensures that the sender does not learn which messages were chosen by the receiver.

Notice that OT protocols can be described as two-party protocols between a sender and a receiver, similar to Zero-Knowledge Proofs. Furthermore, OTs can be used to realise cryptographic tasks, such as secure two-party computation or secure identification [Kil88].

### 3.1.1 All-but-one Oblivious Transfer

The all-but-one Oblivious Transfer scheme is a special form of OT where the receiver learns all messages from the sender, except one. So 1-2-OT and $k$-$n$-OT with $k = n - 1$ can be seen as all-but-one-OTs. The huge advantage of such protocols is that the receiver

can query all messages at once and does not have to invoke multiple Oblivious Transfers. The communication overhead can thus be reduced. This can be later used to realise efficient Zero-Knowledge Proofs based on Oblivious Transfer.

### 3.1.2 Extending Oblivious Transfer

The goal of extending OT is to generate a large number of Oblivious Transfers based on a small number of base Oblivious Transfers. In 1996 Beaver presented an approach to extend $k$ OT's up to $O(k^c)$ OT's based on One-Way Functions and for a given random $c > 0$ [Bea96]. Since the effectiveness and security are based on the underlying One-Way Function, Ishai et al. brought up an approach to extend OTs in the Random Oracle Model (ROM) using a pseudorandom generator (PRG) [IKNP03]. In the work of Ishai et al., extending OT's mean to reduce $\text{OT}_\ell^m$ to $\text{OT}_k^k$ where $m > k$ and $k$ is the security parameter. This means that $m$ Oblivious Transfers of $\ell$ bit strings can be implemented by $k$ Oblivious Transfers of $k$-bit strings. The authors focus on an $\text{OT}_m^k$ approach, which is an $\text{OT}_k^k$ system with some small additional cost. In the literature, this work is called IKNP, based on the authors of the paper.
In the protocol itself, the sender ($\mathcal{S}$) inputs $m$ pairs ($\boldsymbol{x}_{i,0}$, $\boldsymbol{x}_{i,1}$) and pick a random vector $\boldsymbol{s} \in \{0,1\}^k$. The receiver ($\mathcal{R}$) inputs $m$ selection bits $\boldsymbol{r}$ = ($r_1$, ..., $r_m$) and initialises a random $m \times k$ bit matrix $\boldsymbol{T}$. Both parties invoke the $\text{OT}_m^k$ primitive. In this case $\mathcal{S}$ will be the receiver, with input $\boldsymbol{s}$, and $\mathcal{R}$ will be the sender, with input ($\boldsymbol{t}^j$, $\boldsymbol{r} \oplus \boldsymbol{t}^j$). This will be the base OT' for the following protocol. $\mathcal{S}$ now computes $\boldsymbol{Q}$, such that the $j$-th column is $q^j$ = $(\boldsymbol{s}_j \cdot \boldsymbol{r}) \oplus \boldsymbol{t}^j$ and the $i$-th row is $\boldsymbol{q}_i$ = $(\boldsymbol{r}_i \cdot \boldsymbol{s}) \oplus \boldsymbol{t}_j$. Note, that $\boldsymbol{Q}$ is also a $m \times k$ bit matrix. $\mathcal{S}$ sends the pair $(\boldsymbol{y}_{i,0}, \boldsymbol{y}_{i,1})$, where $\boldsymbol{y}_{i,0}$ = $\boldsymbol{x}_{i,0} \oplus H(i, \boldsymbol{q}_i)$ and $\boldsymbol{y}_{i,1}$ = $\boldsymbol{x}_{i,1} \oplus H(i, \boldsymbol{q}_i)$, for 1 < i < m for a given random Oracle $H$: $[m] \times \{0,1\}^k \rightarrow \{0,1\}^l$. Finally, $\mathcal{R}$ outputs $\boldsymbol{z}_i = \boldsymbol{y}_i^{r_i} \oplus H(i, \boldsymbol{t}_i) = \boldsymbol{x}_{i,\boldsymbol{r}_i}$.
It can be shown that this construction is secure against a malicious sender and a semi-honest receiver [IKNP03]. The authors also describe an extension of the protocol, which also ensures security against a malicious sender and a malicious receiver. This is not mentioned in this work, since we want to focus on a generalisation of the protocol called Soft-SpokenOT [Roy22] which uses a modified form of this countermeasure. This approach will be presented in Section 3.2.1.

## 3.2 Vector Oblivious Linear Evaluation

Oblivious Linear Evaluation (OLE) is a special case of OT and describes a two-party protocol between a sender, who holds a pair of field elements, and a receiver, who learns a secret linear combination of these elements [BCGI18]. OLE can be used as a common

building block in a secure computation setting of arithmetic circuits.
An extension of OLE is Vector Oblivious Linear Evaluation (VOLE), where the sender holds a pair of vectors instead of field elements. The motivation for using VOLE instead of OLE is to replace a high number of OLEs with a few, large VOLEs [ADI+17]. The VOLE functionality itself takes a pair of vectors $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}^n \times \mathbb{F}^n$ from the sender, a scalar $x \in \mathbb{F}$ from the receiver and outputs $\boldsymbol{w} = \boldsymbol{u} \cdot x + \boldsymbol{v}$ to the receiver. This functionality can also be randomised by choosing the pair $(\boldsymbol{u}, \boldsymbol{v})$ at random.
The following subsections deal with the question of how to construct generalised subspace VOLE from OT's. Such VOLEs are finally needed in the VOLE in the Head protocol, which will be an essential part of this work.

### 3.2.1 SoftSpokenOT

To build protocols using a high amount of OTs in a secure, but also practical manner, the efficiency of extending OTs need to be increased. Based on the work of Ishai et al., shown in Section 3.1.2, Boyle et al. presented Silent-OT using the learning parity with noise (LPN) assumption and improves extending OT's, but with much stronger assumptions [BCG+19]. SoftSpokenOT, presented by Roy in 2022, instead is a generalisation of the previous protocol by generalising the OTs to VOLE's. For a given security parameter $\lambda$, SoftSpokenOT reduces the amount of communication for each OT from $\lambda$ bits ([IKOS07]) to only $\lambda/k$ bits for any $k$, by using only Minicrypt assumptions [Imp95]. Reducing the communication effort increases the computational overhead by a factor of $2^{k-1}/k$, which is practical for small $k$. For this work, the focus lies on understanding the subspace VOLE protocol.

**IKNP to VOLE.** The authors show how to build a valid VOLE correlation based on the work by Ishai et al. to increase the efficiency of the protocol. For this, a PRG to extend $\binom{2}{1}$-OT to message length $\ell$ is used. The base OT sender, called $P_{\mathcal{S}}$ in the following, obtains two random strings $m_0, m_1$. The base OT receiver called $P_{\mathcal{R}}$, gets its choice bit $b \in \mathbb{F}_2$ and the corresponding message $m_b$. Based on this, the base OT sender computes $u = m_0 \oplus m_1$ and $v = 0 \cdot m_0 \oplus 1 \cdot m_1 = m_1$. The receiver calculates $\Delta = 1 \oplus b \in \mathbb{F}_2$ and furthermore $w = \Delta m_0 \oplus (1 \oplus \Delta) m_1 = m_b$. So if $b = 0$, $\Delta = 1$ then $m_b = m_0$ and the other way around. Notice that the receiver can perform this calculation because he learns the message $m_b$ as a result of the OT. Now $w \oplus v = \Delta m_0 \oplus \Delta m_1 = \Delta \boldsymbol{u}$, which is a VOLE correlation. This construction is called a $\mathbb{F}_2$-VOLE, since $P_{\mathcal{S}}$ gets a vector $\boldsymbol{u} \in \mathbb{F}_2^\ell$ and $P_{\mathcal{R}}$ gets a scalar $\Delta \in \mathbb{F}_2$. Both learn secret shares $v$ and $w$ of the product. Note, that in the IKNP protocol, the sender and receiver are turned around, such that the real sender is the receiver in the base OT protocol and later on the sender in the extending protocol. In this protocol, the

sender $P_{\mathcal{S}}$ is the sender in the base OT's and the receiver in the extending protocol. In this construction, $m_0$ and $m_1$ are chosen by the protocol and so $\boldsymbol{u}$ is random. If $P_{\mathcal{S}}$ now want to use its own $\boldsymbol{u}'$, it can send $\overline{\boldsymbol{u}} = \boldsymbol{u} \oplus \boldsymbol{u}'$ to $P_{\mathcal{R}}$ instead. $P_{\mathcal{R}}$ updates $w' = w \oplus \Delta\overline{\boldsymbol{u}}$ and so $w \oplus v = \Delta m_0 \oplus \Delta m_1 = \Delta \boldsymbol{u}'$, while hiding $\boldsymbol{u}'$. To extend the base OTs, the IKNP protocol calculates $\lambda$ of this $\mathbb{F}_2$ VOLE's.
To derandomise $\boldsymbol{u}$ to all be the same they send $\lambda \cdot \ell$ bits and thus get $\boldsymbol{W}^i \oplus \boldsymbol{V}^i = \Delta_i \boldsymbol{u}$ for $i$-th VOLE correlation. Summarised as a large matrix, this results in $\boldsymbol{W} \oplus \boldsymbol{V} = \boldsymbol{u}\vec{\boldsymbol{\Delta}}$, with $\vec{\boldsymbol{\Delta}}$ to be the row vector of all $\Delta_i$. Using $\boldsymbol{u}_j$ as a choice bit for the $\Delta$-OT, the j-th of the above correlation contains $\boldsymbol{W}_j \oplus \boldsymbol{V}_j = u_j\vec{\boldsymbol{\Delta}}$. Thus, $P_{\mathcal{R}}$ has learned $\vec{\boldsymbol{m}}_j^0 = \boldsymbol{W}_j$ and $\vec{\boldsymbol{m}}_j^1 = \boldsymbol{W}_j \oplus \vec{\boldsymbol{\Delta}}$. $P_{\mathcal{S}}$ instead got its choice bit $u_j$ and $\vec{\boldsymbol{m}}_{u_j} = \boldsymbol{V}_j$. Now $P_{\mathcal{S}}$, the sender of the base OT, is the receiver and $P_{\mathcal{R}}$, the receiver of the base OT, is the sender. Hashing $m_j^x$ finally decollates the OT messages from each other.

**VOLE for Small Fields.** In the next step, Roy now presents the final SoftSpokenOT construction, which uses elements from any small field $F_q$ by using $\binom{q}{q-1}$ OT's, where $q$ is just polynomially large with $\boldsymbol{u}$ taking values in any subfield $\mathbb{F}_p$ of $\mathbb{F}_q$. In addition to that a PRG: $\{0,1\}^\lambda \to \mathbb{F}_p^\ell$ is used. To realise such a protocol, [BGI17] can be used. The authors describe, how to realise an efficient $\binom{2^k}{2^k-1}$-OT protocol based on pseudo-random functions (PRF). This can be applied to any field $\mathbb{F}_{2^k}$ and is used to adapt the above protocol. The sender $P_{\mathcal{S}}$ now gets access to a random function $F : \mathbb{F}_{2^k} \to \mathbb{F}_2^\ell$, while the receiver $P_{\mathcal{R}}$ has a random point $\Delta$ and the restriction $F^*$ of $F$ to $\mathbb{F}_{2^k} \setminus \{\Delta\}$. The equations for the vectors $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ are described in the SoftSpokenOT paper [Roy22]:

$$\begin{aligned}
\boldsymbol{u} = F(0) \oplus F(1) &\implies \boldsymbol{u} = \bigoplus_{x \in \mathbb{F}_{2^k}} F(x) \\
\boldsymbol{v} = 0F(0) \oplus 1F(1) &\implies \boldsymbol{v} = \bigoplus_{x \in \mathbb{F}_{2^k}} xF(x) \\
\boldsymbol{w} = \Delta F^*(0) \oplus (1 \oplus \Delta)F^*(1) &\implies \boldsymbol{w} = \bigoplus_{x \in \mathbb{F}_{2^k}} (x \oplus \Delta)F^*(x)\,.
\end{aligned}$$

Note that the elements $\boldsymbol{m}_0$ and $\boldsymbol{m}_1$ from the general VOLE were exchanged by calling the function $F$ with 0 or 1 respectively. Furthermore, reducing the communication between sender and receiver by a factor of $k$ increases the computations that are needed by a factor $2^k/k$. So there are only $\lambda/k$ VOLEs, which require both parties to evaluate $F$ at every point in a field of size $2^k$. This can now be applied to the small field $\mathbb{F}_q$ using an Ideal Functionality $\mathcal{F}_{OT-\vec{1}}^{q,1,\mathcal{L}}$ for some linear Code $\mathcal{C}$ and some $\mathcal{L}$, were $\mathcal{L}$ denotes the set of allowed selective abort attacks. Notice, that $\mathcal{L}$ is just needed to use a simulator in the security proofs. In the real world, the adversary can perform so-called "selective abort attacks"

which are not possible in the simulator setting. To ensure that the adversary can not distinguish between the real and simulator world, the simulator gets all possible attacks as input such that he can deliver the correct results on the attacks. Let $\mathcal{C}$ be the length one, dimension one code, $\boldsymbol{G}_C = [1]$. This makes $\boldsymbol{U}, \boldsymbol{V}$ and $\boldsymbol{W}$ all become column vectors and $\vec{\boldsymbol{\Delta}}$ become a scalar. The protocol now works as follows: The sender $P_{\mathcal{S}}$ obtains $F$ from the Ideal Functionality $\mathcal{F}_{OT-\vec{1}}^{q,1,\mathcal{L}}$ The receiver $P_{\mathcal{R}}$ gets $x^*$ and $F^*$ from the same functionality. $P_{\mathcal{S}}$ computes then for all $x \in \mathbb{F}_q$ the corresponding $\boldsymbol{r}_x = \text{PRG}(F(x))$. Then $P_{\mathcal{S}}$ outputs $\boldsymbol{u}$, $\boldsymbol{v}$, such that $\boldsymbol{u} = \sum_{x\in\mathbb{F}_q} \boldsymbol{r}_x$ and $\boldsymbol{v} = -\sum_{x\in\mathbb{F}_q} \boldsymbol{r}_x\ x$. $P_{\mathcal{R}}$ instead set $\Delta = x^*$ and calculate for all $x \in \mathbb{F}_q \setminus \{\Delta\}$ the corresponding $\boldsymbol{r}_x = \text{PRG}(\mathbb{F}^*(x))$. Finally, the receiver outputs $\Delta$ and $\boldsymbol{w} = \sum_{x\in\mathbb{F}_q\setminus\{\Delta\}} \boldsymbol{r}_x(\Delta - x)$. This construction is a valid VOLE correlation since the following holds:

$$\boldsymbol{w} = \sum_{x\in\mathbb{F}_q\setminus\{\Delta\}} \boldsymbol{r}_x(\Delta - x) \overset{(1)}{=} \sum_{x\in\mathbb{F}_q} \boldsymbol{r}_x(\Delta - x) = \sum_{x\in\mathbb{F}_q} \boldsymbol{r}_x\ \Delta - \sum_{x\in\mathbb{F}_q} \boldsymbol{r}_x\ x = \boldsymbol{u}\Delta + \boldsymbol{v}$$

The first equation (1) holds since the $x = \Delta$ term would be multiplied by $(\Delta - \Delta)$ and so cancelled out. It can be shown, that this protocol is secure against a malicious sender and a malicious receiver [BGI17].

**Subspace VOLE.** Until here, the construction of $\mathbb{F}_2$-VOLE's from OT's and the generalisation for small fields $\mathbb{F}_q$ were described. In the following part, it is shown how to construct subspace VOLEs for an arbitrary field of polynomial size. The basic idea is to derandomise $\boldsymbol{U}$ in a way, such that $P_{\mathcal{S}}$ send a correction of $\boldsymbol{U}$ to make all columns to be identical. This ensures that each column would use the same set of choice bits. Other protocols, like [KK13], instead correct the rows of $\boldsymbol{U}$ to lie in an arbitrary linear code and not only in a repetition code. Before, each VOLE protocol follows the idea of the correlation of vectors, where $P_{\mathcal{S}}$ receives $\boldsymbol{w} - \boldsymbol{v} = \boldsymbol{u}\Delta$, with $\boldsymbol{u} \in \mathbb{F}_p^\ell$ and $\boldsymbol{v} \in \mathbb{F}_p^\ell$. $P_{\mathcal{R}}$ gets $\boldsymbol{v} \in \mathbb{F}_p^\ell$ and $\Delta \in \mathbb{F}_q$. For the subspace VOLE construction presented by Boyle et al., VOLE now produces a correlation of matrices $\boldsymbol{W}$ - $\boldsymbol{V}$ = $\boldsymbol{U}$ $\boldsymbol{G}_{\mathcal{C}}$ diag($\vec{\boldsymbol{\Delta}}$) [BCGI18]. This especially means, that $\boldsymbol{U}$ gets multiplied by the generator matrix $\boldsymbol{G}_C$ for a given linear Code $\mathcal{C}$. Notice, that the rows of $\boldsymbol{U}$ are then code words of $\mathcal{C}$. The subspace VOLE protocol by Roy works as follows: The sender $R_S$ gets $\boldsymbol{U}'$ and $\boldsymbol{V}$ from an Ideal Functionality $\mathcal{F}_{\text{VOLE}}^{p,q,\mathbb{F}_p^{n_{\mathcal{C}}},\ell,\{X\}}$, where $\boldsymbol{U}'$ denotes an $\ell \times k_C$ matrix over $\mathbb{F}_p$ and $\boldsymbol{V}$ denotes a $\ell \times n_C$ matrix over $\mathbb{F}_q$. The receiver $P_{\mathcal{R}}$ obtains $\vec{\boldsymbol{\Delta}}$ and $\boldsymbol{W}'$. Note, that for any output by the protocol $\boldsymbol{U}, \boldsymbol{V}, \vec{\boldsymbol{\Delta}}, \boldsymbol{W}$ and for every code $\mathcal{C}$ chosen by the adversarie, the output $\boldsymbol{U}', \boldsymbol{V}, \vec{\boldsymbol{\Delta}}, \boldsymbol{W}'$ is unique for an underlying VOLE correlation, for $\boldsymbol{U}' = [\boldsymbol{U}, \boldsymbol{C}]\, \boldsymbol{T}_{\mathcal{C}}$ and $\boldsymbol{W}' = \boldsymbol{W} + [0\, \boldsymbol{C}]\, \boldsymbol{T}_{\mathcal{C}}$ diag($\vec{\Delta}$). $P_{\mathcal{S}}$ divides $\boldsymbol{U}'$ into two parts $[\boldsymbol{U}, \boldsymbol{C}]$, the matrix $\boldsymbol{U}$ itself and a correction syndrome $\boldsymbol{C} \in \mathbb{F}^{\ell \times n_C - k_C}$, by multiplying $\boldsymbol{U}'$ with $\boldsymbol{T}_{\mathcal{C}}^{-1}$. In general, such syndromes are used to detect and correct errors in the transmission of

