







The Return of the SDitH

Carlos Aguilar-Melchor¹ , Nicolas Gama¹ , James Howe¹ ,
Andreas Hülsing^{2*} , David Joseph¹ , and Dongze Yue¹ 

¹ SandboxAQ, Palo Alto, USA, `firstname.lastname@sandboxaq.com`

² Eindhoven University of Technology, The Netherlands, `andreas@huelising.net`

Abstract. This paper presents a code-based signature scheme based on the well-known syndrome decoding (SD) problem. The scheme builds upon a recent line of research which uses the Multi-Party-Computation-in-the-Head (MPCitH) approach to construct efficient zero-knowledge proofs, such as Syndrome Decoding in the Head (SDitH), and builds signature schemes from them using the Fiat-Shamir transform.

At the heart of our proposal is a new approach to amplify the soundness of any MPC protocol that uses additive secret sharing. An MPCitH protocol with N parties can be repeated D times using parallel composition to reach the same soundness as a protocol run with N^D parties. However, the former comes with D times higher communication costs, often mainly contributed by the usage of D ‘auxiliary’ states (which in general have a significantly bigger impact on size than random states). Instead of that, we begin by generating N^D shares, arranged into a D -dimensional hypercube of side N containing only one ‘auxiliary’ state. We derive from this hypercube D sharings of size N which are used to run D instances of an N party MPC protocol. This approach leads to an MPCitH protocol with $1/N^D$ soundness error, requiring N^D offline computation, *only* ND online computation, and *only* 1 ‘auxiliary’. As the, potentially offline, share generation phase is generally inexpensive, this leads to trade-offs that are superior to just using parallel composition.

Our novel method of share generation and aggregation not only improves certain MPCitH protocols in general but also shows in concrete improvements of signature schemes. Specifically, we apply it to the work of Feneuil, Joux, and Rivain (CRYPTO’22) on code-based signatures, and obtain a new signature scheme that achieves a 3.3x improvement in global runtime, and a 15x improvement in online runtime for their shortest signatures size (8.5 kB). It is also possible to leverage the fact that most computations are offline to define parameter sets leading to smaller signatures: 6.7 kB for 60 ms offline, or 5.6 kB for 700 ms offline. For NIST security level 1, online signature cost is around 3 million cycles (1 ms on commodity processors), regardless of signature size.

1 Introduction

Zero Knowledge (ZK) proofs of knowledge have become a fundamental cryptographic tool for modern privacy-preserving technologies and have many applications which range from authentication to online voting to machine learning. The idea of ZK proofs is that one party (a prover) can convince another party (a verifier) of the truth of a statement without revealing any other information about the statement itself.

A method for constructing efficient ZK proofs is to use the so-called MPC-in-the-Head (MPCitH) paradigm [IKO⁺07], in which semi-honest Multi-Party Computation (MPC) protocols are used as a basis. These protocols do not reveal any information on the secret used to prove a statement, even if some of the parties internal execution is revealed to an attacker. At a high-level, the MPCitH protocol has a prover which (i) secretly splits its secret input into *shares*, (ii) simulates “in their head” parties using said shares for the execution of a MPC protocol, and (iii) commits to this *execution* and partially reveals the internal execution of a subset of the parties to a verifier given some challenge. These internal executions can then be

* Andreas Hülsing is funded by an NWO VIDI grant (Project No. VI.Vidi.193.066).

checked for consistency by the verifier. To ensure that the prover has a very low probability to cheat, the verifier runs this protocol multiple times. The zero knowledge aspect of the overall protocol naturally inherits from a resilience to semi-honest adversaries of the underlying MPC protocol, as the verifier will only get to see a subset of the internal executions and the protocol will not reveal anything other than the correctness of the statement.

A recent proposal by Feneuil, Joux, and Rivain [FJR22] used this MPCitH idea to improve signature schemes based on the syndrome decoding (SD) problem; we refer to this work as SDitH. Previous proposals to make a signature scheme based on SD, such as those by Stern [Ste94], suffered from a high soundness error, which aligns to a malicious prover’s probability of cheating. Protocols with a higher soundness error require many more repetitions, compared to a protocol with a smaller soundness error, in order to achieve a target security level. Utilizing MPCitH in [FJR22] has enabled a low soundness error of $1/N$, for a party size N , whilst also being able to use a conservative code-based hardness assumption. At the time of writing, this approach makes the signature scheme the most performant code-based signature scheme for the common “signature size + public key size” metric.

Another reason to focus on this work is due to the NIST PQC standardization process. None of the code-based signature candidates were accepted by NIST into round 2, however, at the time of writing, we have many promising KEM candidates in the fourth round. An MPCitH-based signature, Picnic [ZCD⁺20], was part of the NIST PQC process, but NIST ultimately decided to standardize SPHINCS⁺ due to some security concerns with Picnic’s use of LowMC, but also because “future cryptosystems that evolve out of the multi-party-computation-in-the-head paradigm may eventually prove significantly superior to the third-round Picnic design”.

These two reasons were the motivation for this research; improving and optimizing a promising MPCitH-based signature scheme, which utilizes a well-established and conservative code-based hardness assumption. The contributions of this work are:

1.1 Contributions

- We propose a general geometrical hypercube approach for MPCitH that allows, from a state that was generated and committed for N parties, to obtain the same soundness as in a classical MPC-in-the-head by simulating the work of only $\log_2(N)$ parties instead of N .
- This approach runs multiple linked instances of MPCitH with only one masked auxiliary state, which reduces significantly the communication on the ZK protocol (and thus signature size) with respect to running independent instances of MPCitH with one auxiliary state for each of them.
- Applying these optimizations to SDitH, we observe a reduction of one third in signature size, for similar computational costs and security.
- As for SDitH, the signature resulting from our construction can be split in an offline and an online phase. But, unlike in SDitH, most of the computational cost is associated to the offline phase. Thus the online part of the signature is extremely fast in comparison, even for much smaller signatures.

2 Preliminaries

In this section we describe some standard cryptographic preliminaries which are similar to those in [FJR22]. For the entirety of this paper we will denote \mathbb{F} as a finite field. The Hamming

weight of a vector $\mathbf{x} \in \mathbb{F}^m$, denoted as $\text{wt}(\mathbf{x})$, is the number of non-zero coordinates of \mathbf{x} . We define the concatenation of two vectors $\mathbf{x}_1 \in \mathbb{F}^{m_1}$ and $\mathbf{x}_2 \in \mathbb{F}^{m_2}$ as $(\mathbf{x}_1|\mathbf{x}_2) \in \mathbb{F}^{m_1+m_2}$. For any $m \in \mathbb{N}_{>0}$, the integer set $\{1, 2, \dots, m\}$ is denoted as $[m]$. For a probability distribution D , we use the notation $d \leftarrow D$ to denote the value d is sampled from D . For a finite set S , the notation $s \leftarrow S$ denotes that the value s has been uniformly sampled at random from S . For an algorithm \mathcal{A} , $\text{out} \leftarrow \mathcal{A}(\text{in})$ further means that out is obtained by a call to \mathcal{A} on input in , using uniform random coins whenever \mathcal{A} is probabilistic. We also abbreviate probabilistic polynomial time as PPT.

2.1 Basic Cryptographic Definitions and Lemmas

Definition 1 (Indistinguishability). *Two distributions X, Y are (t, ϵ) -indistinguishable if for an algorithm running in time t , and $D: \{0, 1\}^m \rightarrow \{0, 1\}$, $|\Pr[D(X) = 1] - \Pr[D(Y) = 1]| \leq \epsilon(\lambda)$.*

The distributions are: computationally distinguishable when $t = \text{poly}(\lambda)$, and ϵ is a negligible function in λ ; and statistically indistinguishable when ϵ is a negligible function in λ for unbounded t ,

Definition 2 (Pseudorandom generation (PRG)). *Let $G: \{0, 1\}^* \rightarrow \{0, 1\}^*$ and let $\ell(\cdot)$ be a polynomial such that for any input $s \in \{0, 1\}^\lambda$ we have $G(s) \in \{0, 1\}^{\ell(\lambda)}$. Then, G is a (t, ϵ) -secure pseudorandom generator if (i) Expansion: $\ell(\lambda) > \lambda$ and (ii) Pseudorandomness: the distributions $\{G(s)|s \leftarrow \{0, 1\}^\lambda\}$ and $\{r|r \leftarrow \{0, 1\}^{\ell(\lambda)}\}$ are (t, ϵ) -indistinguishable.*

The standard cryptographic notion of tree PRG (TreePRG), initially proposed by Goldreich, Goldwasser, and Micali [GGM84], is used extensively in our construction. The general idea is to extend a length-doubling PRG and consider it over a tree structure: we start with a master seed (mseed) which is used to label the root node of a tree and expanded using a PRG into N sub-seeds in a structured way. For each node, its label is used as the seed of the PRG function, which generates two seeds that label the two children of the node. By proceeding iteratively at each level, over $\lceil \log_2(N) \rceil$ levels, we construct a binary tree with at least N leaves, labeled with PRG seeds that we note $(\text{seed}_i)_{i \in [N]}$. For any index i^* , someone can get the list of the $N - 1$ seeds $(\text{seed}_i)_{i \in [N], i \neq i^*}$ out of the sibling path of seed_{i^*} , which contains just $\lceil \log_2(N) \rceil$ seeds. This becomes a key component to efficiently generate the witness shares in Section 3.2.