(linear) codes. In [Roy22], the receiver does not receive its part of the VOLE correlation directly, but must first correct it with the syndrome received from the sender. This is needed to prove security, as the receiver should not be able to learn the sender's inputs directly and therefore only receives incorrect values. Therefore, $P_{\mathcal{S}}$ sends the correction syndrome to $P_{\mathcal{R}}$, who then starts to correct $\boldsymbol{W}'$ to maintain the VOLE correlation property after $P_{\mathcal{S}}$ removes $\boldsymbol{C}$ from $\boldsymbol{U}'$. This construction is a valid VOLE correlation since the following holds:

$$\begin{aligned}\boldsymbol{W} &= \boldsymbol{W}' - [0,\, \mathcal{C}]\, \boldsymbol{T}_C \operatorname{diag}(\vec{\boldsymbol{\Delta}}) \\ &\overset{(2)}{=} \boldsymbol{V} + \boldsymbol{U}' \operatorname{diag}(\vec{\boldsymbol{\Delta}}) - [0,\, \mathcal{C}]\, \boldsymbol{T}_C \operatorname{diag}(\vec{\boldsymbol{\Delta}}) \\ &= \boldsymbol{V} + (\boldsymbol{U}' - [0,\, \mathcal{C}]\, \boldsymbol{T}_C) \operatorname{diag}(\vec{\boldsymbol{\Delta}}) \\ &= \boldsymbol{V} + \boldsymbol{U} \cdot \boldsymbol{G}_C \operatorname{diag}(\vec{\boldsymbol{\Delta}})\ .\end{aligned}$$

This protocol is just secure against a semi-honest sender $P_{\mathcal{S}}$. If $P_{\mathcal{S}}$ sends a wrong correction syndrome $\boldsymbol{C}$, the VOLE correlation does not hold anymore. This can be fixed using a consistency check protocol, using a linear $\epsilon$-almost universal hash family $\mathcal{R} \subseteq \mathbb{F}_q^{m \times \ell}$. To check the consistency, $P_{\mathcal{S}}$ send $\tilde{\boldsymbol{U}} = R\boldsymbol{U}$ and $\tilde{\boldsymbol{V}} = R\boldsymbol{V}$ to $P_{\mathcal{R}}$, for a given hash function $R \in \mathcal{R}$. Notice, that $R\boldsymbol{V}$ denotes applying the hash function $R$ on a matrix $\boldsymbol{V}$. $P_{\mathcal{R}}$ abort, if $\tilde{\boldsymbol{V}} \neq R\boldsymbol{W} - \tilde{\boldsymbol{U}} \cdot \boldsymbol{G}_C \operatorname{diag}(\vec{\boldsymbol{\Delta}})$, respectively output $\vec{\boldsymbol{\Delta}}$ and $W$ if it is consistent. The second equation (2) holds by the fact, that the ideal functionality outputs $\boldsymbol{W}' = \boldsymbol{V} + \boldsymbol{U}' \operatorname{diag}(\vec{\boldsymbol{\Delta}})$ if $P_{\mathcal{R}}$ is not corrupted. If the receiver is instead corrupt, he can choose $\boldsymbol{W}$ arbitrarily. In this case the ideal functionality chooses $\boldsymbol{V} = -(\boldsymbol{U} \cdot \boldsymbol{G}_C \operatorname{diag}(\vec{\boldsymbol{\Delta}})) + \boldsymbol{W}$ such that the equation holds. The whole protocol is shown in Figure 3.1.

In this subsection, the idea of extending OTs and the construction of a subspace VOLE was presented. This will be used in the following to construct ZKPs based on VOLE's.

### 3.2.2 Generalised Subspace VOLE Protocol

The Generalised Subspace VOLE Protocol presented by Baum et al. in 2023 is built on the SoftSpokenOT protocol by Roy [Roy22] and shows how to build VOLEs for exponentially large fields [BBdSG+23b]. To achieve this generalisation the authors limit their protocol by choosing the receiver secret $\Delta$ from a subset $\mathcal{S}_\Delta \subseteq \mathbb{F}_q^{m_2}$. The projected set $\mathcal{S}_\Delta^i$ itself has polynomial size and contains the $i$-th coordinate of every element of $S_\Delta$. In SoftSpokenOT, there are no such restrictions, but the subspace VOLEs are only constructed for polynomial-sized fields.

**Construction from SoftSpokenOT.** The basic construction is the same as in SoftSpokenOT, which is described in more detail in Section 3.2.1. In this part, only the differences are pointed out. The ideal functionality $\mathcal{F}_{\text{sVOLE}}^{p,q,S_\Delta,\mathbb{F}_p,\ell,\{2^{S_\Delta}\}}$ now chooses $\Delta$ from a given subspace $\mathcal{S}_\Delta \subseteq \mathbb{F}_q^{n_c}$, instead of choosing $\Delta$ as a random element from $\mathbb{F}_q^{n_c}$. The reason for performing this step is that the authors want to build VOLE correlations over exponential large fields, but also want to use the SoftSpokenOT protocol which just works with polynomial-sized fields. To ensure this, Baum et al. use polynomial sized sub-fields, so it is needed that the cardinality of the set $\mathcal{S}_\Delta$ is polynomial sized ($N = |\mathcal{S}_\Delta| = poly(\lambda)$) and it is required that $\{\boldsymbol{x}\}_{\boldsymbol{x}\in\mathcal{S}_\Delta\setminus\{f_1\}}$ spans $F_q$, viewed as a $k$-dimensional vector space over $\mathbb{F}_p$. This requirement enables the authors to state, that $(\boldsymbol{u}, \boldsymbol{v})$ are sampled independent and uniformly random since $\{(1, \boldsymbol{x})\}_{\boldsymbol{x}\in\mathcal{S}_\Delta}$ spans a $(k+1)$-dimensional vector space. Note that Baum et al. use different notations from Roy, $\mathcal{S}_\Delta = \{f_1, ..., f_N\}$ mean that $\mathcal{S}_\Delta$ contains $N$ elements from $\mathbb{F}_q$. In the SoftSpokenOT paper, these elements were notated by $x$, which now denotes the elements of $\mathcal{S}_\Delta$. Since the goal of the generalised subspace VOLE protocol is to build ZKPs, the authors substitute prover and verifier respectively for the sender and receiver. The prover takes the role of the sender in the SoftSpokenOT protocol, the verifier replaces the receiver. Another change in notation regards the VOLE correlation, replacing $\boldsymbol{W}$ with $\boldsymbol{Q}$. Before, a VOLE correlation was defined as $\boldsymbol{w} = \boldsymbol{u} \cdot \Delta + \boldsymbol{v}$, in the following it will be $\boldsymbol{q} = \boldsymbol{u} \cdot \Delta + \boldsymbol{v}$. This is also caused by applying this protocol to ZKP to distinguish between the witness used by the prover and the VOLE correlation.
At last, the authors adapt the communication model with the ideal functionality. Before, the sender and receiver got their values used in the VOLE correlation at the beginning of the protocol. In the new version, Baum et al. adapt the protocol to the init-get model, which simply means that the init call forces the ideal functionality to produce all values that are needed for the computations, but later on prover and verifier have to send a get-message to receive the values. In the protocol, the prover obtains his values at the beginning of the protocol, but, the verifier does not know his part of the correlation until the verification part. This is needed to prove the security of the system. The rest of the protocol behaves like in SoftSpokenOT for a small domain $\mathcal{S}_\Delta$.

**General Subspace VOLE.** This construction for $\mathbb{F}_p$ subspace VOLE can now be transformed to a more general $\mathbb{F}_p^n$ subspace VOLE protocol. For this, suppose $S_\Delta = S_\Delta^1 \times ... \times S_\Delta^n$, where $S_\Delta^i \in \mathbb{F}_q$. With this it is possible to execute $n$ instances of $\mathcal{F}_{\text{sVOLE}}^{p,q,S_\Delta,\mathbb{F}_p,\ell,\{2^{S_\Delta}\}}$ in parallel and build a single instance of subspace VOLE for $S_\Delta$. Note, that the vectors $\boldsymbol{u} \in \mathbb{F}_p^l$, $\boldsymbol{v} \in \mathbb{F}_q^l$ and $\boldsymbol{q} \in \mathbb{F}_q^l$ stack into matrices $\boldsymbol{U} \in \mathbb{F}_p^{l\times n}$, $\boldsymbol{V} \in \mathbb{F}_q^{l\times n}$ and $\boldsymbol{Q} \in \mathbb{F}_q^{l\times n}$.
Baum et al. adopt the subspace VOLE protocol, such that the rows of $\boldsymbol{U}$ lie in a subspace defined by an arbitrary linear Code $\mathcal{C}$. For this, they use the construction of SoftSpokenOT.

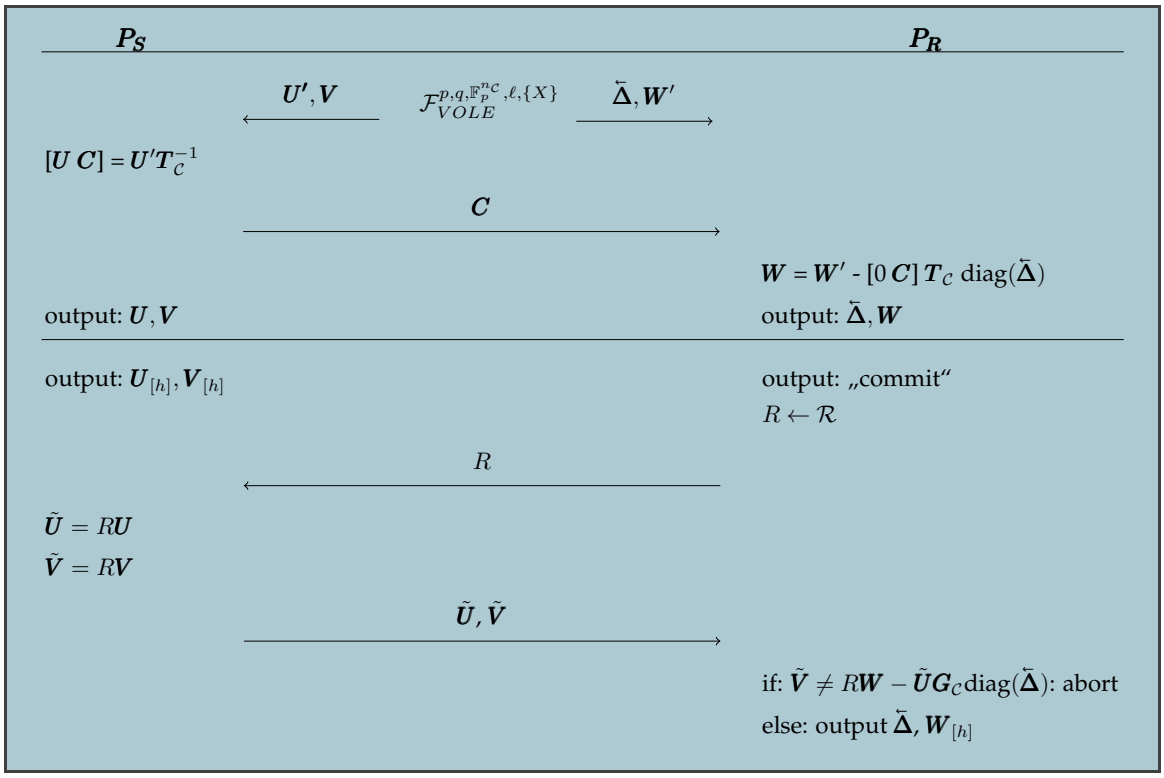



Figure 3.1: The full SoftSpokenOT protocol for subspace VOLE. The above part shows the subspace VOLE part, while the bottom part shows the consistency check which is needed to provide security against malicious $P_S$.

As discussed in Section 3.2.1, derandomising $\boldsymbol{U}$ by multiplying with $\mathcal{C}$'s generator matrix such that the rows of $\boldsymbol{U}$ contain code words of $\mathcal{C}$, works unless the prover $\mathcal{P}$ is malicious and send rows $U_i \notin \mathcal{C}$. To overcome this problem, Roy presented a consistency check to ensure the correctness of $\boldsymbol{U}$ using a linear $\epsilon$-almost universal hash family. Baum et al. point out two fundamental changes in proving the security of their construction, the choice of $\Delta$ as an element from an arbitrary set $S_\Delta$ instead of being uniform in a linear space. The second change is later needed to apply the Fiat Shamir transformation, to build a non-interactive protocol. For this, the adversary can restart the proof between sampling the hash function and the consistency check arbitrarily often. To realise this they will use a hash family $\mathcal{H} \subseteq \mathbb{F}_q^{r \times (\ell+h)}$ consisting of $\ell$-hiding, $\epsilon$-universal linear hash functions. The proof itself is shown in [BBdSG+23b]. Using this protocol enables them to use generalised subspace VOLE's in a fully malicious setting using an ideal functionality $\mathcal{F}_{\text{sVOLE}}^{p,q,S_\Delta,\mathbb{F}_p,\ell,\{2^{S_\Delta}\}}$ and a family of $\ell$-hiding, $\epsilon$-universal linear hash functions.

# 4 Signatures from Zero-Knowledge Protocols

In this chapter, we introduce the two protocols, MPCitH and VOLEitH, in more detail. We explain how to build Zero Knowledge Proofs based on MPC protocols and VOLEs and how they can be used to generate digital signatures.

## 4.1 Multi Party Computation in the Head

The field of post-quantum cryptography deals with the question of how to protect systems against an attacker who has access to a large quantum computer [BL17]. It has been shown that many of our current systems are vulnerable to such adversaries. Therefore, post-quantum cryptosystems are needed to protect us even in this scenario. To face this challenge, the National Institute of Standards and Technology (NIST) started several rounds to standardise algorithms that are secure against post-quantum adversaries[1]. The focus is primarily on asymmetric cryptography, such as digital signatures or Key Encapsulation Mechanisms (KEM), as these are easy to adapt to the post-quantum scenario and are widely used in today's cryptography [CCJ+16]. Over the last few years, the first protocols have already been standardised.
Many proposals have already been submitted through the various standardisation rounds. It has been shown that Multi-Party Computation can be used to realise efficient and secure post-quantum signatures based on zero-knowledge protocols. In the following subsection, we discuss how to build efficient Zero-Knowledge Proofs based on Multi-Party Computation and how to to leverage these protocols to create signatures.

### 4.1.1 Zero-Knowledge from MPC

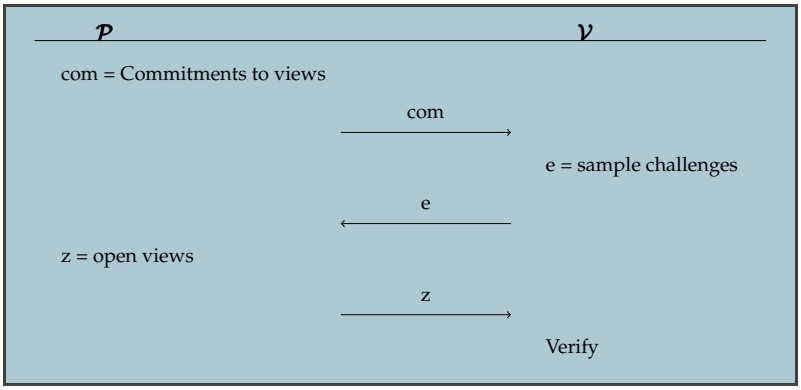
In 2007, Ishai et al. presented a new way of creating efficient zero-knowledge protocols based on MPC, called Multi-Party Computation in the Head (MPCitH) [IKOS07]. The idea is the following: The prover takes any MPC protocol which is secure against $t$ semi-honest parties. Then he generates $n$ virtual parties and shares the witness into $n$ sharings, using a polynomial or additive sharing. Now, the prover can run the MPC protocol by controlling the $n$ parties and receiving the views of each party. He sends the commitment of each view to the verifier. The verifier chooses two different views and requests the openings. The

[1] NIST post-quantum cryptography: `https://www.nist.gov/pqcrypto`

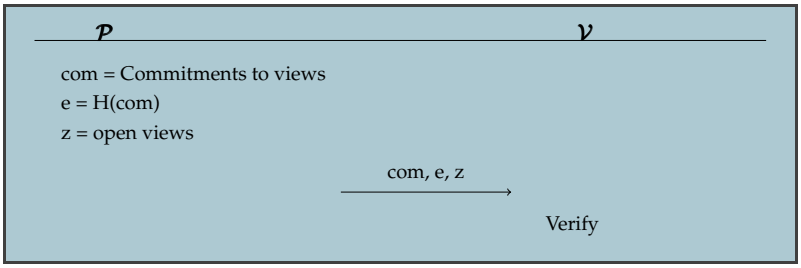
prover sends the openings and the verifier can check if the openings are consistent among themselves and the commitments received before. This approach is called MPCitH since the prover generates and simulates $n$ parties "in his head".
This protocol describes a valid Zero-Knowledge Proof. Completeness is fulfilled, since if the prover knows the witness he can produce a valid sharing and perform the correct MPC protocol such that all views are consistent. But if a malicious prover tries to cheat there must be two or more views that are not consistent with each other. The verifier opens two views at each run of the protocol, so the probability of finding the inconsistent views is at least $1/\binom{n}{2}$. Notice that the binding property of the commitment forces the malicious prover to open the real value. The parties can rerun the protocol multiple times. After $k$ rounds the probability that the verifier accepts the proof of a malicious prover is $(1-1/\binom{n}{2})^k$, so soundness is ensured given that $k$ is big enough. To show zero knowledge, two simulators are required. The first one simulates the MPC protocol and generates the views of the parties. The second one now uses the first simulator to generate the views and generates a valid commitment to two arbitrary views, the remaining commitments are arbitrary or zero. Now, the verifier sends his request to open the commitments. If the request fits the two consistent commitments, the simulator opens the commitments. The probability, that the verifier chooses the two consistent commitments is at least $1/\binom{n}{2}$. If not, the simulator aborts and restarts the generation of the views until he generates the correct commitments. Notice that the verifier can not distinguish between the commitments created by the simulator and a real prover due to the hiding property of the commitment scheme.
At this point, MPCitH can be used to build ZKPs. Since the idea was published, many optimisations have been developed and applied to enable real-world usability.

### 4.1.2 Signatures from MPCitH

The above MPCitH construction describes an interactive protocol between a prover and a verifier. However, for signatures, it would be desirable if the protocol were non-interactive. Then the prover, or in this case, the signer could send the signature with his message and the verifier can check it at any time and without further communication. This can be realised by applying the Fiat-Shamir-Transformation to the protocol. Instead of sending the commitments, receiving a challenge and sending the answer, the signer now prepares the commitments, chooses the challenge by applying a hash function to the commitments and picks the corresponding values to the challenge. He now sends everything to the verifier. This is also shown in Figure 4.1. The Fiat-Shamir-Transformation now ensures that if the interactive protocol is a valid ZKP, the non-interactive protocol is also a valid ZKP. Notice that it must be ensured that the signer cannot choose the challenges




(a) Interactive Zero-Knowledge Proof.