In security proofs, we make use of the following lemma, which in essence says that a large subset A of a product space $X \times Y$ has many large areas.

Lemma 1 (Splitting Lemma [PS00]). *Let $A \subset X \times Y$, and $\Pr[(x, y) \in A] \geq \kappa$. Then for any $\alpha \in [0, 1)$, let*

$$B = \{(x, y) \in X \times Y | \Pr_{y' \in Y}[(x, y') \in A] \geq (1 - \alpha) \cdot \kappa\}, \quad (1)$$

Then the following are true: $\Pr[B] \geq \alpha \cdot \kappa$ and $\Pr[B|A] \geq \alpha$.

Lemma 2 (Forking Lemma for 5-pass protocols [DGV⁺16]). *Let S be an 5-pass signature scheme with security parameter k . Let \mathcal{A} be a PPT algorithm given only the public data as input. Assume that \mathcal{A} , after querying the 2 random oracles $\mathcal{O}1, \mathcal{O}2$ polynomially often in k , outputs a valid signature $(\sigma_0, \sigma_1, \sigma_2, h_1, h_2)$ for message m with a non-negligible probability. Let us consider a replay of this machine \mathcal{A} with the same random tape (as a Turing machine), the same response to the query corresponding to $\mathcal{O}1$ but a different output to $\mathcal{O}2$. Then running \mathcal{A} and its reply results in two valid signatures $(\sigma_0, \sigma_1, \sigma_2, h_1, h_2)$ and $(\sigma_0, \sigma_1, \sigma'_2, h_1, h'_2)$ for the same message m and $h_2 \neq h'_2$ with a non-negligible probability.*

While proving equality of polynomials can be inefficient, we can say something about the likelihood that two polynomials are different *and yet* are equal at certain points. This enables one to reduce the checking of polynomial relations to instead checking simple integer arithmetic relations, up to some well defined probabilistic error.

Lemma 3 (Multi-point Schwarz-Zippel lemma). *Let $P \in \mathbb{F}[x]$ be a non zero polynomial in one variable of degree at most d and $S \subseteq \mathbb{F}$ a non empty set of size at least t . For $R \subseteq S$ drawn uniformly from size t subsets of S ,*

$$\Pr[P(r) = 0, \forall r \in R] \leq \frac{\binom{d}{t}}{\binom{|S|}{t}}. \quad (2)$$

Proof. Let $D \subseteq \mathbb{F}_q$ denote the roots of P . Clearly $|D| \leq d$, since a non zero polynomial in one variable over a field has at most as many roots as its degree. The lemma follows since R is chosen uniformly from the $\binom{|S|}{t}$ size t subsets of S and there are at most $\binom{d}{t}$ size t subsets of D . \square

2.2 Zero-Knowledge Proofs

We define below the required properties for a Zero-Knowledge Proof of Knowledge. A proof of knowledge for some language $L \in NP$ is a two party protocol between prover \mathcal{P} and verifier \mathcal{V} , denoted $\langle \mathcal{P}, \mathcal{V} \rangle$ that should satisfy certain properties. The intention is for \mathcal{P} to prove to \mathcal{V} that their common input belongs to the language, i.e. $w \in L$.

Definition 3 ((Perfect) Completeness). *A proof of knowledge $\langle \mathcal{P}, \mathcal{V} \rangle$ is complete if, when both prover and verifier follow the protocol honestly, and the prover has knowledge of a legitimate witness w , then for every witness $w \in L$ the verifier accepts with probability 1:*

$$\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(w) = 1] = 1. \quad (3)$$

Definition 4 (Soundness). *A proof of knowledge is sound, with soundness error κ , if for a probabilistic polynomial time adversary, \mathcal{A} , with $w \notin L$, the probability of an honest verifier accepting is less than κ :*

$$\Pr[\langle \mathcal{A}, \mathcal{V} \rangle(w) = 1] \leq \kappa. \quad (4)$$

Put differently, this means that a prover without a valid witness w cannot convince the verifier to accept with probability greater than κ .

Definition 5 (Honest Verifier Zero Knowledge (HVZK)). *A proof of knowledge is HVZK if there exists a probabilistic polynomial time simulator \mathcal{S} that, without knowing a witness, outputs transcripts such that its output distribution is computationally indistinguishable from the distribution of transcripts derived from honest executions of the protocol $\langle \mathcal{P}, \mathcal{V} \rangle$.*

This means that running the protocol does not reveal any information about the witness to an honest observer. We use Zero-Knowledge proof as a shorthand for HVZK proof of knowledge.

The main protocol in this paper is a Zero-Knowledge proof. This protocol is built with the MPC-in-the-Head construction, which allows to transform a Multi-Party Computation protocol into a Zero-Knowledge proof. Before introducing the MPC-in-the-Head construction, we will first present some building tools needed for that construction: commitments, and additive secret sharing, a simple but efficient tool to build some MPC protocols.

2.3 Commitments

A commitment scheme is a cryptographic primitive that allows one to publish a value C , called commitment, associated to some other hidden value which can be revealed at a later stage through a procedure called opening using a decommitment value D . Once a party has committed to a hidden value, they should not be able to change the value, and no other party should be able to glean any knowledge of the value that has been committed, until the committing party opens the commitment.

Definition 6 (Commitment scheme). *A commitment scheme consists of two PPT algorithms, \mathbf{com} , \mathbf{open} , where*

- $\mathbf{com}(M)$ - on input $M \in \{0, 1\}^*$ the commitment algorithm outputs $(C, D) \leftarrow \mathbf{com}(M, \rho)$ where ρ is the commitment randomness.
- $\mathbf{open}(C, D)$ outputs M or \perp .

Definition 7 (Correctness). *If $\mathbf{com}(M) \rightarrow (C, D)$, then $\mathbf{open}(C, D) \rightarrow M$.*

A secure commitment scheme has the following two properties:

Definition 8 (Binding). *A commitment scheme is perfectly binding if, for all probabilistic polynomial time (in security parameter κ) algorithm \mathcal{A} , the probability of finding C, D, D' such that $\mathbf{open}(C, D) = M$, $\mathbf{open}(C, D') = M'$, and $M \neq M'$ is zero, and computationally binding if the probability is a negligible function in κ .*

Definition 9 (Hiding). *A commitment scheme is perfectly, statistically, or computationally (respectively) hiding if, for any two messages M, M' , the distributions $\{C : (C, D) \leftarrow \mathbf{com}(M)\}_{\kappa \in \mathbb{N}}$, and $\{C : (C, D) \leftarrow \mathbf{com}(M')\}_{\kappa \in \mathbb{N}}$ are perfectly, statistically, or computationally indistinguishable.*

A commitment scheme cannot be both perfectly hiding and perfectly binding simultaneously. In order to see this, suppose first that the scheme is perfectly binding, and one publishes the commitment $\mathbf{com}^k(\mathit{open}, x)$, therefore no other pair (open, x) outputs $\mathbf{com}^k(\mathit{open}, x)$. Then a computationally unbounded adversary can try inputs (open', x') until they find the correct inputs (open, x) , which uniquely give the correct output.

2.4 Additive Secret Sharing and Computing on Shares

In order to perform multi-party computation (MPC), it is necessary to break up and then distribute the input data of the function to be evaluated amongst multiple parties. In this work, we use an approach to break and use this data called additive secret sharing. It is defined by the following two routines:

- *Share*(\mathbf{x}): The *Share* routine randomly samples the $(N-1)$ -tuple $([\mathbf{x}]_1, [\mathbf{x}]_2, \dots, [\mathbf{x}]_{N-1}) \leftarrow (\mathbb{F}^m)^{N-1}$, and then computes $[\mathbf{x}]_N \leftarrow \mathbf{x} - \sum_{i=1}^{N-1} [\mathbf{x}]_i$. The final output is a tuple of N shares $[\mathbf{x}] \leftarrow ([\mathbf{x}]_1, [\mathbf{x}]_2, \dots, [\mathbf{x}]_N)$.
- *Reconstruct*($[\mathbf{x}]$): The *Reconstruct* routine combines all N shares together by summation to obtain the original value $\mathbf{x} \leftarrow \sum_{i=1}^N [\mathbf{x}]_i$.

In practice, one can compress the output of *Share*(\mathbf{x}) by expanding shares $([\mathbf{x}]_1, [\mathbf{x}]_2, \dots, [\mathbf{x}]_{N-1})$ from random seeds, however most of the terms in the final share $[\mathbf{x}]_N$ must be communicated in full, without compression. We call this final share *aux*, which is defined explicitly in Algorithm 1.

A secret value x can thus be distributed to N parties in a MPC scenario. Each party i in the MPC protocol receives share $[[\mathbf{x}]]_i$. It is important to observe that the parties cannot learn anything of \mathbf{x} unless they have all N shares. The parties are able to perform the following computations and obtain valid shares of a new secret-shared value:

- *Addition of shares:* Let $[[\mathbf{x}_A]], [[\mathbf{x}_B]]$ be two sets of shares distributed among parties. $[[\mathbf{x}_A + \mathbf{x}_B]]_i := [[\mathbf{x}_A]]_i + [[\mathbf{x}_B]]_i$.
- *Addition with a constant:* $[[\mathbf{x} + c]] := [[\mathbf{x}]]_1 + c, [[\mathbf{x}]]_2, \dots, [[\mathbf{x}]]_N$.
- *Multiplication with a constant:* $[[c \cdot \mathbf{x}]]_i := c \cdot [[\mathbf{x}]]_i$.
- *Multiplication of shares:* Multiplication is possible using Beaver triples [Bea92] with additional communication between parties (where the parties are given as additional input a secret-shared triplet $[[a]], [[b]], [[c]]$ where a, b are unknown to all players and $c = ab$). This additional triplet is sacrificed in order to validate another triplet, which is defined in the following.

One can evaluate an arbitrary function f over additive shares by decomposing f into an arithmetic circuit using the four types of computation listed above.

2.5 MPC-in-the-Head paradigm

The MPC-in-the-Head (MPCitH) paradigm originated from the work of Ishai et al. [IKO⁺07] and provides a path towards building zero-knowledge proofs for arbitrary circuits from secure multi-party computation (MPC) protocols. In this work, we use a semi-honest MPC protocol with additive shares that evaluates a Boolean decision function. The protocol has the following properties:

- *N -party decision function evaluation:* The N parties $\mathcal{P}_1, \dots, \mathcal{P}_N$ each possess an additive share $[[\mathbf{x}]]_i$ of the input \mathbf{x} . The parties jointly evaluate a decision function $f : \mathbb{Z}^m \rightarrow \{0, 1\}$ on \mathbf{x} .
- *Semi-honest $(N - 1)$ -Security:* Assuming the parties adhere to the protocol, the additive shares guarantee that any $N - 1$ parties cannot recover any information about the secret \mathbf{x} .

One can efficiently build a zero-knowledge proof of knowledge of a secret value \mathbf{x} for which $f(\mathbf{x}) = 1$, for a predicate f that has either a unique solution, or is hard to fulfill. The prover proceeds as follows:

- Generate shares of the secret $[[\mathbf{x}]] \leftarrow \text{Share}(\mathbf{x})$ and distribute the shares among N imaginary parties.
- Simulate the decision function evaluation procedure among the N imaginary parties “in the head”.
- Commit to the view (initial share, secret random tape, and inbound/outbound communications) of each party and send commitment to the verifier.
- Send the shares of the final computed result $[[f(\mathbf{x})]]$ to the verifier, which should reconstruct to 1.

The verifier performs the following steps to verify the proof:

- Randomly choose $N - 1$ parties, and ask the prover to reveal the views of those parties.
- Upon receiving the views, verify whether the views are consistent with an honest execution of the MPC protocol and agree with the commitments.

- The verifier accepts if the views are consistent and the final shares $\llbracket f(\mathbf{x}) \rrbracket$ indeed reconstruct to 1.

Some challenge randomness of the decision function f will be provided by the verifier. Therefore, the views of each party (input shares, random tape, initial message from preprocessing phase) prior to the joint function evaluation must be committed before the prover receives the randomness to prevent cheating.

The verifier does not learn any information about the secret value because they only see $N - 1$ shares. The random selection of $N - 1$ parties results in a soundness error of $\frac{1}{N}$ for the MPCitH protocol.

2.6 Syndrome Decoding

The Zero-Knowledge proof protocol we propose in this paper uses MPC-in-the-Head to prove a solution is known to a Syndrome Decoding problem. Syndrome decoding (SD) is a problem that is central to many code-based cryptosystems. A syndrome is the result of multiplying a vector with a parity check matrix, which implies that being a codeword is equivalent to having syndrome $\mathbf{0}$. The SD problem can be expressed as follows:

- *Challenge:* Parity-check matrix $\mathbf{H} \leftarrow \mathbb{F}_q^{(m-k) \times m}$, syndrome $\mathbf{y} \in \mathbb{F}_q^{m-k}$.
- *Required Output:* Vector $\mathbf{x} \in \mathbb{F}_q^m$ with $wt(\mathbf{x}) = w$ and $\mathbf{H}\mathbf{x} = \mathbf{y}$.

During challenge generation, \mathbf{H} and \mathbf{x} such that $wt(\mathbf{x}) = w$ are drawn uniformly at random, and then $\mathbf{y} = \mathbf{H}\mathbf{x}$ is calculated. The two most significant approaches to solving the syndrome decoding problem are information set decoding and birthday algorithms. In order for a SD-based cryptosystem to achieve security level λ it is necessary to select parameters such that each approach takes more than 2^λ operations to solve the underlying syndrome decoding instance.

2.7 Syndrome Decoding in the Head

In this section we describe the methodology of generating zero knowledge proofs (ZKP) from MPCitH applied to the syndrome decoding problem, as laid out in [FJR22]. For efficiency, we assume that \mathbf{H} is in standard form $\mathbf{H} = (\mathbf{H}' | I_{m-k})$, where $\mathbf{H}' \in \mathbb{F}_q^{(m-k) \times k}$. This enables us to express

$$\mathbf{y} = \mathbf{H}\mathbf{x} = \mathbf{H}'\mathbf{x}_A + \mathbf{x}_B, \quad (5)$$

so we only need to send \mathbf{x}_A to reveal the solution. The MPC protocol defined divides up \mathbf{x}_A into shares $\llbracket \mathbf{x}_A \rrbracket$, from which parties can reconstruct shares of $\llbracket \mathbf{x} \rrbracket$.

The protocol then verifies that $\mathbf{y} = \mathbf{H}\mathbf{x}$ and that \mathbf{x} has weight less than or equal to w by proving polynomial relations.

2.7.1 Polynomial construction

Let \mathbb{F}_{SD} be the finite field over which the syndrome decoding problem is defined. Let $\mathbb{F}_{\text{poly}} \supseteq \mathbb{F}_{\text{SD}}$ with $|\mathbb{F}_{\text{poly}}| > m$. Let $\phi: \mathbb{F}_{\text{SD}} \rightarrow \mathbb{F}_{\text{poly}}$ define the inclusion of \mathbb{F}_{SD} in \mathbb{F}_{poly} . Let f_i be the points in \mathbb{F}_{poly} .

The prover builds three polynomials, \mathbf{S} , \mathbf{Q} , and \mathbf{P} in order to prove the weight constraint. Polynomial $\mathbf{S} \in \mathbb{F}_{\text{poly}}[X]$ is the interpolation over the points of \mathbf{x} , with $\mathbf{S}(f_i) = \phi(x_i)$, and $\deg(\mathbf{S}) \leq m - 1$ and $\mathbf{Q}[X] \in \mathbb{F}_{\text{poly}}[X]$ is $\mathbf{Q} = \prod_{E}(X - f_i)$, where E is a subset of $[m]$ of order $|E| = w$, such that the nonzero points of \mathbf{x} are contained in E . Accordingly, \mathbf{Q} has degree w .

Polynomial \mathbf{P} is defined as $\mathbf{P} = \mathbf{S} \cdot \mathbf{Q} / \mathbf{F}$, where $\mathbf{F} = \prod_{[m]}(X - f_i)$. Ultimately, the polynomial relation

$$\mathbf{S} \cdot \mathbf{Q} = \mathbf{P} \cdot \mathbf{F}, \quad (6)$$

must be satisfied in order to prove that $wt(\mathbf{x}) \leq w$. The left hand side is designed so that $\mathbf{S}\mathbf{Q}(f_i) = 0$ for all $f_i \in [m]$. This is because \mathbf{S} is zero everywhere that \mathbf{x} is zero (by construction, as \mathbf{S} is interpolated over \mathbf{x}), and \mathbf{Q} is zero everywhere that \mathbf{x} is not zero. Polynomial \mathbf{S} has degree w and \mathbf{Q} has degree m .

On the right-hand side, by construction public polynomial \mathbf{F} is zero everywhere in $[m]$, and polynomial \mathbf{P} is required because \mathbf{F} has degree m , whereas $m < \deg(\mathbf{S}\mathbf{Q}) \leq m + w - 1$. If the prover can convince the verifier that they know \mathbf{P}, \mathbf{Q} such that $\mathbf{S} \cdot \mathbf{Q} = \mathbf{P} \cdot \mathbf{F} = 0$ at all points $f_i \in [m]$, then at each point f_i , either $\mathbf{S}(f_i) = \phi(\mathbf{x}_i) = 0$, or $\mathbf{Q}(f_i) = 0$. But since \mathbf{Q} has degree w , it can be zero at at most w points, therefore \mathbf{S} is nonzero in at most w points f_i , and so \mathbf{x} has weight of at most w .