(b) Non-Interactive Zero-Knowledge Proof

Figure 4.1: Interactive and non-interactive Zero-Knowledge Proof for MPCitH using the Fiat-Shamir-Transformation.

such that the verifier accepts a false proof. This is secure since the signer needs infeasible many calls to the Random Oracle and thus is not able to cheat on the verifier. This was shown by [PS96] and also holds in the post-quantum setting [DFMS19]. Finally, to sign a message $m$ the signer only has to send $e = H(\text{com}||m)$ and thus the verifier can check the ZKP and the corresponding message.

Two very popular MPCitH signature methods, which were standardised by the NIST, are Banquet [BdSGK$^+$21] and Picnic [CDG$^+$20]. The idea of both protocols is very similar, take a One-Way-Function $f_k$, depending on a key $k$, and calculate $y = f_k(x)$ for a given secret input $x$. The prover can now build a ZKP and shows that he knows a suitable $x$ for a given $y$. Due to the properties of the OWF, it is possible that a (malicious) prover can find a $x$ that produces the correct output without knowing it at the beginning of the proof with a very low probability. After applying the Fiat-Shamir-Transformation, such a protocol can be used as a valid non-interactive Zero-Knowledge Proof. Picnic relies on LowMC as the OWF and Banquet uses AES. In general, AES is more expensive but many optimisations have been developed that make AES-based systems more efficient. However, this scheme can be used to build various signatures based on different One-Way Functions.

## 4.2 Vector Oblivious Linear Evaluation in the Head

As discussed before, building efficient Zero-Knowledge Proofs is needed to find secure, but also practical algorithms that can be used to achieve post-quantum security. Multi-Party Computation in the Head (MPCitH) delivered the idea of simulating $n$ parties "in the head" of the prover and thus reducing computational and commuincational overhead. A new approach to build up ZKPs based on generalised subspace VOLEs, is also used in the work by Baum et al. in 2023 [BBdSG+23b]. The authors state that the proof size of current MPCitH protocols often scales linearly with the size of their circuit. Such protocols usually build up on symmetric cryptography, which easily can be made secure in the post-quantum scenario. So this approach is good for small- or medium-sized (arithmetic or boolean) circuits but gets worse when increasing the size. Ligero, a protocol to build zero-knowledge arguments for NP by building upon an honest-majority MPC protocol, achieves the reduction of the proof size to the square root of the circuit size [AHIV17]. A huge disadvantage of this approach is the high amount of Reed-Solomon encoding operations and consistency checks, which increase the computational costs for the prover and the verifier. Thus, Ligero is an improvement, but only if the circuit size is large enough. Therefore, new approaches try different methods like Vector Oblivious Linear Evaluations. The idea is that the prover uses VOLEs to commit to its witness and later on the verifier can check relations on these commitments using information-theoretic techniques. Such protocols reduce the communication to one field element per multiplication gate and can be computed very efficiently [DIO20], [WYKW21]. The downside is that they are designated verifier proofs since the verifier has to keep a secret state, namely his part of the VOLE correlation. Revealing this would break the soundness of the protocol.
To overcome this problem and create efficient zero-knowledge arguments, [BBdSG+23b] presented VOLE in the Head (VOLEitH), a proof system that builds on standard symmetric cryptographic primitives and is publicly verifiable, just like MPCitH, by using generalised subspace Vector Oblivious Linear Evaluation. The following subsection discusses this protocol in more detail and describes how the individual rounds work together. It also shows what adjustments were made to the Quicksilver protocol [YSWW21], such that it fits the rest of the protocol. Furthermore, the creation of digital signatures, based on VOLEitH, will be discussed.

### 4.2.1 Zero-Knowledge from Generalised Subspace VOLE

In this subsection, the VOLEitH protocol is presented in more detail. Since there are many optimisations and proof structures inherited from other protocols, we show the high-level idea and explain how they work together. The entire protocol is shown in Figure 4.2.

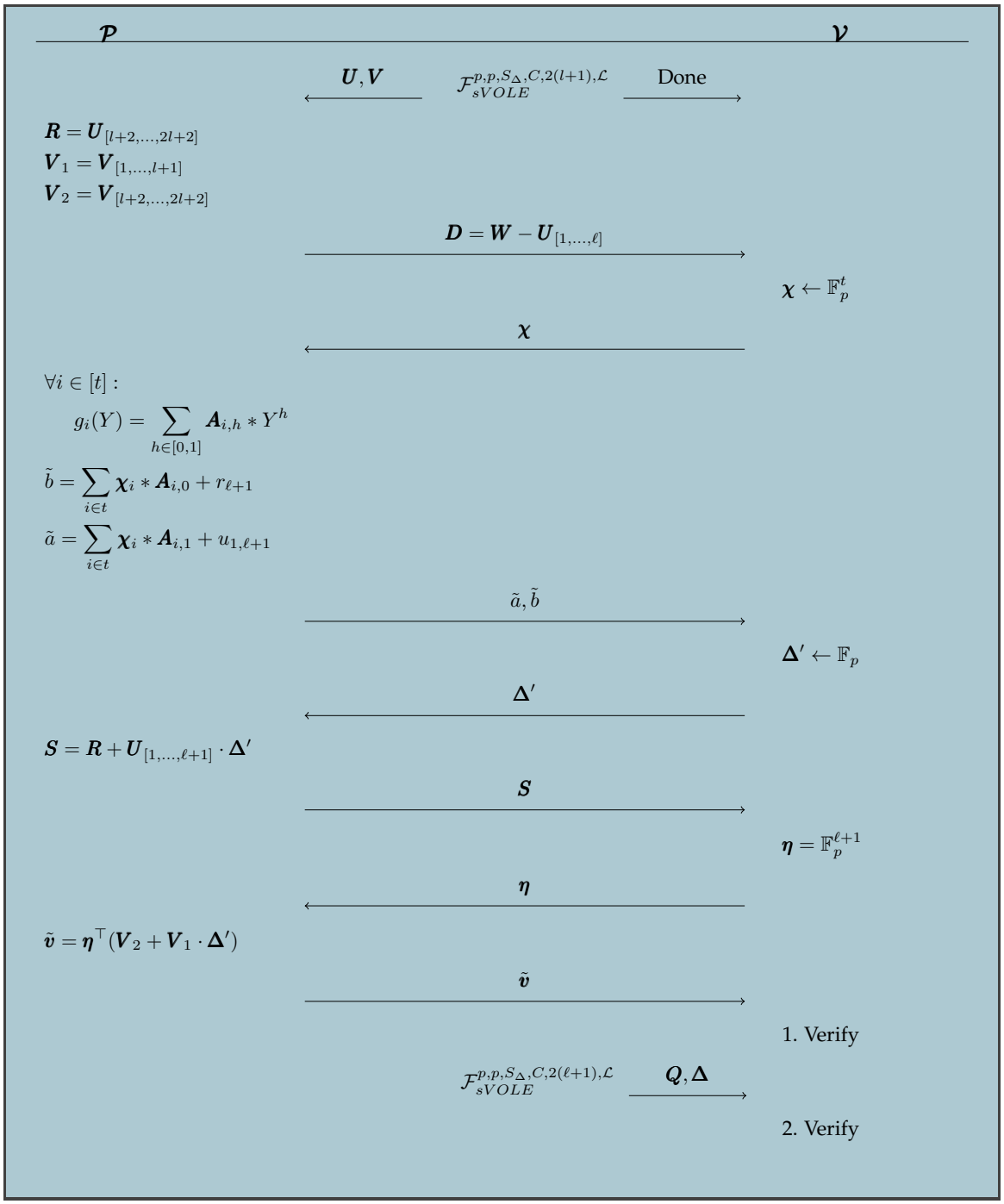



Figure 4.2: The VOLEitH protocol to build efficient public coin Zero-Knowledge Proofs between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ [BBdSG+23b].

**Preparation:** The protocol itself is parametrised with an $[n_{\mathcal{C}}, k_{\mathcal{C}}, d_{\mathcal{C}}]$ linear code $\mathcal{C}$, the set $S_\Delta$ and a leakage-space $\mathcal{L}$. As discussed in Section 3.2.2, the linear code is needed to derandomise elements by embedding them into the code. The leakage space is used to realise the security proof for the protocol since the attacker could perform other attacks on the system that can not be simulated by the simulator. Using these attacks enables the adversary to distinguish between the real and ideal world, so the results of the attacks have to be transferred to the simulator, such that he is enabled to return the attack outputs. To apply the SoftSpoken OT protocol, which works over polynomial-sized fields, the polynomial-sized set $S_\Delta = (S'_\Delta)^{n_{\mathcal{C}}} \subset \mathbb{F}_p^{n_{\mathcal{C}}}$ is needed and enables one to build valid VOLE instances for polynomial sized fields, even if $\mathbb{F}_p^{n_{\mathcal{C}}}$ is exponentially large.
At the beginning of the protocol, both parties receive a set of polynomials as input. Each polynomial $f_i$, for $i \in [t]$ has at least $k_{\mathcal{C}}\ell$ variables and a degree at most two. Those polynomials are needed for the zero-knowledge protocol provided by [YSWW21], where the prover wants to show that he knows a witness $w$, such that for all $i \in [t]$, $f_i(w) = 0$ holds. The polynomials are the encodings of the multiplications within an OWF and the prover has to show that he has calculated everything correctly. Thus, for $t$ multiplications there are exactly $t$ polynomials that need to be checked. This type of proving technique is also called *nullity check* in the literature. Both parties get the same polynomials $f_i$. This is later needed by the verifier to confirm that the prover provided a valid proof. The prover also receives the witness $w$ of length $k_{\mathcal{C}}\ell$ as input. To perform matrix calculations on the witness, the authors assume that $w$ can be split up into $\ell$ vectors.

**Round 1:** The Ideal Functionality provides the parties with all the values they need to calculate the VOLE correlations. [BBdSG+23b] use an adaptation of the Ideal Functionality of [Roy22], which interacts with the prover, the verifier and an attacker. After the init-message from prover and verifier, the functionality randomly selects the values $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{\Delta}$ from the respective fields and sets $\boldsymbol{Q} = \boldsymbol{V} + \boldsymbol{U} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})$. If the prover is corrupt, $\boldsymbol{U}$ and $\boldsymbol{V}$ are sampled by the attacker and $\boldsymbol{Q}$ is recalculated. In addition, a malicious prover is allowed to send a leakage query which contains all additional information available to the attacker. But if the verifier is corrupt, $\boldsymbol{\Delta}$ and $\boldsymbol{Q}$ are chosen by the attacker and $\boldsymbol{V} = \boldsymbol{Q} - \boldsymbol{U} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})$ is returned to the prover. This recomputation of $\boldsymbol{V}$ is needed to ensure a correct VOLE correlation. The Ideal Functionality sends the values $\boldsymbol{U}$ and $\boldsymbol{V}$ to the prover. The verifier values are withheld until $\mathcal{V}$ sends the get message at the end of the protocol. The Ideal Functionality then checks whether the secret $\boldsymbol{\Delta}$ is part of the attacker's leakage queue. If so, the check failed and it is aborted. If not, $\boldsymbol{\Delta}$ and $\boldsymbol{Q}$ are sent to the verifier. In the end, the prover wants to convince the verifier, that he knows a valid witness $w$ and all calculations fit the VOLE correlation $\boldsymbol{Q} = \boldsymbol{V} + \boldsymbol{U} \cdot \boldsymbol{\Delta}$.

The huge improvement of VOLEitH, in contrast to other VOLE-based protocols, is that the Zero-Knowledge Proof is no longer a designated verifier proof. To realise that, Baum et al. observed that $\boldsymbol{Q}$ and $\boldsymbol{\Delta}$ do not have to be known to the verifier during the calculations, but are only required during the verification. At this point, all calculations by the (malicious) prover are completed, so the prover cannot do anything to break the soundness, even if he knows $\boldsymbol{\Delta}$. Hence, $\boldsymbol{\Delta}$ no longer needs to be kept secret. Thus, the interaction between the verifier and the Ideal Functionality is split up, in the first round $\mathcal{V}$ only receives a message "done" and in the verification, he gets $\boldsymbol{Q} = \boldsymbol{V} + \boldsymbol{U} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \text{diag}(\boldsymbol{\Delta})$ and $\boldsymbol{\Delta} \in S_{\boldsymbol{\Delta}}$. The prover receives his values $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{F}_p^{(2\ell+2\times n_{\mathcal{C}})}$ and divides them into two parts, such that each sub-matrix has $\ell+1$ rows, $\boldsymbol{U} = \binom{\boldsymbol{U}_1}{\boldsymbol{R}}$ and $\boldsymbol{V} = \binom{\boldsymbol{V}_1}{\boldsymbol{V}_2}$. All values are usually chosen completely randomly unless the prover or verifier interacts maliciously. In this case, the Ideal Functionality allows the malicious party to choose its own values.
After receiving all the values, the prover commits to its witness $\boldsymbol{w}$ by sending $\boldsymbol{D}$, with $\boldsymbol{D} = \boldsymbol{W} - \boldsymbol{U}_{[1,\dots,\ell]}$, to the verifier. Here, $\boldsymbol{W}$ denotes the matrix containing all $\ell$ vectors $\boldsymbol{w}_i$ as rows. This step is later needed in the verification to validate, that the prover knows the correct witness $w$ and thus provides the soundness of the protocol. It is also ensured, that the verifier does not learn the witness, which is an important feature of Zero-Knowledge Proofs since $\mathcal{V}$ never learns $\boldsymbol{U}_{[1,\dots,\ell]}$ and thus can never remove the masking of $\boldsymbol{W}$.

**Round 2:** The verifier samples a challenge $\boldsymbol{\chi}$ and sends it to the prover. This is an optimisation presented by the Quicksilver protocol. The authors state, that for $t$ multiplication gadgets there will be $t$ equations of the form $\boldsymbol{B} = \boldsymbol{A}_0 + \boldsymbol{A}_1 \cdot \boldsymbol{\Delta}$ that have to be checked by the verifier. Notice that $\boldsymbol{B}$ and $\boldsymbol{\Delta}$ are held by the verifier and $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$ are provided by the prover. Furthermore, this equation forms a valid VOLE correlation. Instead of solving $t$ equations, $\boldsymbol{\chi}$ now can be used to solve only the following equation to verify the proof:

$$\sum_{i\in[t]} \boldsymbol{B}_i \cdot \boldsymbol{\chi}^i = \sum_{i\in[t]} \boldsymbol{A}_{0,i} \cdot \boldsymbol{\chi}^i + \sum_{i\in[t]} \boldsymbol{A}_{1,i} \cdot \boldsymbol{\chi}^i \cdot \boldsymbol{\Delta}\,.$$

Baum et al. now adopt this optimisation. Instead of using one scalar $\boldsymbol{\chi}$ and calculate $\boldsymbol{\chi}^i$ for $i \in [t]$, the challenge $\boldsymbol{\chi}$ will be a vector containing $t$ values $\boldsymbol{\chi}_1, \dots, \boldsymbol{\chi}_t$. Furthermore, they use $\boldsymbol{\Delta}'$ instead of the real $\boldsymbol{\Delta}$. Since $\boldsymbol{\Delta}'$ is a random element from a smaller field $\mathbb{F}_p$, all corresponding calculations can be carried out more efficiently than with $\boldsymbol{\Delta}$. In addition, $\boldsymbol{\Delta}$ has to stay secret to provide the soundness of the protocol, which was the essential problem of designated verifier proofs.

**Round 3:** In this round, the prover performs the Zero-Knowledge Proof and calculates all values that are needed by the verifier to confirm the correctness of the proof. As before, the Quicksilver protocol is used to build an efficient ZKP. One feature, that is exploited by this proof, is that the polynomials $f_i$ are degree separating polynomials, which simply means that any polynomial $f_i$ with degree $d$ can be represented by $\sum_{h=0}^{d} f_{i,h}$ so that all elements in $f_{i,h}$ have exactly degree $h$. Quicksilver now uses these polynomials. The prover evaluates the polynomials $f_i$ and thus builds a valid Zero-Knowledge Proof. To do this, he creates a polynomial $g_i$ for each $i \in [t]$ such that:

$$\begin{aligned} g_i(Y) &= \sum_{h\in[0,2]} f_{i,h}(r_1 + w_1 \cdot Y, ..., r_\ell + w_\ell \cdot Y) \cdot Y^{2-h} \\ &= f_i(w_1, ..., w_\ell) \cdot Y^2 + \sum_{h\in[0,1]} \boldsymbol{A}_{i,h} \cdot Y^h, \end{aligned}$$

where $\boldsymbol{A}_{i,h}$ is the aggregated coefficient for all terms with $Y^h$ and $i \in [t]$. Remember, $f_i(w_1, ..., w_\ell)$ will be zero if the prover is honest and holds a correct witness $w$.
In the next step, the prover calculates all check values that we have discussed in Round 2, so he calculates:

$$\tilde{b} = \sum_{i\in[t]} \boldsymbol{\chi}_i \cdot \boldsymbol{A}_{i,0} + r_{\ell+1},$$

$$\tilde{a} = \sum_{i\in[t]} \boldsymbol{\chi}_i \cdot \boldsymbol{A}_{i,1} + u_{\ell+1}.$$

In addition to the above equation from Quicksilver, Baum et al. add a masking value to the coefficients, namely $r_{\ell+1}$ and $u_{\ell+1}$, which can later be used to recompute the correct coefficients during the verification. Now, $\mathcal{V}$ sends $\tilde{a}$ and $\tilde{b}$ to the verifier. This will later be used in the verification to check the above equation, which will be discussed later in the verification phase. Notice, that this part of the protocol will be performed "in the head" of the prover since the prover now evaluates all polynomials $f_i$ for $i \in [t]$ to reduce the proof size.

**Round 4:** The verifier now samples $\boldsymbol{\Delta}'$ and sends it to the prover. This is necessary since $\boldsymbol{\Delta}$ is needed to build the VOLE correlation in the Quicksilver protocol, as discussed in Round 2, which can not be published to ensure the soundness of the zero-knowledge protocol. Instead of $\boldsymbol{\Delta}$, the value $\boldsymbol{\Delta}'$ can be used to build the VOLE correlation, such that $\tilde{s} = \tilde{b} + \tilde{a} \cdot \boldsymbol{\Delta}'$. This step hides the original $\boldsymbol{\Delta}$ and also ensures the public coin property since $\boldsymbol{\Delta}'$ is sampled at random.

**Round 5:** In this round, the prover uses $\boldsymbol{\Delta}'$ to prepare $S$, which is later needed to prove the correctness of the zero-knowledge protocol. So, $\mathcal{P}$ sends $\boldsymbol{S}$, with $\boldsymbol{S} = \boldsymbol{R} + \boldsymbol{U}_{1,...,l+1} \cdot \boldsymbol{\Delta}'$, to $\mathcal{V}$. This can later be used to verify the correctness of the VOLE correlation formed in Round 3, which will be discussed in the verification step.