In order to verify the polynomial relation of Eq. 6, the polynomial $\mathbf{S} \cdot \mathbf{Q} - \mathbf{P} \cdot \mathbf{F}$ is evaluated at a series of points to check that it evaluates to zero everywhere. This is because, by the Schwartz-Zippel lemma (Lemma 3), it is unlikely that the relation of Eq. 6 holds true at a random point, if the polynomial relation is not true in general. Picking t points at random to test the relation amplifies this result. Therefore the probability that the relation is satisfied at points $\{r_k\}_{k \in [t]}$ without Equation 6 being true becomes some sufficiently small probability we call p . This event is referred to as a false positive, which we denote \mathcal{F} . False positives affect the soundness of a ZKP, as they represent a way to be accepted by a verifier, but without knowledge of a valid witness. Consequently, the soundness of an MPCitH protocol based on syndrome decoding would be

$$1 - \left(1 - \frac{1}{N}\right)(1 - p) = \frac{1}{N} + p - \frac{1}{N} \cdot p. \quad (7)$$

2.7.2 Polynomial relation proof via MPC-in-the-Head

The shares that are distributed to parties are the shares of $\mathbf{x}_A \in \mathbb{F}_{\text{SD}}^k$, the coefficients of $\mathbf{Q} \in \mathbb{F}_{\text{poly}}^w$, and coefficients of $\mathbf{P} \in \mathbb{F}_{\text{poly}}^{w+1}$, as well as the shares of t beaver triplets $(a_k, b_k, c_k = a_k b_k) \in \mathbb{F}_{\text{points}}^3$.

A party's share is denoted with double square brackets and an index, such as $\llbracket \mathbf{x} \rrbracket_i$. Shares of polynomials are shares of the coefficients of the polynomials, and for \mathbf{Q} , only the last $w - 1$ coefficients are shared due to \mathbf{Q} being monic. Instead of evaluating the full relation of Equation 6, we validate that the relation holds true at t randomly selected points $\mathbf{r} \in \mathbb{F}_{\text{points}}^t$, as explained in the previous section to reduce the probability of \mathcal{F} . In order to further reduce p , the points r_i are sampled from a larger space $\mathbb{F}_{\text{points}} \supset \mathbb{F}_{\text{poly}}$ as this makes it even more unlikely that an untrue polynomial relation looks correct at a given point r_i .

In order to verify the multiplication triple $\mathbf{S}(r_k) \cdot \mathbf{Q}(r_k) = \mathbf{P} \cdot \mathbf{F}(r_k)$, we sacrifice a Beaver triple $a_k \cdot b_k = c_k$. The protocol proceeds as follows:

1. Sample $\mathbf{H} \in \mathbb{F}_q^{(m-k) \times m}$, $\mathbf{x} \in \mathbb{F}_q^m$ uniformly at random and compute $\mathbf{H}\mathbf{x} = \mathbf{y} \in \mathbb{F}_q^{(m-k)}$
2. Sample $\mathbf{r}, \boldsymbol{\epsilon} \in \mathbb{F}_{\text{points}}^t \times \mathbb{F}_{\text{points}}^t$ uniformly at random
3. Construct $\llbracket \mathbf{x} \rrbracket$ and express it over \mathbb{F}_{poly} .
4. Interpolate the shares $\llbracket \mathbf{S}(r_k) \rrbracket$ and construct $\llbracket \mathbf{Q}(r_k) \rrbracket$, and $\llbracket \mathbf{F} \cdot \mathbf{P}(r_k) \rrbracket$.
5. Run MPC protocol to verify the triple $(\llbracket \mathbf{S}(r_k) \rrbracket, \llbracket \mathbf{Q}(r_k) \rrbracket, \llbracket \mathbf{P} \cdot \mathbf{F}(r_k) \rrbracket)$ with sacrificed triple $(\llbracket a_k \rrbracket, \llbracket b_k \rrbracket, \llbracket c_k \rrbracket)$.
 - (a) Set $\llbracket \alpha_k \rrbracket = \boldsymbol{\epsilon}_j \cdot \llbracket \mathbf{Q}(r_k) \rrbracket + \llbracket a_k \rrbracket$ and set $\llbracket \beta_k \rrbracket = \llbracket \mathbf{S}(r_k) \rrbracket + \llbracket b_k \rrbracket$.

- (b) Parties open $\llbracket \alpha_k \rrbracket$ and $\llbracket \beta_k \rrbracket$ on bulletin board to construct α_k and β_k .
- (c) Parties set $\llbracket v_k \rrbracket = \epsilon_k \cdot \llbracket \mathbf{F} \cdot \mathbf{P}(r_k) \rrbracket - \llbracket c_k \rrbracket + \alpha_k \cdot \llbracket b_k \rrbracket + \beta_k \cdot \llbracket a_k \rrbracket - \alpha_k \cdot \beta_k$.
- (d) Parties open $\llbracket v_k \rrbracket$ to obtain v_k and check that it encodes zero.

2.7.3 False-positive probability

To evaluate the false positive probability, necessary (along with N) to compute the soundness in Eq. 7, consider that at each point r_k , either $\mathbf{S}(r_k) \cdot \mathbf{Q}(r_k) - \mathbf{P} \cdot \mathbf{F}(r_k) = 0$ or is nonzero, so for i of the t challenge points to satisfy the relation (equivalently, to be roots of $\mathbf{S} \cdot \mathbf{Q} - \mathbf{P} \cdot \mathbf{F}$), there are

$$\frac{\max_{l \leq m+w-1} \binom{l}{i} \binom{|\mathbb{F}_{\text{points}}| - l}{t-i}}{\binom{|\mathbb{F}_{\text{points}}|}{t}}, \quad (8)$$

ways this can happen by Lemma 3, considering that $\mathbf{S} \cdot \mathbf{Q} - \mathbf{P} \cdot \mathbf{F}$ has degree of less than $m + w$, meaning it has at most $m + w - 1$ roots, from which i of the t challenge points could be selected from. For the i points which are roots of the polynomial, having $c_k = a_k b_k$ makes the MPC protocol pass with probability 1; for the $t - i$ cases that the challenge points are not roots, $\mathbf{S}(r_k) \cdot \mathbf{Q}(r_k) \neq \mathbf{P} \cdot \mathbf{F}(r_k)$. In these cases, the MPC protocol will pass iff. $c_k = a_k \cdot b_k + \epsilon_k (\mathbf{S}\mathbf{Q} - \mathbf{P}\mathbf{F})(r_k)$, a value that depends linearly on ϵ_k and thus can only be guessed correctly with probability $1/|\mathbb{F}_{\text{points}}|$. Since it needs to occur for all non-root positions, this gives a probability $(1/|\mathbb{F}_{\text{points}}|)^{t-i}$.

Combining the above reasoning, the probability $\Pr[\mathcal{F}] = p$ of \mathcal{F} , is

$$p \leq \sum_{i=0}^t \frac{\max_{l \leq m+w-1} \binom{l}{i} \binom{|\mathbb{F}_{\text{points}}| - l}{t-i}}{\binom{|\mathbb{F}_{\text{points}}|}{t}} \cdot \left(\frac{1}{|\mathbb{F}_{\text{points}}|} \right)^{t-i}. \quad (9)$$

A less tight but more intuitive bound can be given by considering that each of the t challenge points is either a root of $\mathbf{S} \cdot \mathbf{Q} - \mathbf{P} \cdot \mathbf{F}$ which occurs with probability $\leq \frac{m+w-1}{|\mathbb{F}_{\text{points}}|}$, else it is not a root, and only satisfies the relation if ϵ_k was guessed correctly, with probability $\leq \frac{1}{|\mathbb{F}_{\text{points}}|}$. Summing these two probabilities (for a given challenge point), and considering that this must happen for all t challenge points, we arrive at the loose bound $p \leq \left(\frac{m+w}{|\mathbb{F}_{\text{points}}|} \right)^t$. It is necessary that p be comfortably smaller than $1/N$ which is the target soundness error of the MPCitH protocol in order to preserve zero knowledge for a ZKP.

3 Batch MPC-in-the-Head on a hypercube for ZK proofs

Here we describe how to amplify the soundness afforded to us by MPCitH techniques. We do this by using MPCitH to give a ‘proof of honesty’ of a party which remains hidden in traditional MPCitH.

3.1 Batch MPC-in-the-Head on a hypercube

To efficiently execute the MPC-in-the-Head proof and verification, we use a geometrical approach, illustrated in Figure 1. We split the secret input in N^D shares, and place them on a cubic lattice of dimension D , where each dimension has N slots. The lattice can be represented as a D dimensional hypercube with N points on each side.

We use these shares to set up D MPCitH intertwined executions with N parties each. Since we use an additive secret sharing scheme, the set of shares can be arbitrarily partitioned,

and the shares of a given partition subset added. Each resulting sum can be associated to a party, and the resulting MPC protocol will preserve correctness. Leveraging this property, for each dimension $k \in [D]$ of the hypercube, we run an MPCitH protocol of N parties, each using as share the summation of the N^{D-1} shares that have the same index i_k along the current dimension. This is visualized in Figure 1.

The significance of this technique is that one can identify a cheating leaf party (among N^D leaf parties) with only ND parties simulating the MPC protocol, and only one auxiliary (and thus large) share. In contrast, the original protocol of [FJR22] would require N^D executions of the MPC protocol or D parallel executions of N parties with D auxiliary shares. This can thus be seen as a speedup of the first approach, or as a communication reduction of the second approach.

Note that D parallel composition of an N party protocol is a trade-off (reducing computation, increasing communication) with respect to doing an N^D party protocol. Our approach is a direct improvement: we reduce the computational cost of an N^D party protocol, without increasing its communication cost. Of course, parallel composition can be applied whether our improvement is used or not. Thus, when parallel composition is used with a vanilla MPCitH protocol to explore different trade-offs, the vanilla N party protocol can be replaced with $\lceil \log_2 N \rceil$ 2-party protocols using our construction and thus improve the initial trade-off by reducing computational costs.

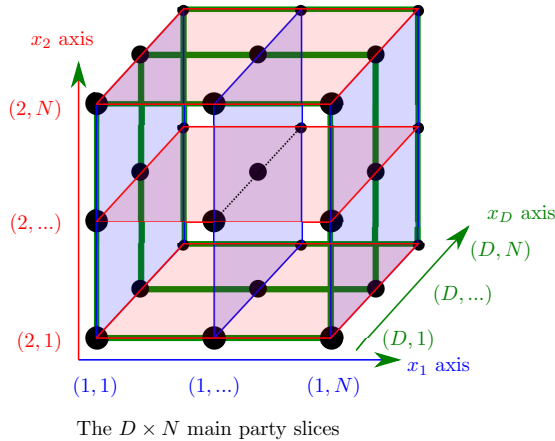


Fig. 1: A visualization of N^D leaf parties arranged on a dimension 3 hypercube, where there are only $N \times D$ main parties, including N main parties for each MPCitH run, once per dimension $k \in [D]$.

3.2 Leaf witness share generation

For SDitH [FJR22] the master seed is expanded into N party seeds. The witness shares for parties $1, \dots, N-1$ are then generated by expanding these seeds into random $[\mathbf{x}]$, $[\mathbf{Q}]$, and $[\mathbf{P}]$ in their respective domains. And the shares for party N are defined to be the difference between the sum of the random shares for parties $1, \dots, N-1$, and the witness $\mathbf{x}, \mathbf{Q}, \mathbf{P}$.

In our protocol, it is necessary to generate N^D leaf seeds, from which the polynomial shares and other randomness (e.g. Beaver triple shares) are generated. In practice, this part

Algorithm 1 ZK proof from Syndrome Decoding on a hypercube in the head

Input: Both parties have $H = (H'|I_{m-k}) \in \mathbb{F}_{\text{SD}}^{(m-k) \times m}$ and the syndrome $y \in \mathbb{F}_{\text{SD}}^{(m-k)}$.
The prover knows $x \in \mathbb{F}_{\text{SD}}^m$ with $y = Hx$ and $\text{wt}(x) \leq w$.

Round 1 (Computation of witness):

1. Choose $E \subset [m]$ such that $|E| = w$ and the non-zero coordinates of x are in $|E|$.
2. Compute $Q(X) = \prod_{i \in E} (X - \gamma_i) \in \mathbb{F}_{\text{poly}}(X)$.
3. Compute $S(X) \in \mathbb{F}_{\text{poly}}(X)$ by interpolation over the coordinates of x .
4. Compute $P(X) = S(X) \cdot Q(X) / F(X)$ with $F(X) \in \mathbb{F}_{\text{poly}}(X)$ s.t. $F(X) = \prod_{i=1}^m (X - \gamma_i)$.
5. Sample a root seed: $\text{seed} \leftarrow \{0, 1\}^\lambda$.
6. Expand root seed seed_i recursively using TreePRG to obtain N^D leafs and $(\text{seed}_{i'}, \rho_{i'})$
7. Initialize each main party share to zero: The index of a party is $(k, j) \in [1, \dots, D] \times [1, \dots, N]$ and contains all leaf parties whose k -th coordinate is j
- for** each party $(k, j) \in [1, \dots, D] \times [1, \dots, N]$ **do**
Set $\llbracket \mathbf{x}_A \rrbracket_{(k,j)}$, $\llbracket \mathbf{Q} \rrbracket_{(k,j)}$, $\llbracket \mathbf{P} \rrbracket_{(k,j)}$, $\llbracket \mathbf{a} \rrbracket_{(k,j)}$, $\llbracket \mathbf{b} \rrbracket_{(k,j)}$, and $\llbracket \mathbf{c} \rrbracket_{(k,j)}$ to zero.
8. Generate polynomial shares (at leaf level):
for each leaf $i' \in [1, \dots, N^D]$ **do**
 if $i' \neq N^D$ **then**
 $\{\llbracket \mathbf{a} \rrbracket_{i'}, \llbracket \mathbf{b} \rrbracket_{i'}, \llbracket \mathbf{c} \rrbracket_{i'}\} \leftarrow \text{PRG}(\text{seed}_{i'})$, $(\llbracket \mathbf{x}_A \rrbracket_{i'}, \llbracket \mathbf{Q} \rrbracket_{i'}, \llbracket \mathbf{P} \rrbracket_{i'}) \leftarrow \text{PRG}(\text{seed}_{i'})$
 $\text{state}_{i'} = \text{seed}_{i'}$
 else
 $\llbracket \mathbf{a} \rrbracket_{N^D}, \llbracket \mathbf{b} \rrbracket_{N^D} \leftarrow \text{PRG}(\text{seed}_{N^D})$, $\llbracket \mathbf{c} \rrbracket_{N^D} = \langle \mathbf{a}, \mathbf{b} \rangle - \sum_{i' \neq N^D} \llbracket \mathbf{c} \rrbracket_{i'}$
 $\llbracket \mathbf{x}_A \rrbracket_{N^D} = x_A - \sum_{i' \neq N^D} \llbracket \mathbf{x}_A \rrbracket_{i'}$
 $\llbracket \mathbf{Q} \rrbracket_{N^D} = \mathbf{Q} - \sum_{i' \neq N^D} \llbracket \mathbf{Q} \rrbracket_{i'}$, $\llbracket \mathbf{P} \rrbracket_{N^D} = \mathbf{P} - \sum_{i' \neq N^D} \llbracket \mathbf{P} \rrbracket_{i'}$,
 $\text{aux} = (\llbracket \mathbf{x}_A \rrbracket_{N^D}, \llbracket \mathbf{Q} \rrbracket_{N^D}, \llbracket \mathbf{P} \rrbracket_{N^D}, \llbracket \mathbf{c} \rrbracket_{N^D})$, and $\text{state}_{N^D} = \text{seed}_{N^D} \parallel \text{aux}$
 add the leaf party's shares to the corresponding main party share and represent the leaf party by its index on the hypercube $i' = (i_1 \dots i_D)$, where $i_k \in [1, \dots, N]$
 for each main party index p in $\{(1, i_1), (2, i_2), \dots, (D, i_D)\}$ **do**
 $\llbracket \mathbf{x}_A \rrbracket_p += \llbracket \mathbf{x}_A \rrbracket_{i'}$, $\llbracket \mathbf{Q} \rrbracket_p += \llbracket \mathbf{Q} \rrbracket_{i'}$, and $\llbracket \mathbf{P} \rrbracket_p += \llbracket \mathbf{P} \rrbracket_{i'}$
 $\llbracket \mathbf{a} \rrbracket_p += \llbracket \mathbf{a} \rrbracket_{i'}$, $\llbracket \mathbf{b} \rrbracket_p += \llbracket \mathbf{b} \rrbracket_{i'}$, and $\llbracket \mathbf{c} \rrbracket_p += \llbracket \mathbf{c} \rrbracket_{i'}$
10. leaf parties commit to their state $\mathbf{com}_{i'} = \text{Com}(\text{state}_{i'}, \rho_{i'})$.
11. Compute $h = \text{Hash}(\mathbf{com}_1, \dots, \mathbf{com}_{N^D})$ and send to the verifier

Round 2 (Get evaluation points):

The verifier picks t challenge points, which we denote as vectors $\mathbf{r} \in \mathbb{F}_{\text{points}}^t$ and $\boldsymbol{\epsilon} \in \mathbb{F}_{\text{points}}^t$, and sends $(\mathbf{r}, \boldsymbol{\epsilon})$ to the prover.