**Round 6:** In the case of a malicious prover, it could be possible that he does not send the correct value $\boldsymbol{S}$, but any arbitrary value $\boldsymbol{S}^*$. To detect this, the verifier can send another challenge $\boldsymbol{\eta}$ to the prover.

**Round 7:** The prover now calculates $\tilde{\boldsymbol{v}} = \boldsymbol{\eta}^\top (\boldsymbol{V}_2 + \boldsymbol{V}_1 \cdot \boldsymbol{\Delta}')$ and sends it to the verifier. Once the verifier has learned the correct VOLE Correlation $\boldsymbol{Q} = \boldsymbol{V} + \mathcal{C}(\boldsymbol{S}) \cdot \boldsymbol{\Delta}$, he can verify the subspace VOLE relation between $\tilde{\boldsymbol{v}}$ and $\boldsymbol{\eta}^\top \mathcal{C}(S)$.

**Verification:** The verification is carried out by the verifier and serves to check the correctness of the Zero-Knowledge Proof. It can be divided into two parts, checking whether the ZKP was carried out correctly and whether the prover sent the correct values.
To check the Quicksilver proof, the above-discussed equation $\tilde{s} = \tilde{b} + \tilde{a} \cdot \boldsymbol{\Delta}'$ must be satisfied. Since $\tilde{a}$ and $\tilde{b}$ were sent by the prover, the focus now relies on how to calculate $\tilde{s}$ and why this delivers the correctness of the proof itself. The verifier starts calculating $\boldsymbol{S}' = \boldsymbol{S} + \binom{\boldsymbol{D}}{0} \cdot \boldsymbol{\Delta}' = \boldsymbol{R} + \binom{\boldsymbol{W}}{u_{\ell+1}} \cdot \boldsymbol{\Delta}'$, which simply follows by the commitment $\boldsymbol{D}$, which was send in Round 1, and $\boldsymbol{S}$ received in Round 4. Notice, that the first $\ell$ rows of $\boldsymbol{S}'$ have the form: $r_j + w_j \cdot \boldsymbol{\Delta}'$ and $s'_{\ell+1} = r_{\ell+1} + u_{\ell+1} \cdot \boldsymbol{\Delta}'$. Based on this, $\mathcal{V}$ can now build a polynomial $c_i$, using the common polynomials $f_i$, i.e.:

$$\begin{aligned} c_i(Y) &= \sum_{h \in [0,2]} f_{i,h}(s'_1, ..., s'_\ell) \cdot Y^{2-h} \\ &= \sum_{h \in [0,2]} f_{i,h}(r_1 + w_1 \cdot \boldsymbol{\Delta}', ..., r_\ell + w_\ell \cdot \boldsymbol{\Delta}') \cdot Y^{2-h}, \end{aligned}$$

which exactly matches the calculation for $g_i(Y)$ performed by $\mathcal{P}$, if you evaluate $g_i$ on $\boldsymbol{\Delta}'$. Again, $c_i(Y)$ for $i \in [t]$ can be can be checked by solving $t$ equations or using $\boldsymbol{\chi}$ to build one equation, namely $\tilde{s}$:

$$\tilde{s} = \sum_{i \in [t]} \boldsymbol{\chi}_i \cdot c_i(\boldsymbol{\Delta}') + s_{\ell+1}.$$

The value $s_{\ell+1}$ has to be added, since the values $\tilde{a}$ and $\tilde{b}$ are masked using $u_{\ell+1}$ and $r_{\ell+1}$, which is exactly the last row of $\boldsymbol{S}'$. Finally, the equation $\tilde{s} = \tilde{b} + \tilde{a} \cdot \boldsymbol{\Delta}'$ can be checked, if

$\tilde{a}$, $\tilde{b}$, $\boldsymbol{S}$ are transmitted correctly and the prover knows a valid witness $w$. The equation is satisfied since both parties evaluate the same polynomials on the same points. It is also ensured that $\mathcal{V}$ does not learn $w$, since it is always masked by $\boldsymbol{U}$ respectively $\boldsymbol{R}$, which are unknown to the verifier.
The second verification step is needed to ensure, that $\boldsymbol{S}$ was correctly sent by the prover. For this, $\mathcal{V}$ provided the challenge $\boldsymbol{\eta}$ and the prover replied using $\boldsymbol{V}_1, \boldsymbol{V}_2$ and $\boldsymbol{\Delta}'$. The verifier interacts again with the ideal functionality and receives his values $\boldsymbol{Q}$ and $\boldsymbol{\Delta}$. To understand, how to prove the correctness of $\boldsymbol{S}$, a deeper look at the values of $\boldsymbol{Q}$ is needed. Remember, that $\boldsymbol{Q}$ was sent by the ideal functionality and the following holds:

$$\begin{aligned} \boldsymbol{Q} &= \boldsymbol{V} + \boldsymbol{U} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta}) \\ &= \binom{\boldsymbol{V}_1}{\boldsymbol{V}_2} + \binom{\boldsymbol{U}_{[1,\dots,\ell+1]}}{\boldsymbol{R}} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta}) \,. \end{aligned}$$

Furthermore, $\boldsymbol{Q}_1 = \boldsymbol{V}_1 + \boldsymbol{U}_{[1,\dots,\ell+1]} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})$ and $\boldsymbol{Q}_2 = \boldsymbol{V}_2 + \boldsymbol{R} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})$. In the check itself, the verifier now calculates $\boldsymbol{Q}_2 + \boldsymbol{Q}_1 \cdot \boldsymbol{\Delta}'$. This is a valid check for $\boldsymbol{S}$:

$$\begin{aligned} \boldsymbol{Q}_2 + \boldsymbol{Q}_1 \cdot \boldsymbol{\Delta}' &= (\boldsymbol{V}_2 + \boldsymbol{R} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})) + (\boldsymbol{V}_1 + \boldsymbol{U}_{[1,\dots,\ell+1]} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})) \cdot \boldsymbol{\Delta}' \\ &= (\boldsymbol{V}_2 + \boldsymbol{R} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})) + (\boldsymbol{V}_1 \cdot \boldsymbol{\Delta}' + \boldsymbol{U}_{[1,\dots,\ell+1]} \cdot \boldsymbol{\Delta}' \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta})) \\ &= (\boldsymbol{V}_2 + \boldsymbol{V}_1 \cdot \boldsymbol{\Delta}') + (\boldsymbol{R} + \boldsymbol{U}_{[1,\dots,\ell+1]} \cdot \boldsymbol{\Delta}') \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta}) \\ &= \tilde{\boldsymbol{v}} + \boldsymbol{S} \cdot \boldsymbol{G}_{\mathcal{C}} \cdot \mathrm{diag}(\boldsymbol{\Delta}) \\ &= \tilde{\boldsymbol{v}} + \mathcal{C}(\boldsymbol{S}) \cdot \mathrm{diag}(\boldsymbol{\Delta}) \,. \end{aligned}$$

This equation only holds if the prover sends a correct value for $\boldsymbol{S}$ and forms a valid VOLE correlation. Notice, that the authors use $\boldsymbol{\eta}$ to reduce the communication costs of this proof since $\mathcal{P}$ only has to send a vector of length $\ell + 1$ instead of sending an $(2\ell + 2 \times n_{\mathcal{C}})$-dimensional matrix. This can easily be applied by multiplying $\boldsymbol{\eta}^\top$ by the equation above. Additionally, $\boldsymbol{\eta}$ ensures that the prover has to commit to its value for $\tilde{\boldsymbol{v}}$.

**Malicious Prover:** After presenting the protocol, we describe the possibilities of a malicious prover to pass the final checks, without knowing a correct witness $w$, in more detail. Furthermore, we will highlight how the above checks assist the verifier in identifying the incorrectness inside the proof, which is also shown in Figure 4.3.
After receiving the values from the Ideal Functionality, the malicious prover $\mathcal{P}^*$ commits to his invalid witness, $w^*$. Since $w^*$ is not a correct witness, there will be a polynomial $f_i$, such that $f_i(w^*) = \boldsymbol{e}i \neq 0$. After receiving $\boldsymbol{\chi}$ by $\mathcal{V}$, $\mathcal{P}^*$ starts to prepare the polynomials $g_i$ and calculates $\tilde{a}$ and $\tilde{b}$. Now, the malicious prover can send the correct values $\tilde{a}$ and $\tilde{b}$ or he samples two error values, i.e. $E_{\tilde{a}}, E_{\tilde{b}} \in \mathbb{F}_p^{kc}$, and sends $\tilde{a} + E_{\tilde{a}}$ respectively $\tilde{b} + E_{\tilde{b}}$

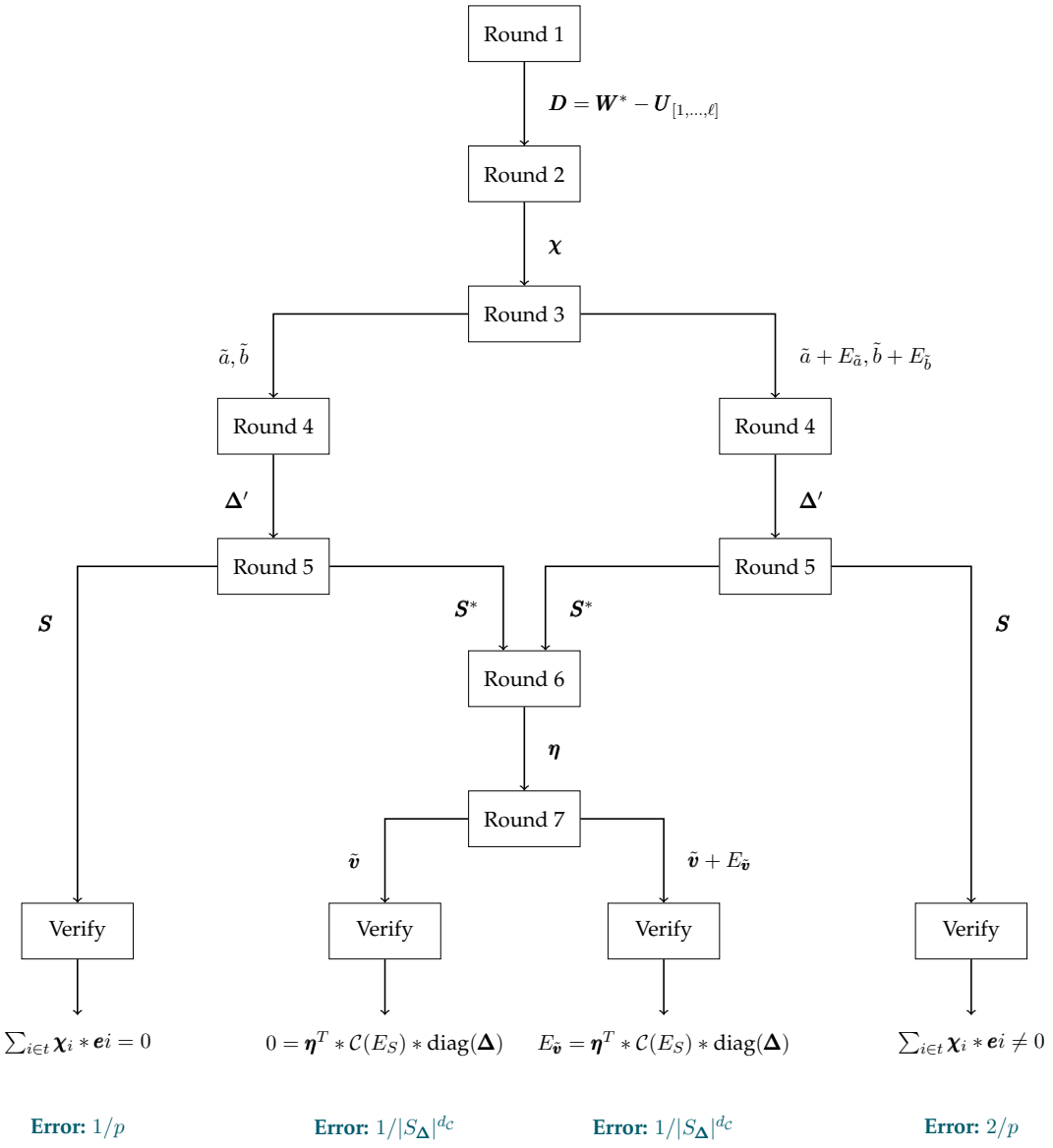



Figure 4.3: Execution of the VOLEitH protocol by a malicious prover $\mathcal{P}^*$ using an invalid witness $w^*$. It is shown, which values of $\mathcal{P}^*$ can be sent and how the challenges help the verifier to recognise the false witness. The soundness error is highlighted in colour.

instead. As discussed before, the verifier now gets the value $\boldsymbol{S}$ by the malicious prover, which again can be the correct $\boldsymbol{S}$ or a malicious value $\boldsymbol{S}^*$, and which is needed by the $\mathcal{V}$ to reconstruct the polynomials and check the Quicksilver constraints.
At this point, the first verification step enables the verifier to detect the incorrect witness. In the case of getting the correct values $\tilde{a}$, $\tilde{b}$ and $\boldsymbol{S}$, the equation $\tilde{s} = \tilde{b} + \tilde{a} \cdot \boldsymbol{\Delta}'$ only holds if the summation of all $\boldsymbol{\chi}_i \cdot \boldsymbol{e}i = 0$. This follows from the fact that there is a part of $w^*$, such that $f_i(w^*) = \boldsymbol{e}i \neq 0$ and so $\tilde{s} = \tilde{b} + \tilde{a} \cdot \boldsymbol{\Delta}'$ is equivalent to $\sum_i \boldsymbol{\chi}_i \cdot \boldsymbol{e}i \cdot \boldsymbol{\Delta}' = E_{\tilde{b}} + E_{\tilde{a}} \cdot \boldsymbol{\Delta}'$. In the case of a correct $\tilde{a}$ and $\tilde{b}$, $E_{\tilde{a}} = E_{\tilde{b}} = 0$ and thus the summation of $\boldsymbol{\chi}_i \cdot \boldsymbol{e}i$ has to be zero since $\boldsymbol{\Delta}'$ is not known to the prover at this point. Remember that $\boldsymbol{\chi}_i$ comes from the finite field $\mathbb{F}_p$. Therefore, the malicious prover has a probability of $1/p$ to find a correct value $\boldsymbol{e}i$ such that the summation is zero. In the other case, where $\mathcal{P}^*$ sends incorrect values $\tilde{a}$ and $\tilde{b}$ using $E_{\tilde{a}}$ respectively $E_{\tilde{b}}$ but a correct value $\boldsymbol{S}$, $\boldsymbol{e}i$ can be chosen to fit to $E_{\tilde{a}}$ or $E_{\tilde{b}}$ and thus the probability is $2/p$.
In the second case, $\mathcal{V}$ receives an incorrect value $\boldsymbol{S}^*$, which hides the incorrect values $\tilde{a}+E_{\tilde{a}}$ and $\tilde{b}+E_{\tilde{b}}$, and which cannot be detected directly. To solve this, Round 6 is required, where the prover must show that he has sent a correct $\boldsymbol{S}$. Based on the equation, discussed in more detail in the description of the protocol, the malicious prover has to fulfil the equation $-E_{\tilde{\boldsymbol{v}}} = \boldsymbol{\eta}^\top \mathcal{C}(E_S) \cdot diag(\boldsymbol{\Delta})$, where $E_{\tilde{\boldsymbol{v}}}$ denotes the error that $\mathcal{P}^*$ adds to $\tilde{\boldsymbol{v}}$ and $E_S$ denotes the error added to $\boldsymbol{S}$. There are two scenarios again, in the first one $\mathcal{P}^*$ sends a correct value for $\tilde{\boldsymbol{v}}$, thus $E_{\tilde{\boldsymbol{v}}} = 0$ and $\boldsymbol{\eta}^\top \mathcal{C}(E_S) \cdot diag(\boldsymbol{\Delta}) = 0$. This happens if $\mathcal{C}(E_S) = 0$, which is only the case if $\boldsymbol{S} = \boldsymbol{S}^*$, or if there are at least two indices $i, j$ such that there are two values inside the commitment that are contrary to each other, more mathematically: $\boldsymbol{\eta}_i \cdot \mathcal{C}(E_{S,i}) = -\boldsymbol{\eta}_j \cdot \mathcal{C}(E_{S,j})$. This results in a soundness error of $|S_\Delta|^{-d_\mathcal{C}}$, since $\mathcal{P}^*$ does not know $\boldsymbol{\eta}$ during this calculation and thus has to find a pair $\mathcal{C}(E_{S,i}), \mathcal{C}(E_{S,j})$ that fits the above equation. Both are embedded in the linear code, thus the probability of finding such a pair is $|S_\Delta|^{-d_\mathcal{C}}$. In the second scenario, the malicious prover sends an incorrect value for $\tilde{\boldsymbol{v}}$, thus $E_{\tilde{\boldsymbol{v}}} \neq 0$. The prover now has to ensure, that the values $\mathcal{C}(E_S) \cdot diag(\boldsymbol{\Delta})$ fit to the corresponding values in $E_{\tilde{\boldsymbol{v}}}$. This happens with probability at most $|S_\Delta|^{-d_\mathcal{C}}$, since each value for $\boldsymbol{\Delta}$ is sampled randomly from $S_\Delta$ and $\mathcal{P}^*$ has to guess $d_\mathcal{C}$ parts of $\boldsymbol{\Delta}$ to ensure the correct embedding.
In summary, the result is a soundness error of $3/p + 2|S_\Delta|^{-d_\mathcal{C}}$, which is still pretty big. Analogous to MPCitH, the checks can be carried out multiple times. So the verifier starts again at Round 4, sends another $\boldsymbol{\Delta}'$ and checks the prover's answers. This results in a negligibly small soundness error.

### 4.2.2 Signatures from VOLEitH

Notice that the above VOLEitH construction is very general, but also has a big soundness error and thus many repetitions are needed to get a reliable ZKP. Since the goal was to achieve small and efficient signatures, Baum et al. present a VOLEitH construction for small fields based on repetition codes, which got a smaller soundness error. The idea is to adapt the Quicksilver protocol, such that an extension field $\mathbb{F}_{p^k}$ can be used to prove the constraints, even when the witness is committed over $\mathbb{F}_p$ [BBdSG+23b]. This helps the authors in the AES use case since AES is defined over an extension field of $\mathbb{F}_2$. For a given repetition parameter $\tau$, the prover receives his values $\boldsymbol{u} \in \mathbb{F}_p^{\ell+r\tau}$ and $\boldsymbol{V} \in \mathbb{F}_q^{(\ell+r\tau)\times\tau}$ from the ideal functionality and commits to his witness by sending $\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{u}_{[1\ldots\ell]}$, for $q = p^r$. After committing, the values $\boldsymbol{u}_i$ and $\boldsymbol{w}_i$ are embedded in the $\mathbb{F}_{q^\tau}$ and all $\boldsymbol{v}_i$ are getting lifted into $\mathbb{F}_{q^\tau}$. This allows all further calculations to be carried out in $\mathbb{F}_{q^\tau} = \mathbb{F}_{p^{r\tau}}$. The Quicksilver protocol now has to be adapted, such that the polynomials $f_i \in \mathbb{F}_{p^k}$ are also embedded into $\overline{f}_i \in \mathbb{F}_{q^\tau}$ and thus the Quicksilver output $c_i$ can be described by:

$$c_i(Y) = \sum_{h\in[0,2]} \overline{f}_{i,h}(\boldsymbol{v}_1 + \boldsymbol{w}_1 \cdot Y, ..., \boldsymbol{v}_\ell + \boldsymbol{w}_\ell \cdot Y) \cdot Y^{2-h}.$$

The authors also point out that $r\tau | k$ must apply so that every lifted and embedded element can be transformed back into an element of $\mathbb{F}_{p^{rk}}$. The verifier now sends $t$ values $\boldsymbol{\chi}_i \in \mathbb{F}_{q^\tau}$ as before and gets the values $\tilde{a}$ and $\tilde{b}$, which were masked by $u^*$ and $v^*$:

$$u^* = \sum_{i\in r\tau} \boldsymbol{u}_i \cdot X^{i-1}, \quad v^* = \sum_{i\in r\tau} \boldsymbol{v}_i \cdot X^{i-1}.$$

To check the corresponding VOLE correlation $v^* = q^* - u^* \cdot \boldsymbol{\Delta}$, the verifier receives the values $\boldsymbol{Q} \in \mathbb{F}_q^{(\ell+r\tau)\times\tau}$ and $\boldsymbol{\Delta} \in \mathbb{F}_q^\tau$. The elements $q_i$ and $\boldsymbol{\Delta}_i$ also need to be shifted in $\mathbb{F}_{q^\tau}$ to fit the values received by the prover. Now he can check the constraint $c = \tilde{a} \cdot \boldsymbol{\Delta} + \tilde{b}$ and thus verify the proof.

Notice, in the generalised subspace VOLEitH framework a code-switching step is necessary to prove the VOLE constraint, which is not the case using repetition codes. Accordingly, no checks are needed to verify that the prover performs honestly during the code-switching. Furthermore, the code lifting soundness error is much smaller in comparison. Assuming that a malicious prover does not have a correct witness $w^*$, there must be an embedded $\overline{f}_i \neq 0$ which still satisfies $\sum_{i\in[t]} \overline{f}_i(w) \cdot \boldsymbol{\chi}_i = 0$. This happens with probability $1/p^{r\tau}$ since $\boldsymbol{\chi}_i \in \mathbb{F}_{q^\tau}$ is sampled after the prover committed to $\overline{f}_i(w)$. Otherwise, the malicious prover can try again to guess the secret state $\boldsymbol{\Delta}$, which he succeeds with probability $2/|\mathcal{S}_\Delta|$. This creates a total soundness error of $1/p^{r\tau} + 2/|\mathcal{S}_\Delta|$.

FAEST, the first VOLEitH-based signature scheme, is built on repetition codes and delivers very short signatures, even for big circuits [BBdSG+23a]. Since there are no Ideal Functionalities in the real world, the authors had to adapt this part and present a compiler to replace all calls to the Ideal Functionality with Vector Commitments. The idea is to commit to random values and open the challenged parts during the verification, such that all constraints can be checked individually. This scheme is insecure if a (malicious) prover can predict the verifier's challenges or learn them from old challenges. This can be prevented by the predetermined structure of the protocol, which carries out the verification steps of the verifier and thus the opening of the commitments at the end of the protocol execution. Since all of the prover's calculations are then completed, he is no longer able to change the values he sent. Finally, there is an interactive zero-knowledge protocol based on VOLE, which has to be made non-interactive using Fiat-Shamir-Transformation. Therefore, the structure for creating signatures from VOLEitH is very similar to the MPCitH-based signatures described previously: Take a VOLEitH protocol and apply the Fiat-Shamir-Transformation to make it non-interactive. However, the additional step of replacing the Ideal Functionalities is required. This can, again, be adapted using different One-Way Functions. In the case of FAEST, AES is used.

# 5 Comparing MPCitH and VOLEitH

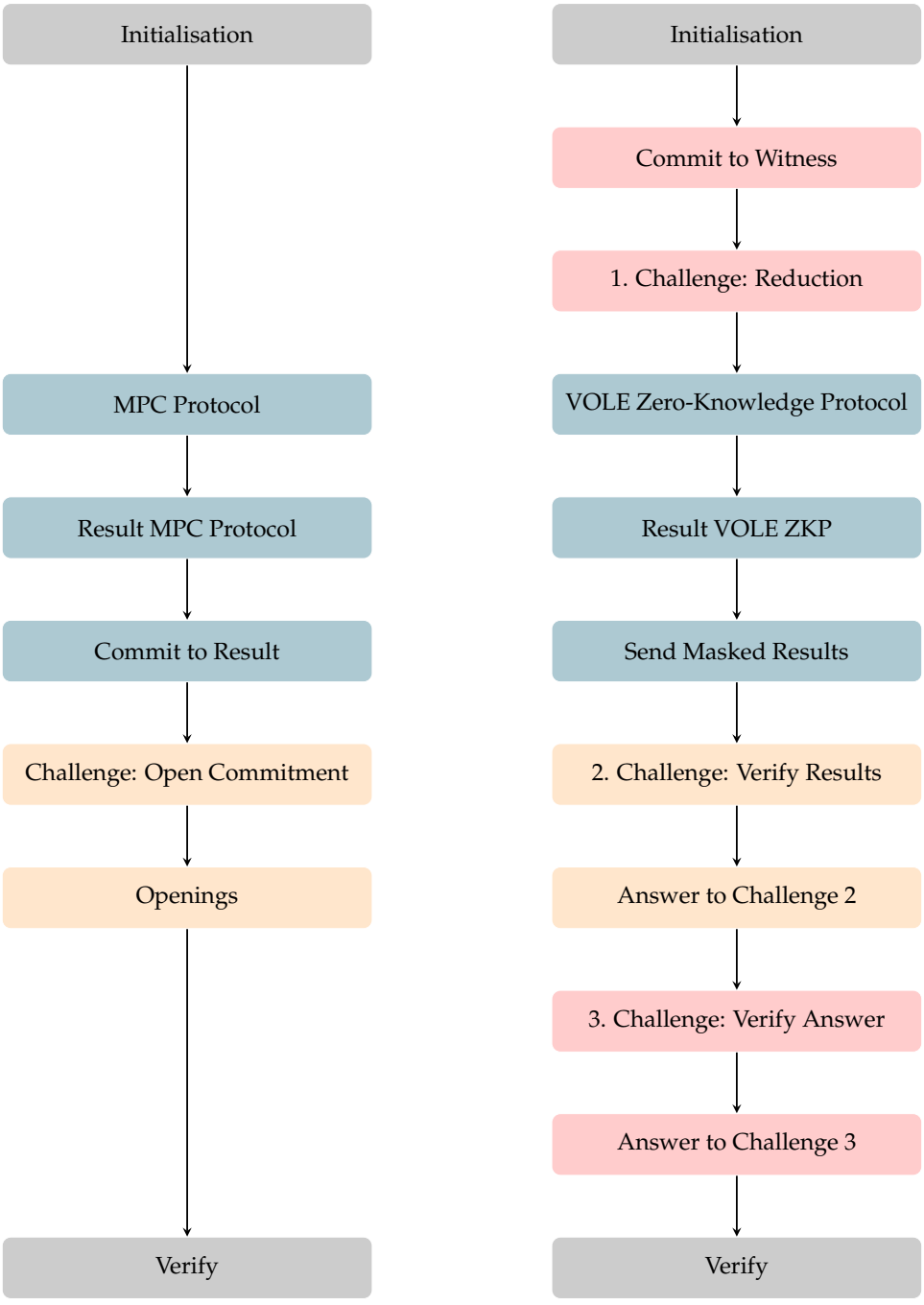
This part of the work deals with the comparison between MPCitH and VOLEitH. To do this, we highlight similarities and differences and present the computational complexity and communication costs of the protocols in more detail. The goal is to find out in which scenarios each protocol excels, which we can then leverage to find a good starting point for optimisations.

## 5.1 MPCitH against VOLEitH

In this section, we compare MPCitH and VOLEitH to show where both protocols have similar approaches and where they differ. We highlight some bottlenecks of both protocols to show the need for further optimisations.

### 5.1.1 Similarities

Since VOLEitH is always described as "just like MPCitH", it stands to reason that there are some similarities between both protocols. We pay particular attention to finding high-level descriptions, even if the underlying structures differ slightly.
The main goal of both protocols is to build practical signatures, which are secure against active (post-quantum) adversaries, and thus aim to improve symmetric cryptography. To realise this, publicly verifiable ZKPs are used. The first major overlap is the structure, which is also shown in Figure 5.1. Both approaches execute an underlying protocol "in the head" of the prover and send the hidden result to the verifier. The verifier then responds with a challenge that forces the prover to verify the result previously sent. In VOLEitH, additional challenges are required to verify that the prover does not cheat when opening the values and to implement further optimisations within the protocol. This results in very small, but also efficient, ZKPs that can easily be verified and later be used to build digital signatures. In addition to the structure of the prover, the structure of the verifier in VOLEitH is also very similar to MPCitH. During the verification, the steps of the prover are carried out by the verifier to check whether the received calculations are correct. This leads to the fact that in practical implementations, such as Banquet or FAEST, the verifier and prover time do not differ greatly from each other.
Both protocols use the same cryptographic primitives to ensure the security and efficiency of the protocols, in particular One-Way Functions and Commitment Schemes. In MPCitH,




(a) MPCitH (b) VOLEitH

Figure 5.1: Comparison of the structure of MPCitH and generalised subspace VOLEitH. The execution of the underlying protocol and the publication of the hidden result are marked in blue, while the opening of the result is marked orange. The additional steps from VOLEitH are shown in red.

commitments are used to commit to the views of each party, before being sent to the verifier for verification. VOLEitH requires a special form of commitment, so-called all-but-one commitments, which are used to exchange the ideal functionality and thus provide a real-world application. Such all-but-one commitments can also be applied to reduce the computational overhead in MPCitH [BBM+24]. This results in faster and more efficient algorithms that require less memory and computing power. OWFs are used in both protocols to build efficient signatures that cannot be broken by any (post-quantum) adversary.

### 5.1.2 Differences

In MPCitH protocols, the prover simulates parties "in the head", so there are several virtual parties which have their share of the witness and calculate a common functionality. In the case of the BGW protocol this would be its common circuit. Each of the parties produce their view, which can be sent to the verifier, who checks whether they are consistent with each other. In VOLEitH the parties are neither holding a share of the witness nor compute some common functionality, instead there is one VOLE correlation that can be checked by a verifier. Instead of opening the views of the individual parties, the prover has a witness and a common set of polynomials, which are evaluated on the whole witness and the results are summarised. The results are then masked and sent to the verifier. The masked values are later part of the verification of the VOLE correlation components but do not directly verify the correctness of the proof. The reason for these differences lies in the different cryptographic approaches of both protocols. MPCitH is based on MPC protocols, which are designed to compute shared functionality with multiple parties. So it must be ensured that during the execution "in the head" of the prover, all calculations are consistent with each other and the secret has been shared correctly. On the other hand, VOLEitH uses VOLEs, which is a special case of OT and is carried out only between two parties. Thus, no sharing needs to be created, and the prover does not simulate any other parties. However, the verifier must be able to check all *and*-gates and thus verify a correct calculation by the prover. To realise this efficiently, polynomials are used, which are recalculated by the verifier, but no virtual parties are needed.
The nature of VOLE protocols is to work over large fields, and thus provide a big set of potential keys that reduce the attackers chances of winning. In VOLEitH, Baum et al. also present a generalised subspace VOLE protocol which is public coin verifiable and works over any finite field, even if it is exponentially large. To improve the soundness error and get smaller proofs, FAEST uses a special VOLEitH protocol for small fields, which is also the approach of MPCitH protocols to keep calculations simple and proofs small. However, VOLEitH provides the possibility of working on arbitrarily large, finite fields, which is not necessarily the case with MPCitH.

### 5.1.3 Bottleneck of MPCitH

To get a better sense of limitations and optimisations on MPCitH, this subsection will highlight the main bottlenecks and thus provide a basic overview before we present our optimisations in Chapter 6.

**Scalability.** The biggest limitation of Multi-Party Computation in the Head protocols is the scalability since the proof size increases linearly with the size of the circuit, and thus MPCitH is not efficient for big circuits [BBdSG+23b]. As discussed before, [AHIV17] tries to solve this problem but has immense overhead due to the Reed Solomon calculations and is only useful for very large circuits. So there is no general proof system based on Multi-Party Computation that realises efficient Zero-Knowledge Proofs for any circuit size. In 2024, [LJWJ24] provided a high-performance hardware implementation for Picnic3 and thus demonstrated how to enhance the scalability of MPCitH protocols. Their main inside was that the prover has to simulate a lot of parties to get a reliable bound for the soundness error without having to repeat the proof arbitrarily often. This can lead to shorter proof sizes but also increases the computational complexity compared to current MPCitH approaches.

**Exchanging Building Blocks.** In the last years of research, it turned out, that MPCitH protocols can be easily adapted by exchanging the underlying MPC protocol. This enables the use of a wide variety of protocols and increasingly efficient and smaller proofs, but comes with a variety of new challenges. One difficulty is the underlying cryptographic problem. Protocols, like BBQ or Banquet, are built on AES and thus use symmetric primitives. Others like Picnic are using more MPC-friendly primitives, which means that the underlying structure performs many local operations and requires little communication between parties. But there are also protocols building up on hard problems like syndrome decoding [Fen22]. All deliver a different degree of security, run time and proof size. Depending on the context, they can all be used equally well, so there is no optimal underlying problem. This also makes it more difficult to implement such protocols, as they are not simply exchanged in blocks, as one might expect, but further optimisations are made to make the specific protocol faster and more efficient. This also makes it difficult to compare the protocols with each other and to use them in real-world applications. Therefore, for every application of MPCitH protocols, it must be checked beforehand whether the requirements are sufficient. Thus, MPCitH cannot be used in a generalised manner.

**Costs.** Most of today's MPCitH protocols are optimised for either efficiency or proof size. Efficient protocols aim to reduce the computational complexity or the runtime, while approaches that optimise the proof size may use a higher amount of parties or more rounds, trading runtime for a smaller proof size [BKPV24]. This allows one to use different parameters to optimise the protocol for a specific problem but also makes the generalisability of MPCitH protocols and their application in new subject areas more difficult. Another significant trade-off is between the performance and security of the protocol. A higher level of security often requires more complex or additional computations, for example, additional checks or challenges, and more communication between the participants in the protocol [AMGH+23]. Trying to reduce the communication between parties often results in smaller proofs and more efficient computation times, but can also lead to a higher computational overhead on individual parties. This can be a problem if individual parties have less computing capacity.
An example of this is the use of Vector Commitments. In [BBM+24] the authors show an improvement of all but one Vector Commitments by using batching and applying it to MPCitH and VOLEitH. This reduces the size of the elements sent during the opening but also reduces the security within the protocol. This is due to the use of rejection sampling, which reduces the entropy of the challenge space. The scheme remains secure because the attacker would have to perform the same steps as the prover and would also have to guess the call of a hash function, but it is more vulnerable than before.

**Side-Channel Vulnerability.** Based on the high popularity of MPCitH protocols and the high theoretical security, people try to exploit vulnerabilities in practical implementations, for example by applying side-channel attacks [GSE21]. In [SBWE20], the authors show how vulnerable MPCitH-based protocols are against side channels and that a minimal leakage suffices to break the security of the whole protocol. During the standardisation by the NIST, while many of the submitted protocols already provide protection against side-channel attacks, there were also some unprotected approaches that, while secure in theory, can reveal errors if implemented incorrectly or used in practice [AAC+22]. Therefore, for each application, it must be checked again whether the implemented MPCitH protocol is secure and can be used, as they are not inherently secure in practice. Countermeasures against side-channel attacks include constant-time implementations and masking schemes. These techniques have been applied to protect the Picnic3 signature [ABE+21]. While these countermeasures require additional computations and reduce performance, they enhance the security of the protocol.

### 5.1.4 Bottleneck of VOLEitH

We also want to take a closer look at the limitations of VOLEitH. As VOLEitH is a comparatively new concept, research into its structure and composition has been limited so far. As such, it stands to reason that the following listed weak points and problems may not be exhaustive.

**Public Verifiability.** To realise signatures, publicly verifiable protocols are necessary. As discussed before, most protocols that are based on VOLE are designated verifier proofs and thus not publicly verifiable. To solve that, [BBdSG+23b] presented VOLEitH which uses the code-switching step and additional challenges. This results in additional overhead and more complex calculations, which can be optimised for specific applications, like FAEST, but can not be applied in general. Another example of less generalisability is the adaptation of the protocol to Quicksilver since the ZKP and most of the challenges are adjustments to fit the VOLEitH scenario.

**Costs.** Building one VOLE instance is very expensive. Producing random values in $\mathbb{F}_2$ and lifting them into a larger field $\mathbb{F}_{2^k}$, which is needed for FAEST, forces the verifier to perform $O(2^k)$ calculations. For a given security parameter $\lambda$, this verification is infeasible for a single instance using $k = \lambda$ [BBdSG+23a]. To solve this issue, the authors run several VOLE instances in parallel over small fields, such that all calculations can be done using a polynomial amount of work. For a given repetition parameter $\tau \in \mathbb{N}$, $k_0 = \lceil \lambda/\tau \rceil$ and $k_1 = \lfloor \lambda/\tau \rfloor$ are chosen as the small-field size parameter. Based on this, two other repetition parameters $\tau_0 = \lambda \bmod \tau$ and $\tau_1 = \tau - \tau_0$ are selected such that $k_0 \cdot \tau_0 + k_1 \cdot \tau_1 = \lambda$. This ensures that the concatenation of results together forms a valid VOLE correlation over the $\mathbb{F}_{2^\lambda}$ and results in a trade-off between signature size and speed. Decreasing $\tau$ results in larger fields and thus in smaller proofs, but more work for the prover and verifier.
Another problem is the high communication overhead. In the general subspace VOLE version of the protocol, there are several interactions between the prover and the verifier where huge vectors and matrices are sent. This overhead scale is linear with the amount of needed VOLE correlations. Thus, a lot of optimisations are included to reduce this overhead and shorten the proof size, but the overhead is not to be neglected. This could also be a problem for the scalability. This is discussed in more detail in Section 5.2.

**Vulnerabilities.** An inherent problem of VOLE protocols are selective fault attacks [Sch22]. For a given VOLE correlation $y = a \cdot x + b$, a malicious verifier can introduce an error, namely $y' = y + (a' - a) \cdot x$, and thus learn if a specific bit of $x$ equals zero. Such attacks allow the adversary to introduce errors depending on the secret inputs of

an honest prover, even if the VOLE correlation is correct and thus leak parts of the secret input. Furthermore, errors, that are introduced during the evaluation could propagate through the whole protocol and lead to incorrect outputs.