Round 3: For each axis $k \in [1, \dots, D]$ execute Algorithm 2 between the main parties $(k, 1), \dots, (k, N) \rightarrow (\llbracket \boldsymbol{\alpha} \rrbracket_k, \llbracket \boldsymbol{\beta} \rrbracket_k, \llbracket \mathbf{v} \rrbracket_k)$. Prover builds hash $h' = \text{Hash}(H_1, \dots, H_D)$ where $H_k \leftarrow \text{Algorithm2}(\llbracket \mathbf{x}_A \rrbracket, \llbracket \mathbf{Q} \rrbracket, \llbracket \mathbf{P} \rrbracket, \llbracket \mathbf{a} \rrbracket, \llbracket \mathbf{b} \rrbracket, \llbracket \mathbf{c} \rrbracket, \mathbf{r}, \boldsymbol{\epsilon})$ and sends h' to the verifier.

Round 4: Verifier uniformly picks $(i_1^*, \dots, i_D^*) \leftarrow [1, \dots, N]^D$ and sends it to prover.

Round 5: Prover sends $(\text{state}_{i_1, \dots, i_D}, \rho_{i_1, \dots, i_D}) \forall (i_1, \dots, i_D) \neq (i_1^*, \dots, i_D^*)$.

Prover also sends $\mathbf{com}_{(i_1^*, \dots, i_D^*)}$, $\llbracket \boldsymbol{\alpha} \rrbracket_{(i_1^*, \dots, i_D^*)}$, $\llbracket \boldsymbol{\beta} \rrbracket_{(i_1^*, \dots, i_D^*)}$

Verification: Verifier accepts if and only if:

1. For each $i' \neq i^*$, expand all states to get leaf party states (they have $D \log N$ seeds in the sibling path, and each of these is expanded down to the leaf party level, giving $N^D - 1$ leaves), and use \mathbf{com}_{i^*} provided. Then compute h and verify that it is equal to the one from Step 12, where $h = \text{Hash}(\mathbf{com}_1, \dots, \mathbf{com}_{i^*}, \dots, \mathbf{com}_{N^D})$
2. For $(k \in [1, \dots, D])$: Run Alg. 3 to get $\llbracket \boldsymbol{\alpha} \rrbracket, \llbracket \boldsymbol{\beta} \rrbracket, \llbracket \mathbf{v} \rrbracket$, and each of the H_k and check that:
 - (a) $\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{v}$ is the same for all D runs of Algorithm 3.
 - (b) $H = \text{Hash}(H_1, \dots, H_D)$ agrees with h' provided in Round 3.

Algorithm 2 Execute Π on a full set of parties**Input:** $[\mathbf{x}_A], [\mathbf{Q}], [\mathbf{P}], [\mathbf{a}], [\mathbf{b}], [\mathbf{c}], r, \epsilon$.**Output:** $[\alpha], [\beta], [\mathbf{v}], H$ Parties locally set $[\mathbf{x}_B] = \mathbf{y} - \mathbf{H}'[\mathbf{x}_A]$.Parties locally compute $[\mathbf{S}]$ via interpolation of $[\mathbf{x}] = ([\mathbf{x}_A] \parallel [\mathbf{x}_B])$.Compute $[\alpha], [\beta], [\mathbf{v}]$ coordinate-wise:**for** $l \in [t]$ **do**Parties locally evaluate $[\mathbf{S}(r_l)], [\mathbf{Q}(r_l)], [\mathbf{P}(r_l)]$.Parties set $[\alpha_l] = \epsilon_l [\mathbf{Q}(r_l)] + [\mathbf{a}_l]$.Parties set $[\beta_l] = [\mathbf{S}(r_l)] + [\mathbf{b}_l]$.Parties open $[\alpha_l]$ and $[\beta_l]$ to get α_l, β_l .

Parties locally set

$$[\mathbf{v}_l] = -[\mathbf{c}_l] + \langle \epsilon_l F(r_l) \cdot [\mathbf{P}(r_l)] \rangle + \langle \alpha_l, [\mathbf{b}_l] \rangle + \langle \beta_l, [\mathbf{a}_l] \rangle - \langle \alpha_l, \beta_l \rangle.$$

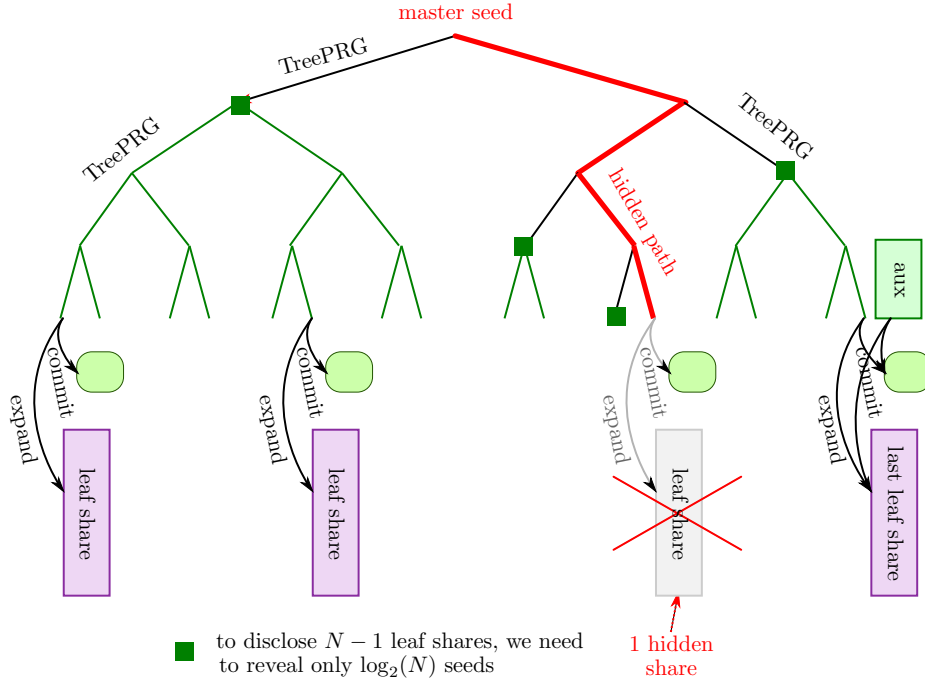
Compute $H = \text{Hash}([\alpha], [\beta], [\mathbf{v}])$ 

Fig. 2: Witness generation via seed expansion for a depth 3 tree. The N^D leaf party witness shares are derived directly from their seeds, but the $N \cdot D$ main party witness shares are defined as the sum of their leaf party shares. Subsequently To open all the leaf seeds except one, we reveal only the $\log(N^D)$ sibling nodes along the hidden path (which requires $\log(N^D)$ space).

is done identically to in [FJR22], whereby TreePRG is used to recursively expand the master seed until one has N^D leaf seeds.

As depicted in Figure 2, the master seed is expanded to generate the leaf party seeds, which are then expanded into the leaf witness shares in the canonical way. The leaf parties are indexed by $i' \in [1, \dots, N^D]$

Algorithm 3 Verify a partition of parties

Input: Secret-shares $\llbracket \mathbf{x}_A \rrbracket, \llbracket \mathbf{Q} \rrbracket, \llbracket \mathbf{P} \rrbracket, \llbracket \mathbf{a} \rrbracket, \llbracket \mathbf{b} \rrbracket, \llbracket \mathbf{c} \rrbracket, \mathbf{r}, \epsilon$. These secret-shares have already been aggregated across all disclosed leaf parties.

Index i^* and communication $\alpha, \beta, \mathbf{v}$ of the hidden party i^* .

Output: $\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket \mathbf{v} \rrbracket, H$

Parties locally set $\llbracket \mathbf{x}_B \rrbracket = \mathbf{y} - \mathbf{H}' \llbracket \mathbf{x}_A \rrbracket$.

Parties locally compute $\llbracket \mathbf{S} \rrbracket$ via interpolation of $\llbracket \mathbf{x} \rrbracket = (\llbracket \mathbf{x}_A \rrbracket \parallel \llbracket \mathbf{x}_B \rrbracket)$.

Compute $\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket \mathbf{v} \rrbracket$ coordinate-wise:

for $l \in [t]$ **do**

if Party does not contain hidden leaf party i^* **then**

 Parties locally evaluate $\llbracket \mathbf{S}(r_l) \rrbracket, \llbracket \mathbf{Q}(r_l) \rrbracket, \llbracket \mathbf{P}(r_l) \rrbracket$.

 Parties set $\llbracket \alpha_l \rrbracket = \epsilon_l \llbracket \mathbf{Q}(r_l) \rrbracket + \llbracket a_l \rrbracket$.

 Parties set $\llbracket \beta_l \rrbracket = \llbracket \mathbf{S}(r_l) \rrbracket + \llbracket b_l \rrbracket$.

 The party that contains the hidden leaf party i^* adds the contribution α_l, β_l

 Parties open $\llbracket \alpha_l \rrbracket$ and $\llbracket \beta_l \rrbracket$ to get α_l, β_l .

 Parties locally set

$$\llbracket v_l \rrbracket = -\llbracket c_l \rrbracket + \langle \epsilon_l F(r_l) \cdot \llbracket P(r_l) \rrbracket \rangle + \langle \alpha_l, \llbracket b_l \rrbracket \rangle + \langle \beta_l, \llbracket a_l \rrbracket \rangle - \langle \alpha_l, \beta_l \rangle.$$

else

 The party that contains the hidden leaf party i^* sets $\llbracket v_l \rrbracket_{i^*}$ so that $v_l = 0$

 Compute $H = \text{Hash}(\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket \mathbf{v} \rrbracket)$

3.3 Leaf witness shares on a hypercube

A geometric mapping is necessary in order to manipulate the results in the hypercube setting described in Section 3.1. Section 3.2 explained how to output N^D leaf parties and their witness shares. To arrange them on a hypercube, we rewrite the index $i' \in [1, \dots, N^D]$ equivalently as $i' = (i_1, \dots, i_D)$ where each $i_k \in [1, \dots, N]$.

To reveal the entire hypercube, except for a single leaf party, it is enough to reveal the sibling path of the hidden leaf party. The verifier (who will eventually receive $N^D - 1$ leaf nodes) can reconstruct the hypercube geometry themselves, using the same indexing convention as the signer.

3.4 Main party witness shares

To construct the witness shares of the main parties in dimension k , one aggregates the shares of all leaf parties (i_1, \dots, i_D) which share the same index i_k . E.g., in dimension 1, the share of \mathbf{Q} of the j^{th} main party, denoted $(1, j)$, would be $\llbracket \mathbf{Q} \rrbracket_{(1, j)} = \sum_{i_2, \dots, i_D} \llbracket \mathbf{Q} \rrbracket_{(j, i_2, \dots, i_D)}$, which is a sum over N^{D-1} of the leaf party shares of \mathbf{Q} . Figure 1 is helpful to visualize how this aggregation is performed.

One can consider that the following high-level flow is used to generate and ultimately aggregate the shares in order to generate the main party shares. On the left hand side the TreePRG is used as a compression technique; in the middle, the leaf seeds are expanded into shares and arranged in a hypercube geometry; on the right the shares are aggregated in order to provide the MPCitH inputs. It is helpful to think of the TreePRG compression and the hypercube arrangement/aggregation as separate techniques, which are combined here for the purposes of generating efficient signatures.

$$\text{seed} \xrightarrow{\text{TreePRG}} \{\text{seed}_{i'}\}_{i' \in [N^D]} \xrightarrow{\text{PRG}} \{\llbracket \mathbf{x} \rrbracket_{i'}, \llbracket \mathbf{P} \rrbracket_{i'}, \llbracket \mathbf{Q} \rrbracket_{i'}\}_{i' \in [N^D]} \xrightarrow{\Sigma} \llbracket \mathbf{x} \rrbracket_k, \llbracket \mathbf{P} \rrbracket_k, \llbracket \mathbf{Q} \rrbracket_k,$$

3.5 Proofs of security

The proofs of this section closely follow those set out in [FJR22] due to the similarity of the underlying hardness assumptions. Protocol 1 implicitly defines the interaction between an *honest prover* that executes the odd rounds 1,3,5 and a *honest verifier* who executes the even rounds 2,4. Throughout the security proof, a general prover, who does not necessarily know the secret, is a party that reads and produces the same type of messages as the honest prover, without necessarily following the algorithm.

We first show that an honest prover is accepted with certainty, and conversely, any prover who commits to a bad witness that does not encode the SD secret in the first round has probability lower than $\epsilon \approx 1/N^D$ of being accepted. Consequently, any prover that has a higher rate of acceptance necessarily knows the secret. Then, we prove that the protocol is zero knowledge, since its transcript distribution can be simulated without the secret.

Theorem 1 ((Perfect) Completeness). *Protocol 1 is perfectly complete. That is to say, a prover with knowledge of a witness w (contained in sk) who performs $\mathcal{P}(sk)$ correctly, will be accepted by a verifier $\mathcal{V}(pk)$ with probability 1.*

Proof. Proof of Theorem 1 For any choice of randomness for \mathcal{P}, \mathcal{V} , the computations of \mathcal{P} pass all of the the verification checks of \mathcal{V} by construction. \square

Lemma 4. *A prover $\tilde{\mathcal{P}}$ that commits to a bad witness s.t. $\mathbf{S} \cdot \mathbf{Q} \neq \mathbf{P} \cdot \mathbf{F}$ in Round 1 of protocol 1 and is unable to find a commitment/hash collision has probability $\leq \epsilon = (p + (1 - p)/N^D)$ of being accepted by an honest verifier \mathcal{V} .*

Proof of Lemma 4. For \mathcal{V} to accept, given $\mathbf{S} \cdot \mathbf{Q} \neq \mathbf{P} \cdot \mathbf{F}$, one of two scenarios must occur:

1. the random value which $\llbracket \mathbf{v} \rrbracket$ encodes happens to be zero with probability p , or otherwise.
2. $\tilde{\mathcal{P}}$ must cheat on the communications he sends, which correspond to the MPCitH protocols on the main parties, so that it *appears* that the resulting \mathbf{v} is the zero vector.

After the initial commitment, \mathcal{V} sends the challenge points \mathbf{r}, ϵ . In the first scenario, with probability p the plaintext vector \mathbf{v} generated by the MPC protocol is the zero vector (i.e. on all t points, it happens to be the case that $\delta = (\mathbf{S} \cdot \mathbf{Q} - \mathbf{P} \cdot \mathbf{F})(\mathbf{r})$ is zero, and/or that the beaver triplets committed in round 1 satisfy $\mathbf{c} - \mathbf{a}\mathbf{b} = \epsilon\delta$.)

However, with probability $(1 - p)$, at *at least* one of the challenge points, $\mathbf{S} \cdot \mathbf{Q}(r_i) \neq \mathbf{P} \cdot \mathbf{F}(r_i)$, meaning at least one of the coordinates of $\mathbf{v} = \mathbf{c} - \mathbf{a}\mathbf{b} - \epsilon\delta$ is non zero. In this case, the communications $\llbracket \boldsymbol{\alpha} \rrbracket, \llbracket \boldsymbol{\beta} \rrbracket, \llbracket \mathbf{v} \rrbracket$ resulting from an honest execution will not be accepted therefore $\tilde{\mathcal{P}}$ must alter some communications so that the resultant \mathbf{v} is the zero vector.

In Round 3, $\tilde{\mathcal{P}}$ commits to his communications to D independent SDitH runs (one for each dimension on the hypercube). Let us assume that he needs to cheat on the communications of a single run (out of D), and without loss of generality, this can be cheating on the shares of $\boldsymbol{\alpha}$ (cheating on $\boldsymbol{\beta}$ or \mathbf{v}) are equally valid).

The $\llbracket \boldsymbol{\alpha} \rrbracket$ in this dimension consist of N main party shares $\llbracket \boldsymbol{\alpha} \rrbracket_i$. So $\tilde{\mathcal{P}}$ must pick one to cheat on, having $1/N$ chance of success. Each of the main party shares consists of the sum of $N^{(D-1)}$ leaf shares in that particular slice, and all but one of the leaf shares will be opened. Therefore $\tilde{\mathcal{P}}$ must cheat on the share $\llbracket \boldsymbol{\alpha} \rrbracket$ of a single leaf party s , shifting its value by $\delta \neq 0$. Cheating on more than one leaf party means certain detection as all but one leaf parties are opened, and cheating on none means that \mathbf{v} is not the zero vector so won't be accepted.

However, leaf party s belongs to a single main share for each run of SDitH (one for each dimension of the hypercube). In each of these other main shares, their value for $\llbracket \boldsymbol{\alpha} \rrbracket$ must

be shifted by the same δ , as they cannot offset this value using other leaf parties, as all but one leaf party is revealed in Round 5 so this would mean certain detection. Thus each main share to which s belongs must cheat in their respective SDitH. No other cheating pattern is possible, because all leaf parties bar one are revealed in Round 5, so only one leaf party can cheat by δ , and this is exhibited in one main party for each dimension.