## 5.2 Communication Costs

The communication between the prover and verifier is an essential part of analysing the efficiency of zero-knowledge protocols. Therefore, many approaches try to reduce communication and compress the data to be sent.

### 5.2.1 Communication of MPCitH

Communication is one of the most limiting factors in MPCitH protocols since in some approaches the costs grow linearly with circuit size ([IKOS07], [GMO16]). There are some approaches, like [AHIV17] and [DdSGOT21], that reduce this dependency and use less communication. This leads to more complex calculations. To address this problem, Giacomelli et al. presented the ZKBoo protocol that provides efficient proofs for boolean circuits while reducing circuit complexity [GMO16]. For this, circuit decomposition is used as the underlying MPC protocol. The approach is to break large circuits into smaller subcircuits, which can be calculated more easily. Since circuit decomposition only needs to satisfy two-privacy and correctness, which are lower assumptions than traditional MPC protocols, this reduces overhead. This protocol continues to build on the MPCitH paradigm, but the number of parties is set to three, which leads to further savings in communication. The successor protocol, [CDG$^+$17], further improves this approach, but remains with a maximum of three parties. For the comparison of MPCitH and VOLEitH, we want to use [KKW18] as a representative for MPCitH protocols. We are particularly interested in the structure of the protocol, the required communication and the computational complexity.

**Sketch [KKW18].** One of the most optimised protocols using the MPCitH paradigm was presented by [KKW18][2] and creates efficient Zero-Knowledge Proofs for boolean circuits based on symmetric key cryptography. The protocol can be seen in Figure 5.2. In contrast to [CDG$^+$17], this protocol achieves signatures that are smaller and can be expanded to include any number of parties.
At the beginning of [KKW18], the prover and verifier get an arithmetic circuit $C$ as a statement. The prover now wants to show that he has a witness $w$ such that $C(w) = 1$. To do this, he carries out an offline phase in which he chooses seeds for all $N$ parties and

[2] We are referring to the e-print archive version (`https://eprint.iacr.org/2018/475.pdf`), which is updated and contains bug fixes.

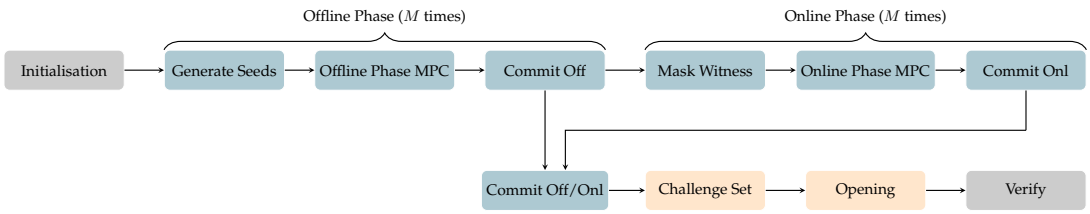



Figure 5.2: Sketch of the [KKW18] protocol. The blue steps are the execution of the MPC protocol and the commitments, while the orange part indicates the challenge and the corresponding openings.

commits to them by applying a cryptographic hash function. The prover then masks the witness using random shares and performs the online phase of the protocol. He simulates an $N$-party protocol using the masked witness, the seeds and also commits to the results. These steps are repeated $M$ times. $\mathcal{P}$ then hashes all commitments from the offline and online phases, which are hashed again and sent to the verifier. $\mathcal{V}$ chooses a $\tau$-sized challenge set, one challenge party for each challenge and sends it to the prover. $\mathcal{P}$ opens the seed and commitment of the online phase for all runs that are not part of the challenge. For all challenge runs $\mathcal{P}$ sends the commitments, the states, the masked witnesses and the associated output of the MPC protocol which do not contain the elements of the challenge party, except for his commitment. To verify the proof, $\mathcal{V}$ simulates the offline and online phases of the protocol and checks whether the output matches the hash sent by the prover. The general structure of the MPCitH protocols discussed so far can also be seen here. Prover and verifier have a common circuit, the prover calculates the MPC protocol for $N$ virtual parties "in the head" and sends the (hashed) commitments to $\mathcal{V}$. The verifier then sends challenges, the prover opens the commitments and $\mathcal{V}$ can check whether the results are consistent with each other after simulating the prover's steps.

**Communication of [KKW18].** Based on these steps, we now take a closer look at the communication costs. There are two crucial steps of the protocol that require communication between the prover and verifier, sending the commitments after completing the MPC protocol and publishing the openings based on the verifier's challenges.
After the prover has simulated the MPC protocol "in his head", he calculates the hashes for the online and offline phases. In order not to transfer all the values individually, he hashes them again. Although this leads to an additional step in the calculation of the proof and thus later in the verification, it also means that only one hash value has to be transmitted. For a given security parameter $\kappa$, the authors use a hash function with output length $2\kappa$. The communication to send this hash is $2\kappa$.
The openings in this protocol are divided into two parts, the openings for the challenge

set, which was sent by the verifier, and the openings for all elements that are not part of the challenge set. This is a special feature of this protocol, as previous MPCitH approaches only send answers related to the challenges to the verifier. This is due to the cut-and-choose technique the protocol uses. The idea is that a malicious prover can cheat in the preprocessing phase, which an honest verifier cannot detect. To prevent that, the prover opens several of its preprocessing runs to show that everything was calculated correctly. Since no witness is involved in the preprocessing phase, the security is guaranteed. The openings to the challenge set are still to check the correctness of the proof and the witness. In the first part, the prover opens the seeds and commitments for all runs that are not part of the $\tau$-sized challenge set. So the prover has to open all-but-$\tau$ of its $M$ runs. To realise this efficiently, [KKW18] uses Merkle Trees. The idea is to expand a given seed again and again using a hash function to create a binary tree. To publish elements, it is enough to publish the seed, as the rest can be calculated directly from the seed. This ensures that to publish all-but-$\tau$ out of $M$ values, exactly $\kappa \cdot \tau \cdot \log(\frac{M}{\tau})$ communication is needed. In addition to the seeds, the commitments from the online phase are also sent, which results in an additional overhead of $2\kappa$ per commitment. This means that $3\kappa \cdot \tau \cdot \log(\frac{M}{\tau})$ communication costs are required for this part of the openings.

The prover opens the $\tau$ challenge runs and sends all the values to the verifier. First $\mathcal{P}$ sends the states and the randomness used in the hashes. Based on the structure of the Merkle Trees and the all-but-one openings, this requires $\kappa \cdot \log(n)$ amount of communication. This is needed so that the verifier can independently calculate the commitments to show the correctness of the prover's calculations. Since the state of party $n$ also contains the result of the offline phase of the MPC protocol, an additional overhead is required. The result has exactly the size of the used and-gates of the circuit, denoted by $|C|$, and requires exactly $|C|$ communication. Equivalent to this, the prover publishes the results of the online phase, which have the same size and therefore also require $|C|$ of effort. $\mathcal{P}$ publishes the masked witness, which can be used later to allow the verifier to simulate the execution of the MPC protocol. Since the witness itself needs to be protected again, it is masked beforehand, but since the prover also carries out the calculations on the masked witness, the same proof is created. Masking only hides the witness but does not expand it, so $|w|$ elements must be sent. Finally, the prover publishes the randomness used for the commitments and the commitments themselves. Since the randomness strings are of length $\kappa$ and the hashes of length $2\kappa$, this step requires $3\kappa$ of communication. This means that this second opening requires $(\kappa \cdot \log(n) + 2|C| + |w| + 3\kappa)$ of communication.

This results in the following overall complexity for communication:

$$\text{commCost} = 2\kappa + 3\kappa \cdot \tau \cdot \log(M/\tau) + \tau \cdot (\kappa \cdot \log(n) + 2|C| + |w| + 3\kappa) \; .$$

### 5.2.2 Communication of VOLEitH

VOLE correlations can be used to provide very efficient and fast proofs. Protocols like Wolverine [WYKW21] or Quicksilver [YSWW21] achieve communication costs of one field element per multiplication gate and at the same time can be calculated mathematically efficiently [BBdSG+23b]. These protocols allow further optimisations but have the problem that they are still designated verifier proofs and thus cannot be used for signatures. [BBdSG+23b] solve this problem, but only at the detriment of an increase in costs, both in the calculation of the proofs and in the communication between the prover and verifier. As discussed before, many optimisations have already been incorporated to reduce the communication effort.

**Communication Generalised Subspace VOLE.** The costs of the VOLEitH protocol are made up of two large parts, the costs to construct a VOLE correlation and the round-specific costs which are used to perform the ZKP. The first part of constructing the VOLE correlation involves creating and distributing the secret values for the prover ($\boldsymbol{U}$ and $\boldsymbol{V}$) and verifier ($\boldsymbol{Q}$ and $\Delta$). In the following, these costs are simply referred to as sVole.
In the first round of the VOLE protocol, the prover has to commit to the witness at the beginning of the protocol. This step is needed to build a VOLE correlation that confirms the correctness of the witness without revealing it to the verifier. Since $\boldsymbol{W}$ rows consist of all elements from $(\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_\ell) \in (\mathbb{F}_p^{k_\mathcal{C}})^\ell$ and is masked by $\boldsymbol{U}_{1,...,\ell}$, the prover must send a matrix of size $\ell \times k_\mathcal{C}$ to the verifier. To send an element of $\mathbb{F}_p$, $log_2(p)$ bits are required. This step is specific to the VOLEitH protocol and requires $\ell \cdot k_\mathcal{C} \cdot log_2(p)$ communication.
After calculating the Quicksilver constraints, the prover sends the masked coefficients. In general, the prover evaluates $t$ polynomials and has to send the coefficients of all $t$ polynomials to the verifier for checking. However, this would mean that communication is directly dependent on the size of the circuit since $t$ describes the number of multiplications. To prevent this, the optimisation presented in [YSWW21] was introduced via an additional challenge $\boldsymbol{\chi}$, as shown in Section 4.2. The idea was to use the characteristic of degree-separated polynomials and to aggregate all coefficients. This resulted in $t$ coefficients being represented by the aggregated coefficients for each degree, namely $\tilde{a}$ and $\tilde{b}$. The challenge $\boldsymbol{\chi}$ ensures that the prover sends the correct values, while the respective coefficients are still masked using the rows of $\boldsymbol{U_1}$ and $\boldsymbol{R}$. This means that the prover has to send two elements of the $\mathbb{F}_p^{k_\mathcal{C}}$ and requires $2 \cdot k_\mathcal{C} \cdot log_2(p)$ communication effort for this.
To check the coefficients, the verifier needs the values of $\boldsymbol{U}_1$ and $\boldsymbol{R}$, which where used for masking. Since these must remain secret to ensure the secrecy of the witness, the prover sends the value $\boldsymbol{S}$. To do this, $\mathcal{P}$ uses the provided challenge $\Delta'$, which forces him to send the correct values. While analysing the protocol and communication, we noticed

that $\boldsymbol{S}$ is not an element of $\mathbb{F}_p^{(\ell+1)\times n_\mathcal{C}}$ as described in the paper, but should be an element of $\mathbb{F}_p^{(\ell+1)\times k_\mathcal{C}}$. This can be deduced from the structure of $\boldsymbol{U}_1, \boldsymbol{R} \in \mathbb{F}_p^{(\ell+1)\times k_\mathcal{C}}$. This requires the prover to send $(\ell+1)\cdot k_\mathcal{C} \cdot log_2(p)$ elements to the verifier.
In the last step of the protocol, $\mathcal{P}$ sends the second part of his VOLE correlation, the values of $\boldsymbol{V}$. These are later needed by the verifier to check the correctness of the correlation and thus verify the proof. As $\boldsymbol{V}$ is a big matrix, which consists of $(\ell+1)\times n_\mathcal{C}$ elements, this would lead to a big communication overhead. To prevent this, [BBdSG+23b] presents the approach that the verifier sends a challenge $\eta$. The idea is that $\eta \in \mathbb{F}_p^{(\ell+1)}$ is multiplied by the message that the prover wants to send and so instead of an $(\ell+1)\times n_\mathcal{C}$ matrix, only a vector of size $n_\mathcal{C}$ has to be sent. This costs the prover $n_\mathcal{C}\cdot log_2(p)$ communication. The verifier can also multiply the challenge $\eta$ on its part of the VOLE correlation and has less effort to check the values obtained from $\mathcal{P}$.
Viewed over the complete protocol for generalised subspace VOLEs using a linear code, the communication costs can be summarised as:

$$\text{commCost} = \text{sVole} + (2\ell+2)\cdot k_\mathcal{C}\cdot log_2(p) + n_\mathcal{C}\cdot log_2(p)\ .$$

### 5.2.3 Comparison to MPCitH

Based on the elaboration by Baum et al. VOLEitH should provide much more efficient proofs than the previous MPCitH approaches. To examine this in more detail, we use the comparison of both protocols from Figure 5.1 and use our analysis of the communication effort to show where VOLEitH is better or even worse than conventional MPCitH methods. As before, we use [KKW18] as a representative for MPCitH protocols. An overview of this is shown in Table 5.3. For the sake of completeness, the verifier communication is also shown. This is not important for further analysis, because by using the Fiat-Shamir transform to create signatures, the prover calculates everything and sends it to the verifier, but the verifier itself never interacts with the prover.
At the beginning both protocols have a preprocessing phase, in [KKW18] it is the offline phase, while [BBdSG+23b] carries out the interaction with the ideal functionality and the division of the matrices. Both are independent of the interaction with the verifier and do not require any communication. A special feature of VOLEitH is that the values for the VOLE correlation must be created and distributed to the respective party. This requires a lot of calculations and communication since elements have to be sent sublinearly in the size of the witness. Another additional step is that the prover masks the witness and sends it to the verifier. This allows the verifier to later check the correctness of the calculation and thus check the proof. MPCitH does not need such a step, since later arguments are always made about the consistency of the views of the individual parties and therefore

| MPCitH | | VOLEitH | |
|---|---|---|---|
| Steps | Communication | Steps | Communication |
| Initialisation | ——— | Initialisation | sVole |
| ——— | ——— | Commit to Witness | $\ell \cdot k_{\mathcal{C}} \cdot \log_2(p)$ |
| ——— | ——— | 1. Challenge | $t \cdot \log_2(p)$ |
| MPC Protocol | ——— | VOLE ZKP | ——— |
| Result | ——— | Result | ——— |
| Commit | $2\kappa$ | Send Masked Result | $2 \cdot k_{\mathcal{C}} \cdot \log_2(p)$ |
| Challenge | ——— | 2. Challenge | $\log_2(p)$ |
| Openings | $3\kappa \cdot \tau \cdot \log(M/\tau) + \tau \cdot (\kappa \cdot \log(n) + 2\lvert C\rvert + \lvert w\rvert + 3\kappa)$ | Answer | $(\ell+1) \cdot k_{\mathcal{C}} \cdot \log_2(p)$ |
| ——— | ——— | 3. Challenge | $(\ell+1) \cdot \log_2(p)$ |
| ——— | ——— | Answer | $n_{\mathcal{C}} \cdot \log_2(p)$ |
| Verify | ——— | Verify | ——— |

Table 5.3: Communication effort based on the steps performed in the protocols. The verifier's communication is shown in colour and can be ignored after applying the Fiat-Shamir Transformation.

discrepancies arise in the proof. The results of the preprocessing phase of [KKW18] will also be partially opened later in the protocol as part of the openings to ensure that the prover did not cheat during this phase. So the communication effort is only postponed until a later point in the protocol and thus makes the openings more expensive, but at the same time does not require an exchange between the prover and verifier before the actual execution of the protocol.

In the next step, both protocols execute the respective steps of the prover to create a valid proof for the given witness. Both require no communication between $\mathcal{P}$ and $\mathcal{V}$ since the prover executes the protocol "in the head". To publish the results in the MPCitH scenario, a commitment is sent from the prover to the verifier. In the specific example of using [KKW18], this is a hash that contains all information about the offline and online phases. The authors assume that the hash function for a given security parameter $\kappa$ has an output size of $2\kappa$. The size of the security parameter depends entirely on the application and is usually specified as 128 or 256-bit security for AES. Since the use of such hash functions is very expensive, the optimisation has already been applied to send only one hash as the

result of the calculation, which in the end increases the calculation time in the verification, but enables a more efficient transmission with less communication. Calculating the masked result in the VOLEitH scenario is much cheaper, as it only depends on the quality of the code and the finite field. Every calculation is defined in an extended field above the $\mathbb{F}_2$. Thus, only one bit of communication is required per field element. Since the coefficients of the polynomials are the encodings within the linear code, a code with a lower dimension $k_{\mathcal{C}}$ allows for much more efficient encoding and fewer values to be transmitted. The degree-separated polynomials make it possible to compress the results to such an extent that only the aggregated coefficients have to be sent. The focus is on degree two relations, i.e. polynomials with a degree of at most two. Therefore, only two elements are needed for the aggregated coefficients so that the verifier can check the calculations. This framework can be extended for arbitrary degree $d$ relations. So VOLEitH is much more efficient in compressing the results and requires less communication than [KKW18].

Next we take a look at the opening step, which is also the most expensive part of the protocol. We need to open the provers values to the verifier as the correctness of the proof is built upon the verifiers simulation of the provers calculations. It has to be ensured that the secrets of the respective parties remain protected, which is achieved through masking, and that the verifier is still able to expose a malicious prover. In VOLEitH, $\mathcal{P}$ simply sends the matrix $S$, which the verifier can use to check the previously sent aggregated coefficients. The advantage of this is that multiple instances of the protocol, in particular the different views of the parties, do not have to be published and in addition an efficient representation via the finite elements of the field $\mathbb{F}_p$ is possible. Furthermore, this matrix offers the possibility to check the calculations of all polynomials without requiring any further effort. The big difference between this step and the MPCitH method is that the verifier does not request a subset of the previously calculated values, but rather receives the concrete calculations for all polynomials. This is because a verifier cannot deduce the secret of the prover even with all the values of the VOLE correlation and a (malicious) prover cannot change them in previous phases without being noticed. This means there is no multi-stage opening, as is the case with [KKW18]. The difficulty in this protocol is that a (malicious) prover can falsify the values in the preprocessing phase and the verifier must be able to detect such modifications. For this, the cut-and-choose technique is used, in which $\mathcal{P}$ precalculates many preprocessing phases and the online phase is based on these. The verifier selects $\tau$ elements of the online phase that are used for the correctness of the proof and all other preprocessing phases serve to show that the prover has calculated everything correctly. This means that a lot of hashes are sent again in the [KKW18] protocol, which are much more expensive compared to elements of the $\mathbb{F}_p$. Thus the openings with VOLEitH are significantly cheaper than with MPCitH since the verifier only needs the masked val-

ues to evaluate the polynomials and can detect errors directly in the VOLE correlation. With [KKW18] this is much more complicated since a (malicious) prover can cheat in the preprocessing phase and this has to be found out separately by the verifier. The cut-and-choose technique is a good approach but requires a lot of communication. As is usual with most MPCitH protocols, there is again a linear dependency between circuit size, witness length and overall communication, which also makes this approach particularly suitable for small to medium-sized circuits. But [BBdSG+23b] also try to be efficient especially for small to medium-sized circuits, which is easily feasible for AES, and the previous analysis also shows that VOLEitH requires more expensive and longer calculations for larger circuits. How this directly affects communication and the size of the proof is not discussed in the paper and should be investigated further. The poor scalability in both protocols results from the direct dependency on the size of the circuits. In MPCitH, communication is required when calculating the binary multiplications to ensure the correct degrees and calculations. In the generalised subspace VOLEitH approach, there are as many polynomials multiplications. Therefore both protocols got the same dependencies within the circuit. To reduce the soundness error, both approaches use the repeated execution of the protocol and the associated openings based on new randomness and new challenges. This means that the already very high costs in the MPCitH scenario continue to rise, while the cheaper variant of VOLEitH is better designed for multiple rounds.
In the last part of the protocol, VOLEitH checks the previously sent elements. This requires an extra challenge, i.e. $\eta$, which forces the prover to send the second part of his VOLE secret masked. This is needed to ensure that the previous elements are codewords of the linear code and that a malicious prover has not chosen incorrect elements. Optimisations have also been added here to ensure that only a vector of length $n_{\mathcal{C}}$ has to be sent and not a matrix. This means that the communication effort is directly related to the quality of the code, in this case, the length of the code words. The resulting overhead can only be attributed to checking the correctness of the encodings and is not required in comparable MPCitH approaches.
In summary, the MPCitH approach of [KKW18] requires less communication before and after the prover executes the proof and sends the associated openings. VOLEitH provides better data structures for sending data and can therefore publish the results of the proof and the associated elements needed for verification with less communication. Furthermore, many optimisations were applied to VOLEitH to reduce the communication effort. These ensure faster calculations and less communication effort, but also a lower adaptability of the protocol.

**Repetition Code VOLE.** The VOLE construction over small fields used in FAEST requires far less communication than the generalised VOLE approach described above. One reason for this is the use of Repetition Codes, as this means that fewer challenges and answers have to be sent to ensure that all encodings are correct. Furthermore, different from MPCitH, the classic "opening step" in in which the prover sends all used elements to the verifier no longer exists. These are advantages that arise from the type of proof, especially through the VOLE correlation. The prover only has to commit the witness and then send his part of the VOLE correlation to the verifier. This allows the verifier to check the correctness of the proof and verify that the witness is correct. Therefore, the necessary communication is reduced to the first challenge from the verifier, which is no longer necessary after applying the Fiat Shamir transformation, and the masked result from the prover. Since these are now elements of $\mathbb{F}_{q^\tau}$, they require more communication effort than the previous elements of $\mathbb{F}_p$, but can still be implemented efficiently. Any remaining communication from the previous, generalised version of the protocol is no longer needed. Notice that we do not present this approach in Table 5.3 as we focus on the generalised VOLEitH approach due to its proximity to MPCitH.

## 5.3 Computational Complexity

In this part of the work, we deal with the computational complexity of the prover and verifier in MPCitH and VOLEitH. The focus is on a high-level description, as the effort can be significantly affected by parameterisation and the use of different algorithms.

### 5.3.1 Complexity of MPCitH

For a precise examination of the computational complexity of MPCitH, we again refer to [KKW18]. A brief overview of the protocol itself is given in Section 5.2.1.

**Prover.** The big difference between [KKW18] and conventional MPCitH protocols was the division into a preprocessing phase and the associated online phase. The idea is that the prover does a lot of calculations in advance and thereby reduces the effort for the verifier during the verification. Since a malicious prover can cheat in this step, it must be ensured that all values have been calculated correctly. This step requires a lot of effort, which can be reduced by using Merkle Trees and hash functions. Below we briefly recap the computational complexity of both phases.
In the first step of the preprocessing phase, the prover creates the seeds that will be used to later create the Merkle Trees. To do this he uses an algorithm to create some randomness. The advantage of seeds is that they can be expanded by using a hash functions and by that

less randomness has to be used in the protocol. This simultaneously leads to more efficient calculations and easier openings. Since a single Merkle Tree is created for each of the $n$ simulated parties, $n$ seeds are required. Based on the seeds, $\mathcal{P}$ now creates the Merkle Trees, calculates the offline phase of the protocol and commits to the results. The offline phase only consists of calculating the *aux*, which is independent of the witness and allows the verifier to check the preprocessing. Implementing these calculations is comparatively easy. The prover calculates the online phase of the protocol by simulating all $n$ parties with the masked inputs. The input must be masked to ensure the zero-knowledge property of the witness. These steps are executed $M$ times. To keep the communication effort low, the prover now calculates the hashes of the online and offline phase for each of the $n$ parties, combines these into a single hash for the online and offline phase and sends the combined hash to the verifier. This of course leads to additional calculations within the proof and also later in the verification, but significantly reduces the communication effort. This is a common trade-off in MPCitH and other protocols.

**Verifier.** The limiting factor with MPCitH protocols is that the verifier simulates the steps of the prover to ensure that he has calculated everything correctly. This means that prover and verifier complexity behave very similarly to each other. Based on the cut-and-choose technique and the $\tau$ sized challenge set, the verifier first recalculates the commitments received from the offline phase. Since $\mathcal{V}$ only received hashes as a result of the proof, he must also hash the calculated commitments. This allows him to verify that the correct values were used in the online phase, but does not ensure that a (malicious) prover has not cheated. In the next step, he calculates the Merkle Trees based on the seeds sent for all runs that are not part of the challenge set and determines the associated hashes. This allows the verifier to ensure that the prover has carried out the offline phase correctly and at the same time ensures that the preprocessing phase of the elements in the challenge set remains secret. meaning the verifier does not receive any additional information. By calculating the commitments in the first step of verification and the openings of the prover, the verifier now has all the values that the verifier used together with the masked witness in the online phase and can now simulate this phase for each run of the challenge set. He calculates all hashes again and combines them, as in the last step of the prover.
In summary, the division into two phases requires many additional steps, which increases the time required for the prover and verifier. At the same time, these steps also enable the use of an MPC protocol, which is simple and quick to calculate. This is an advantage over conventional MPCitH approaches and ensures very efficient signatures.

### 5.3.2 Complexity of VOLEitH

Based on the analysis of [KKW18], we now introduce the computational complexity of VOLEitH. We will first look at the generalised approach of VOLEitH and its comparison with the MPCitH approach presented above.

**Prover.** At the beginning of the protocol, the VOLE correlation must be created. This is done by executing the Ideal Functionality. However, such functionality does not exist in real-world applications, which is why Baum et al. use the compiler discussed in Section 4.2.2. This means that all requests for the ideal functionality are implemented through Vector Commitments, which are calculated by the prover. The next step is the prover's commitment to the witness by sending it masked to the verifier. This is needed to ensure the zero-knowledge property and to provide the verifier with the masked witness for later evaluation of the proof. Since the witness is represented as a matrix, $\mathcal{P}$ requires the subtraction of two matrices, which is linear in the size of these matrices. To calculate the proof, the prover must evaluate the $t$ polynomials over the masked witness. To do this, it calculates the respective coefficients of the polynomials on the masked witness, sums these up and sends them to the verifier. In the two remaining challenges that $\mathcal{P}$ receives, he must again perform linear size operations on matrices to show the correctness of his proof. After applying the Fiat-Shamir transformation, all of the verifier's challenges are now calculated by the prover. To do this, $\mathcal{P}$ hashes all the messages that he would send to the verifier and uses this hash as a challenge. This means that communication between prover and verifier is no longer required and the verifier is still able to check the correctness of the proof. The protocol is therefore non-interactive and can be used as a digital signature scheme.

**Verifier.** During the execution of the generalised Zero-Knowledge Proof for VOLEitH, the verifier must send three challenges to reduce the communication overhead as well as a commitment element for the prover. For this purpose, random vectors are drawn. These are no longer needed after applying the Fiat-Shamir Transformation, as the prover then selects all challenges and sends them to the verifier. In the first step of the verification, $\mathcal{V}$ calculates the random masking of the witness, which was used by the prover in the proof. This allows the polynomial to be evaluated at the same points and thereby prove the correctness of the proof without publishing the witness itself. The verifier also calculates the coefficients for all $t$ polynomials on the masked witness, sums these up and compares them with the elements that he received from the prover. Finally, the verifier checks the correctness of the VOLE correlation. For this, he needs his two elements of the VOLE correlation and the elements obtained from the prover through the challenges. Notice

that the verifier has no costs when creating the VOLE correlation, as the prover takes care of this completely. The prover is forced by the Vector Commitment to use correct values, otherwise the verifier will notice inconsistencies.

### 5.3.3 Comparison to MPCitH

After analysing the computational complexity of both protocols, we now compare both protocols with each other, analogous to Section 5.2. We focus on the bottlenecks of both protocols. The comparison is also shown in Table 5.4.
The first noticeable thing is the distribution of calculations across the protocol. In VOLEitH, the prover creates the VOLE correlation at the beginning of the protocol. To do this, he has to form random matrices, create a valid VOLE correlation and create a Vector Commitment using GGM Trees. This step is specific to VOLEitH, as checking the correctness of the proof is shown at the end via the VOLE correlation. MPCitH does not require this as the consistency of the proof is shown later across the openings of each party. Thus, the prover does not require any additional overhead in this step. Notice that randomness is still generated later in the protocol and the associated Merkle Trees are created. This step is also carried out $M$ times and is therefore part of the MPC protocol. This only postpones the effort and does not eliminate it. The same applies in the subsequent step of VOLEitH, the commitment to the witness. This step is required so that the verifier can evaluate the polynomials using the masked witness. The special thing here is that the original commitment with the elements of the matrix $\boldsymbol{U}_1$ is removed in further calculations and a new masking is carried out using the elements of $\boldsymbol{R}$. This allows the verifier to create the correct VOLE correlation at the end using $\boldsymbol{R}$ and $\boldsymbol{U}_1$ and the verifier can correctly calculate $\boldsymbol{w} \cdot \Delta'$ without having to unmask $\boldsymbol{w}$ itself. This means that the zero-knowledge property remains fulfilled. Also in [KKW18] the masked witness is published by the prover so that the verifier can carry out the later calculations on it. However, only the values for the challenge set are published, otherwise, a (malicious) verifier would be able to determine the individual values of the witness based on the values of the preprocessing and thus break the property of zero knowledge. This is not needed in VOLEitH, since the values for unmasking are never transferred to the verifier and through this no conclusions can be drawn about the witness itself. Furthermore, masking using randomness is used in both protocols to protect the witness and at the same time allow the verifier to check the proof using the witness's calculations.
The next step is to calculate the MPC protocol respectively the VOLE ZKP "in the head" of the prover. The big challenge in [KKW18] is the $M$ repetition of the offline and online phases and the associated cut-and-choose technique. The larger $M$ is, the longer the prover and verifier need to calculate the proof. At the same time, more communication is

| MPCitH | | | VOLEitH | | |
|---|---|---|---|---|---|
| Steps | Prover | Verifier | Steps | Prover | Verifier |
| Initialisation | ——— | ——— | Initialisation | sVole | ——— |
| ——— | ——— | ——— | Commit to Witness | Matrix_Sub | ——— |
| ——— | ——— | ——— | 1. Challenge | ——— | Rand |
| MPC Protocol | $M \cdot n \cdot$ Rand<br>$M \cdot n \cdot$ Merkle<br>$M \cdot$ Offline<br>$M \cdot n \cdot$ Commit<br>$M \cdot$ Online<br>$2 \cdot$ Hash | ——— | VOLE ZKP | $t \cdot$ Poly_Eval | ——— |
| Result | $2 \cdot$ Hash | ——— | Result | Summation | ——— |
| Commit | Hash | ——— | Send Masked Result | ——— | ——— |
| Challenge | ——— | ——— | 2. Challenge | ——— | Rand |
| Openings | ——— | ——— | Answer | Matrix_Add | ——— |
| ——— | ——— | ——— | 3. Challenge | ——— | Rand |
| ——— | ——— | ——— | Answer | Matrix_Add | ——— |
| Verify | ——— | $\tau \cdot$ Commit<br>Hash<br>$M \cdot$ Offline<br>$(M - \tau) \cdot$ Merkle<br>$(M - \tau) \cdot$ Hash<br>Hash<br>$\tau \cdot$ Online<br>$\tau \cdot$ Hash<br>Hash | Verify | ——— | Matrix_Add<br>$t \cdot$ Poly_Eval<br>Summation<br>check_VOLE |

Table 5.4: Overview of computational complexity for prover and verifier in MPCitH and VOLEitH. The costs are generally stated and can vary due to the use of different algorithms. The verifier's communication is shown in colour and can be ignored after applying the Fiat-Shamir Transformation.

required, but the probability that a cheating attacker will not be caught decreases as the protocol executes more rounds. Therefore, a larger $M$ provides more overhead, but also more security. There is no such direct connection in VOLEitH. Here a verifier can, similar to conventional MPCitH, send several challenges ($\Delta'$) and thereby detect a cheating prover. In addition, VOLEitH does not require as much randomness as the approach described above. When creating the seeds for the GGM Tree, randomness is required once. This also applies to the creation of the VOLE correlation or the challenges of the verifier.

All further calculations are then executed based on these values. With [KKW18], on the other hand, fresh randomness is needed in every round to create seeds for new Merkle Trees for each party. The effort here also increases linearly with the number of parties to be simulated and the number of runs. The last building block is the commitment used for the offline phase, which should later enable the verifier to verify the correctness of the preprocessing phase. This step is specific to this MPCitH protocol since a (malicious) prover can cheat in the preprocessing phase and only this additional step enables the verifier to detect this. This step is executed very often and compressed by hashing to keep communication costs low. None of this is needed with VOLEitH since only the $t$ polynomials need to be evaluated. Remember, $t$ was the number of multiplications within the circuit, which is the same as $|C|$ in MPCitH approaches. So the number of polynomials to be evaluated grows linearly with the size of the circuit, which makes the approach efficient for small and medium-sized circuits, but can become more difficult for larger calculations. Since the verifier can carry out every calculation himself and the prover does not have to hold back any values, no additional calculations are required here in addition to the later challenges. These are done to allow the verifier to detect a cheating prover and thereby achieve good soundness in the protocol. For this, the prover needs linear operations in the size of the matrices in the calculations, with most of these operations being matrix additions.

Another advantage of VOLEitH is that the result of the Quicksilver ZKP can simply be summed up and does not need to be further compressed for transmission. In [KKW18] this is much more complicated, as additional hashes are necessary for efficient transmission. This means that both the prover and later the verifier have to calculate these hashes, but saves costs in communication. Such savings also exist in Baum et al.'s protocol, using the challenge $\eta$ to calculate the answer to the last challenge. This is also an additional step that the prover and verifier have to execute, which only serves to keep communication low. Thus, both protocols use techniques to reduce communication and thereby utilise the trade-off between communication and computational complexity.

The final part is the verification of the proof. As already discussed, the verifier simulates the steps of the prover in both protocols and then checks whether the values sent by the prover match the recalculated results. The big difference between [KKW18] and VOLEitH is the amount of values to be calculated. In VOLEitH, the verifier first calculates the new masking of the witness to evaluate the common polynomials to the same values as the prover. Then it calculates the coefficients, compares them with the prover's coefficients and then obtains his VOLE correlation values using the all-but-one Vector Commitments, which were calculated by the prover. These calculations can be carried out very efficiently because they only contain linear dependencies in the size of the matrices and polynomial evaluation. In particular, the Vector Commitment described above only needs to be

opened by the prover and does not require any calculations by the verifier. The two protocols differ fundamentally in this, since in [KKW18] the verifier also has to carry out the calculation of the Merkle Trees again. This is related to the verification of the preprocessing phase, in which the verifier must ensure that all hashes have been calculated correctly. To do this, he gets the seeds used by the prover and has to create the associated Merkle Trees from them. This allows him to calculate the randomness used and the associated hashes without receiving them directly from the prover. This saves communication and ensures that the prover has not chosen incorrect values. In addition, $\mathcal{V}$ can calculate all hashes of the preprocessing phase itself and therefore does not have to receive all states of the prover, which again saves communication. To check the online phase, the verifier must recalculate all commitments, simulate the online phase and hash the results together. The hash function used is an important part of the entire protocol, as the complexity of the calculation accounts for a very large part of the total calculations.

**Repetition Code VOLE** By using repetition codes and applying them in AES-based signatures, [BBdSG+23b] not only optimise on the necessary communication cost, as discussed in Section 5.2, but also significantly reduce the computational complexity. These results are primarily achieved by improving on the calculations with the witness, being able to use vectors for masking instead of matrices, as well as the removal of the additional challenges to check the correctness of the sent values. However, this also requires new, additional operations on repetition codes, such as embedding or lifting into different dimensional spaces. This approach took advantage of the fact that AES is defined above the extension field of $\mathbb{F}_2$ and this makes complicated calculations easier to execute. The calculations within the Quicksilver proof remain as before, only an additional step is added to calculate the masking of the result. In the verification, the prover now needs to calculate the additional liftings and embeddings on top, but otherwise only has to calculate the coefficients and check the VOLE correlation. After using the compiler, which replaces the ideal functionality with Vector Commitments, the calculation of such a commitment by the prover is again required. In this part, we only consider the generalised VOLEitH approach in Table 5.4, as it is more similar to MPCitH.

# 6 Modifications

In the previous descriptions, we highlighted some bottlenecks in the execution of MPCitH and VOLEitH protocols. This chapter now deals with the modification of the protocols for improved use in digital signatures. We describe the rough idea and the theoretical effects on the complexity of the protocol.

## 6.1 Reducing the Amount of Rounds in [KKW18]

The MPCitH approach described by [KKW18] divides the proof into a preprocessing and an online phase. The cut-and-choose technique is used for verification, which forces the prover to execute multiple rounds and to carry out expensive calculations, such as building the Merkle Trees or hashing the results, several times. Reducing the number of rounds would therefore lead to an improvement in signing and verification time, as the effort for the prover and verifier would be reduced. In addition, less communication would be needed as the openings become smaller and thus fewer elements have to be sent.
In a first step, we want to take a closer look at whether the $M$ times of execution of the prover and the resulting calculations of the verifier can be combined into a single round without major adjustments to the protocol. To implement this, the prover would only draw one seed in the preprocessing phase and use it to calculate the Merkle Tree for the $n$ parties. He then runs the offline and online phases of the protocol and calculates the commitments for them. Since only one round is executed, the prover does not have to calculate the final hashes to reduce communication, but can directly send the hash of the offline and online phases. The problem with this approach is the verification. To check the preprocessing phase, the verifier receives the seed of the Merkle Trees in the original protocol. Since only one round was carried out here, there is only one tree and therefore only one seed. This allows him to reconstruct the entire tree. By that, all states and inputs of the $n$ parties can be reconstructed. Furthermore, $\mathcal{V}$ can learn the mask of the witness and thus break the zero-knowledge property of the protocol. If the verifier does not check the preprocessing phase, but only the online part, he can no longer ensure that the prover has calculated the states and randomness correctly. This means that a (malicious) prover could cheat in the preprocessing phase without the verifier being able to prevent this. Therefore, the number of rounds can only be minimised, but not reduced to one round.

Presume now that prover and verifier run the original protocol but use fewer rounds than described in the paper. This means that the proof can be calculated more quickly, but the verifier can use fewer openings to check offline and online phases. This reduces the probability that the verifier will expose a cheating prover and thus worsens the soundness of the protocol. Thus, computational and communication complexity will be saved at the cost of security, which describes the trade-off we discussed before.

## 6.2 Introducing Vector Commitments into [KKW18]

The first approach of simply reducing the number of rounds without major adjustments to the protocol itself led to a reduction in security. Therefore, we now want to look at what happens if we replace individual building blocks of the protocol and thereby save calculations. In particular, we want to take the approach of [BBdSG+23b] and use Vector Commitments based on GGM Trees, which replace the previous Merkle Trees. The idea is to save the recalculation of the Merkle Trees by the verifier and replace it with openings of the Vector Commitments. This should result in savings in verification time but additional communication overhead.
In the preprocessing phase, the prover now uses the seeds to construct a GGM Tree. For this, he needs a collision-resistant hash function and a pseudo-random generator to expand the seeds. Notice, that he has to calculate a deeper tree because he needs $2n$ elements, $n$ for the calculation of the states and $n$ as randomness for the online phase, but the structure of the GGM Vector Commitments provides that the last expansion creates an element for further calculation and the second element is the associated commitment (for detailed information consider Section 2.5). Therefore, more seed expansions need to be done to create enough elements, resulting in a larger tree. By using the Vector Commitments, the prover can skip the subsequent preprocessing step, the calculation of the commitments for states and randomness, since he can then use the elements of the last expansion step of the GGM construction. The prover then carries out the calculation of the offline and online phases as usual and receives the challenges from the verifier. In the verification, $\mathcal{V}$ recalculates the commitments of all parties that are not the challenge party for the online phase based on the states and randomness received from the prover. This step can be saved by the prover performing an all-but-one opening of the Vector Commitment. To verify the offline phase, the verifier has to recreate all Merkle Trees based on the seeds. By applying the Vector Commitments, the prover can open the commitments and thus allow the verifier to check all calculations without having to recalculate all elements. In both steps, the verifier just has to recalculate the hash based on the openings. The rest of the verification is carried out as before.

By applying these changes, calculations by the verifier are saved at the expense of communication and calculations by the prover. The big advantage is that the verification time is reduced because $\mathcal{V}$ does not have to reconstruct the Merkle Trees to check the received values. This is desirable because when such signatures are used in the real world, for example in software updates or emails, many entities have to verify the received content, so efficient verification is needed. But this also means that a larger tree and the associated Vector Commitment have to be created during the preprocessing phase. Unfortunately, there is currently no evidence whether the GGM construction is more efficient than the creation of Merkle Trees. So it should be further investigated to what extent the GGM construction is cheaper or more expensive than creating Merkle Trees to describe the changed effort of the prover in more detail. However, some improvements can be applied to GGM Trees ([BBM+24], [BCdSG24]) and will help improve this approach, but results in a much higher amount of communication. The prover no longer has to send all states, randomness and seeds, but only their commitments. In addition, he also has to open them for verification. The security of the protocol is still ensured and follows from the hiding and binding properties of the vector commitments. Thus, the introduction of Vector commitments in the [KKW18] protocol leads to an improved verification time, as not all values have to be recalculated. The prover has to build larger trees and calculate a vector commitment, which can lead to more effort and require more communication.

## 6.3 One Tree Approach

What remains with the previous adjustment are the $M$ rounds of preprocessing. In 2024, [BBM+24] presented FAESTer, a new VOLEitH-based signature and improvement of the FAEST algorithm presented in this work. The new idea is to create all the randomness necessary for the calculations from a wider GGM Tree and thus reduce the creation and expansion of seeds. We now want to apply this to the previously described scheme to save costs in the preprocessing.

In the preprocessing step, the verifier usually creates $M$ seeds and calculates the resulting Merkle Trees. With the introduction of Vector Commitments, described in Section 6.2, this step is replaced by building a GGM Tree and the associated Vector Commitment. Since the creation of the seeds and their expansion are still executed $M$ times, this approach is still very expensive. Therefore, the idea of [BBM+24] can now be applied by replacing the $M$ times drawing of seeds and GGM Tree expansion with computing a large tree based on one seed. This GGM Tree contains more elements because it has to create all the values for the $M$ executions of the offline and online phase of the MPC protocol, but at the same time saves the creation of many seeds and expansion steps. Afterwards, the prover and

verifier perform the normal protocol as described in section 6.2.
These changes to the protocol do not ensure that all rounds of preprocessing can be saved. However, creating the seeds and calculating the resulting trees is the most complex and therefore expensive step of this phase. This means that the prover has to carry out fewer calculations, which leads to an improvement in the signing time. But there is still an increased communication effort due to opening the Vector Commitments during verification. This approach can therefore be used to achieve improvements in the use of GGM Trees and Vector Commitments in the [KKW18] protocol, but to what extent this is practical and more efficient than calculating the Merkle Trees needs to be further investigated. Notice, that this type of tree extension can also be applied to Merkle Trees. However, the previous form of the protocol cannot be used for this, as it publishes the seeds of the individual trees to verify the preprocessing phases. Since this would compromise the security of the protocol if there was only one tree, the protocol would have to be adapted to allow for a large Merkle Tree.

# 7 Conclusions

This chapter summarises the results of this thesis and describes some ideas for future work.

## 7.1 Summary

In this work, we have shown a detailed description of the VOLEitH approach, a comparison to MPCitH and improvements to digital signatures. In the beginning, we explained the basic idea behind VOLE and showed the extension of Oblivious Transfer to SoftSpokenOT. We showed how SoftSpokenOT creates VOLEs over any polynomial-sized field and how the consistency of the VOLE correlation helps to be secure against malicious senders. Based on this, we explained the generalised subspace VOLE approach used in VOLEitH. The idea is that there is no restriction on the size of the field, but all calculations are performed on a subfield of polynomial size. This means there are fewer limitations, but the SoftSpokenOT approach can still be used.
To develop a better understanding of digital signatures and create a good basis for VOLEitH, we took a closer look at Zero-Knowledge Proofs using MPC protocols. To do this, we presented the MPCitH approach, where the prover executes an MPC protocol by simulating $n$ parties "in the head". We also showed how the Fiat Shamir Transformation can be used to construct a non-interactive proof, which forms a digital signature. Building on this, we introduced VOLEitH. We have described the individual rounds in detail and discussed why the individual steps are necessary. We have also shown which security mechanisms are in the protocol against a malicious prover. We discussed the idea behind FAEST, the first VOLEitH-based signature, and have shown that FAEST is a very specific protocol that was very tailored to AES and is therefore not very generalisable.
The authors of VOLEitH emphasise in their paper that their approach is up to twice as fast as comparable MPCitH protocols. Since it was not immediately obvious to us why this was the case, we provided a comparison of MPCitH and VOLEitH. We looked at general similarities and differences, which showed that the structure of both approaches is very similar, but the generalised subspace VOLEitH approach requires significantly more steps to protect against malicious provers. We have worked out a few bottlenecks where we saw room for optimisations and described some significant trade-offs. We further looked at communication and calculation complexity. Since MPCitH is a very general approach,

we decided to use [KKW18] as a representative for MPCitH protocols in the comparison, as it is very optimised. We have explained the general structure of [KKW18] in more detail, which achieves optimisations by dividing the steps of the prover into an offline and online phase. The subsequent comparison showed that although VOLEitH has to carry out more steps, but they are cheaper to calculate and can also be sent efficiently. However, there is a clear difference between the generalised VOLEitH approach and the optimised variant for FAEST, as many additional steps are saved through lifting and embedding in repetition codes and more efficient data structures can be used.
In the final part of this work, we use the previously collected knowledge to develop optimisations for digital signatures. Since the VOLEitH approach and the signatures based on it are already very optimised, we primarily looked at the [KKW18] protocol. It turned out that the expensive calculation of the Merkle Trees by prover and verifier is a bottleneck and that the preprocessing has to be carried out $M$ times is also a limitation. Thus, we tried to combine the $M$ rounds into one without fundamentally changing the structure of the protocol. This is not possible because the verifier is then able to learn the prover's secret or is no longer able to detect a cheating prover. Based on the VOLEitH approach, we then used Vector Commitments using the GGM Tree construction to calculate the randomness and commitment of the offline phase. This means that the prover has to calculate larger trees to represent the entire randomness, but the verifier does not have to recompute all the trees but can check the provers commitments. In this way we achieve a theoretical improvement in the verifier time, but an increase in communication. Unfortunately, there are no comparisons of GGM and Merkle Trees, so it is not possible to estimate whether one approach is more efficient than the other. Based on the elaboration of [BBM+24], we also described how the $M$-fold calculation of the Vector Commitments from different GGM Trees can be combined into one large Vector Commitment using a single GGM Tree. This again requires the calculation of a larger GGM Tree but saves multiple steps of recalculation and allows the Vector Commitment to be calculated more efficiently.
In summary, we have shown how VOLEs can be created based on Oblivious Transfer. With MPCitH and VOLEitH, we presented two approaches to create Zero-Knowledge Proofs. Such interactive protocols can be transformed to be non-interactive using Fiat Shamir Transformation. It can be shown that such protocols are post-quantum secure signature schemes. We compared MPCitH and VOLEitH, highlighted bottlenecks and described theoretical optimisations of [KKW18].

## 7.2 Future Work

In this section, we note a few approaches for future work.

**Optimisations on VOLEitH.** Based on its calculation of the masking and the evaluation of the polynomials, VOLEitH is very efficient in terms of the calculation complexity and the required communication. Nevertheless, the size of all matrices and the number of polynomials is directly dependent on the circuit size. VOLEitH achieves good results for small and medium-sized circuits but is not very efficient for very large circuits. Therefore, it should be checked whether this dependency can be solved. One approach could be to consider only a subset of the and-gates. This could save calculations, but would probably also increase the soundness error. Additionally, it is necessary to ensure that the omitted and- gates are chosen randomly so that the attacker cannot learn which values are being checked by the verifier.
Furthermore, it should be investigated to what extent the generalised VOLEitH protocol can be implemented in practice. Previous work focused on the small-field version, which is also used in FAEST. This shortens many steps of the subspace VOLE protocol for arbitrarily big fields and allows calculations to be performed more efficiently. Therefore, the generalised protocol should also be tested in practice.

**Implementation of Modifications.** During this work, we presented the MPCitH and VOLEitH approaches in detail, highlighted bottlenecks and developed modifications to the [KKW18] protocol. Since we have only given a high-level description of the changes and their supposed impact on complexity and security, the next logical step would be to have them practically implemented and tested. The focus should be primarily on the adjustments to the protocol and their effects on the creation of digital signatures, as improving such signatures was also part of this work.

**Merkle Trees against GGM Trees.** As we have already discussed in Section 6.2, there are no known works on the complexity of creating Merkle Trees or GGM Trees. Therefore, we cannot determine whether the optimisations presented improve on the existing work. In the worst case, our optimisations might even slow down the protocol, should it turn out that creating Vector Commitments via GGM Trees is much more complicated than the normal commitments via Merkle Trees. Consideration of this topic can also bring further advantages in improving MPCitH and VOLEitH since many of the protocols rely on such trees.

## References


[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, et al. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. *NIST report*, 2022.

[ABB+23] Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Victor Dyseryn, Andre Esser, Philippe Gaborit, Mukul Kulkarni, Victor Mateu, Marco Palumbi, et al. PERK. *Hal open science*, 2023.

[ABE+21] Diego F Aranha, Sebastian Berndt, Thomas Eisenbarth, Okan Seker, Akira Takahashi, Luca Wilke, and Greg Zaverucha. Side-Channel Protections for Picnic Signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 239–282, 2021.

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure Arithmetic Computation with Constant Computational Overhead. In *Annual International Cryptology Conference*, pages 223–254. Springer, 2017.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017.

[AMGH+23] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the SDitH. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 564–596. Springer, 2023.

[BBdSG+23a] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Sebastian Ramacher, Christian Rechberger, Emmanuela Orsini, Lawrence Roy, et al. FAEST: Algorithm Specifications. *https://faest.info/faest-spec-v1.1.pdf*, 2023.

[BBdSG+23b] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly

Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head. In *Annual International Cryptology Conference*, pages 581–615. Springer, 2023.

[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. *Cryptology ePrint Archive*, 2024.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences*, 37(2):156–189, 1988.

[BCdSG24] Dung Bui, Kelong Cong, and Cyprien Delpech de Saint Guilhem. Improved All-but-One Vector Commitment with Applications to Post-Quantum Signatures. *Cryptology ePrint Archive*, 2024.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 489–518. Springer, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing Vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 896–912, 2018.

[BdSGK+21] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and Fast Signatures from AES. In *IACR International Conference on Public-Key Cryptography*, pages 266–297. Springer, 2021.

[Bea96] Donald Beaver. Correlated Pseudorandomness and the Complexity of Private Computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488, 1996.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 163–193. Springer, 2017.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 1–10, 1988.

[BKPV24] Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier Verbel. Biscuit: New MPCitH Signature Scheme from Structured Multivariate Polynomials. In *International Conference on Applied Cryptography and Network Security*, pages 457–486. Springer, 2024.

[BKST15] Khodakhast Bibak, Bruce M. Kapron, Venkatesh Srinivasan, and László Tóth. On an Almost-Universal Hash Function Family with Applications to Authentication and Secrecy Codes. Cryptology ePrint Archive, Paper 2015/1187, 2015. `https://eprint.iacr.org/2015/1187`.

[BL17] Daniel J Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017.

[Ble11] Gerrit Bleumer. *Random Oracle Model*, pages 1027–1028. Springer US, Boston, MA, 2011.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A General Composition Theorem for Secure Reactive Systems. In Moni Naor, editor, *Theory of Cryptography*, pages 336–354, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Bui24] Dung Bui. Shorter VOLEitH Signature from Multivariate Quadratic. *Cryptology ePrint Archive*, 2024.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray A Perlner, and Daniel Smith-Tone. *Report on Post-Quantum Cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology . . . , 2016.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-Quantum Zero-Knowledge and Signatures from Symmetric-

Key Primitives. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 1825–1842, 2017.

[CDG+20] Melissa Chase, David Derler, Steven Goldfeder, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, et al. The Picnic Signature Scheme. *Submission to NIST Post-Quantum Cryptography project*, 2020.

[CW77] J Lawrence Carter and Mark N Wegman. Universal Classes of Hash Functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, 1977.

[DDM+06] Anupam Datta, Ante Derek, John C Mitchell, Ajith Ramanathan, and Andre Scedrov. Games and the Impossibility of Realizable Ideal Functionality. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 360–379. Springer, 2006.

[DdSGOT21] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient Zero-knowledge MPCitH-based Arguments. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3022–3036, 2021.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39*, pages 356–383. Springer, 2019.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DIO20] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-Point Zero Knowledge and Its Applications. *Cryptology ePrint Archive*, 2020.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a New Multivariable Polynomial Signature Scheme. In *International conference on applied cryptography and network security*, pages 164–175. Springer, 2005.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[Fen22] Thibauld Feneuil. Building MPCitH-based signatures from MQ, MinRank, rank SD and PKP. *Cryptology ePrint Archive*, 2022.

[FR23] Thibauld Feneuil and Matthieu Rivain. Threshold Computation in the Head: Improved Framework for Post-Quantum Signatures and Zero-Knowledge Arguments. *Cryptology ePrint Archive*, 2023.

[FS86] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

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

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. {ZKBoo}: Faster {Zero-Knowledge} for Boolean Circuits. In *25th usenix security symposium (usenix security 16)*, pages 1069–1083, 2016.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The Knowledge Complexity of Interactive Proof-Systems. *ACM Symposium on Theory of Computing*, 1985.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that Yield Nothing But Their Validity All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[Gol01] Oded Goldreich. *Foundations of Cryptography - Basic Applications*, volume 2. Cambridge university press, 2001.

[GSE21] Tim Gellersen, Okan Seker, and Thomas Eisenbarth. Differential Power Analysis of the Picnic Signature Scheme. In *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings 12*, pages 177–194. Springer, 2021.

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-Tree: Halving the Cost of Tree Expansion in COT and DPF. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 330–362. Springer, 2023.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

[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*, pages 21–30, 2007.

[Imp95] Russell Impagliazzo. A Personal View of AverageCase Complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 134–147. IEEE, 1995.

[Kil88] Joe Kilian. Founding Crytpography on Oblivious Transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, 1988.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT Extension for Transferring Short Secrets. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 54–70. Springer, 2013.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. Cryptology ePrint Archive, Paper 2018/475, 2018.

[Lam79] Leslie Lamport. Constructing Digital Signatures from a One Way Function. *Technical Report SRI-CSL-98, SRI International Computer Science Laboratory*, 1979.

[LJWJ24] Guoxiao Liu, Keting Jia, Puwen Wei, and Lei Ju. High-Performance Hardware Implementation of MPCitH and Picnic3. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):190–214, 2024.

[Mer79] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems*. Stanford university, 1979.

[PER24] PERK Team. Improvements for the PERK Signature Scheme, 2024.

[PS96] David Pointcheval and Jacques Stern. Security Proofs for Signature Schemes. In *International conference on the theory and applications of cryptographic techniques*, pages 387–398. Springer, 1996.

[Rab05] Michael O. Rabin. How To Exchange Secrets with Oblivious Transfer. *IACR Cryptol. ePrint Arch.*, 2005:187, 2005.

[Roy22] Lawrence Roy. SoftSpokenOT: Communication–Computation Tradeoffs in OT Extension. *Cryptology ePrint Archive*, 2022.

[RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21:120–126, 01 1978.

[SBWE20] Okan Seker, Sebastian Berndt, Luca Wilke, and Thomas Eisenbarth. SNI-in-the-head: Protecting MPC-in-the-head protocols against side-channel analysis. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1033–1049, 2020.

[Sch22] Peter Scholl. (Vector) Oblivious Linear Evaluation: Basic Constructions and Applications, 2022.

[SG12] Rajeev Sobti and Ganesan Geetha. Cryptographic Hash Functions: A Review. *International Journal of Computer Science Issues (IJCSI)*, 9(2):461, 2012.

[Sho94] Peter W Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[Sta21] StarkWare. ethSTARK documentation – version 1.1. Technical report, IACR preprint archive 2021, 2021.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1074–1091. IEEE, 2021.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2986–3001, 2021.