The only way to avoid detection using this method, is if the (uniformly random) challenge i^* in Round 4 gives the exact coordinates of s , as this means the main party to which s belongs in each dimension is the one that remains hidden. This has probability $(1/N)^D$, and is equivalent to the challenge leaving hidden the exact leaf party s out of N^D leaf parties. Hence, in a non-false positive scenario, $\tilde{\mathcal{P}}$ has $\leq 1/N^D$ chance of cheating. This yields the bound $p + (1 - p)/N^D$ for the prover to be accepted using a bad witness in round 1. \square

Theorem 2 (Soundness). *If an efficient prover $\tilde{\mathcal{P}}$ with knowledge of only (\mathbf{H}, \mathbf{y}) can convince verifier \mathcal{V} with probability*

$$\tilde{\epsilon} = \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle \rightarrow 1] > \epsilon = \left(p + (1 - p) \frac{1}{N^D} \right), \quad (10)$$

where p is bounded in Equation 9, then there exists an extraction function E that produces a commitment collision, or a good witness \mathbf{x}' such that $\mathbf{H}\mathbf{x}' = \mathbf{y}$ and $wt(\mathbf{x}') < w$ by making an average number of calls to $\tilde{\mathcal{P}}$ is bounded from above:

$$\frac{4}{\tilde{\epsilon} - \epsilon} \cdot \left(1 + \frac{2\tilde{\epsilon} \ln 2}{\tilde{\epsilon} - \epsilon} \right) \quad (11)$$

Should a prover $\tilde{\mathcal{P}}$ cheat with probability $p \leq \epsilon$ then this is not an issue, as it corresponds to ordinary vanilla cheating, i.e. cheating on a particular node, hoping that node does not have to be revealed at challenge time, or by hoping to guess some polynomials $\mathbf{S}, \mathbf{Q}, \mathbf{P}, \mathbf{F}$ which do not satisfy $\mathbf{S} \cdot \mathbf{Q} = \mathbf{P} \cdot \mathbf{F}$ in general, but which are equal at the challenge points which are subsequently selected.

Sketch of proof of Theorem 2: The proof largely follows the soundness proof for the original SDitH [FJR22] protocol. The main difference lies in the details of witness extraction. More specifically, in the argument why we can extract. In our case, we are running D instances of SDitH in parallel. For each of these instances, the state of each party is secret shared. These secret shares are arranged in a hypercube, such that every share is used as a secret share of D different instances. The first message contains a commitment to each of these secret shares.

Regarding extraction we prove (as for SDitH) that we can extract a candidate witness (an \mathbf{x} s.t. $\mathbf{H}\mathbf{x} = \mathbf{y}$) as soon as we see two accepting transcripts that agree on the commitments, i.e., the first message, but disagree on the second challenge. As we always open all but one commitment, and this second challenge that decides which commitment not to open differs for the two transcripts, we learn the openings of all commitments (assuming that the commitment scheme is binding). It remains to argue that this is sufficient for extraction.

The soundness proof for the original SDitH protocol also shows that a candidate witness can be extracted from two accepting transcripts that share the same commitments but differ in the second challenge. This does not immediately imply extraction in our case as we committed to secret shares of the state and communications of the parties. However, we can rephrase the extraction condition shown for SDitH as the following: Extraction is possible given the opened state and communication for *all* parties, so long as each party is verified in at least one accepting transcript. As only one commitment is not opened per transcript, there is the state and communication of exactly one party per SDitH proof that is not verified in each transcript.

As the second challenges differ between the two transcripts per assumption, there has to be *at least* one dimension, in which the unopened leaf party secret shares belong to different main parties. In this dimension, we have obtained the openings of all main parties. Furthermore, in this dimension, the state and communication of each main party was verified during the verification of at least one of the transcripts. Therefore, we can apply the extraction argument of the original SDitH protocol. Equivalently, one now has knowledge of all leaf parties which together represent a complete sharing of the witness, and by the argument above, all leaf parties have been verified in at least one transcript. It remains to show that the candidate witness is a good witness, i.e., has $wt(\mathbf{x}) < w$. This follows the same argument from SDitH proof.

Proof of Theorem 2. Assume the commitment scheme is perfectly binding (as opposed to computationally binding), as per Definition 8. For two sets of transcripts with the same initial commitment $h = \text{Hash}(\mathbf{com}_1, \dots, \mathbf{com}_{N^D})$, but different challenge leaf parties $i^* \neq j^*$, either:

- $\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{Q} \rrbracket, \llbracket \mathbf{P} \rrbracket$ differ and one finds a collision in the commitment hash, or
- the openings are equal in both transcripts, and therefore the shares $\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{Q} \rrbracket$, and $\llbracket \mathbf{P} \rrbracket$ are also equal in both transcripts.

In the second case, the witness can be recovered from two transcripts with $i^* \neq j^*$ where $i^*, j^* \in [1, \dots, N^D]$ are the challenge indices in the first and second transcripts respectively. This is because in the first case the verifier receives the $N^D - 1$ leaf parties which are not i^* , and in the second transcript they receive the $N^D - 1$ leaf parties which are not j^* . Thus with both transcripts, the verifier knows the full set of witness shares and so can reconstruct the full witness by summing all of the N^D leaf party shares.

Now we explain why this means that the extractor function is able to learn a good witness. Firstly, consider the hypercube geometry: $i^* \neq j^*$ means that their coordinates in the hypercube are not equal in *at least* one position $(i_1^*, \dots, i_D^*) \neq (j_1^*, \dots, j_D^*)$. Let the first coordinate in which they differ be $i_k^* \neq j_k^*$. Then for the MPCitH protocol for dimension k , one has two transcripts with different hidden (main) parties, where the sum of witness shares for both runs has been successfully verified. This scenario almost identically resembles the protocol of [FJR22], thus the remainder of the proof of soundness proceeds in the same manner.

In the following we demonstrate that to generate two such accepted transcripts with the same initial commitment but different challenge points, the witness must be good. Call $\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{Q} \rrbracket$, and $\llbracket \mathbf{P} \rrbracket$ a good witness if

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

R_h is the random variable for the randomness used to generate the initial commitment, with r_h being a given value of R_h .

The extractor works by simple application of the Forking lemma, lemma 2: $\tilde{\mathcal{P}}$ is run with honest \mathcal{V} until successful transcript T_1 is found, having second challenge i^* . Then rewind $\tilde{\mathcal{P}}$, using the same randomness r_h as in T_1 until one gets a successful transcript T_2 with different second challenge j^* . Then extract the witness. If the witness is bad, start over.

Next we estimate how many calls to $\tilde{\mathcal{P}}$ the extractor E makes. Let $\alpha \in (0, 1)$ such that $(1 - \alpha) \cdot \tilde{\epsilon} > \epsilon$. We say r_h is ‘good’ if $\Pr[\text{succ}_{\tilde{\mathcal{P}}}|r_h] \geq (1 - \alpha) \cdot \tilde{\epsilon}$. By the splitting lemma (Lemma 1), $\Pr[r_h \text{ is good} | \text{succ}_{\tilde{\mathcal{P}}}] \geq \alpha$, which implies that a good randomness can be found after gathering roughly $1/\alpha$ successful transcripts. Also, by (the converse of) Lemma 4, when

r_h is good, since the probability $(1 - \alpha)\tilde{\epsilon} > \epsilon$, then the initial commitment provided by the transcript necessarily encodes a good witness, that can be extracted from any other successful transcript that starts from r_h .

Given a good transcript T_1 (i.e. a success in the outer loop) we now provide a lower bound on the number of iterations of the inner loop in order to find another good transcript T_2 with the same randomness r_h such that $i^* \neq j^*$.

$$\begin{aligned}
 \Pr[\text{succ}_{\tilde{\mathcal{P}}} \cap i^* \neq j^* | r_h \text{ good}] &= \Pr[\text{succ}_{\tilde{\mathcal{P}}} | r_h \text{ good}] - \Pr[\text{succ}_{\tilde{\mathcal{P}}} \cap i^* = j^* | r_h \text{ good}] \\
 &\geq \Pr[\text{succ}_{\tilde{\mathcal{P}}} | r_h \text{ good}] - \frac{1}{N^D} \\
 &\geq (1 - \alpha)\tilde{\epsilon} - \frac{1}{N^D} \\
 &\geq (1 - \alpha)\tilde{\epsilon} - \epsilon.
 \end{aligned} \tag{12}$$

Then by running $\tilde{\mathcal{P}}$ for L repetitions one has a probability greater than $1/2$ of obtaining a second transcript T_2 with a different challenge leaf party than T_1 , where both T_1 and T_2 are generated using the same (good) randomness r_h , where

$$L > \frac{\ln 2}{\ln \frac{1}{1 - ((1 - \alpha)\tilde{\epsilon} - \epsilon)}} \simeq \frac{\ln 2}{(1 - \alpha)\tilde{\epsilon} - \epsilon}. \tag{13}$$

Denote the expected number of calls to $\tilde{\mathcal{P}}$ as $\mathbb{E}(\tilde{\mathcal{P}})$. Then $\mathbb{E}(\tilde{\mathcal{P}})$ can be written as a recursive formula; as a function of firstly the probability of succeeding in the outer loop to obtain T_1 , and secondly the probability of obtaining T_2 with L calls once one has found a successful transcript T_1 . Step by step, this is

1. Make an initial call to $\tilde{\mathcal{P}}$.
2. If we do not find T_1 , with probability $(1 - \Pr[\text{succ}_{\tilde{\mathcal{P}}})]$, then repeat the procedure from Step 1.
3. If we find a successful T_1 , then r_h is good with probability α by the splitting lemma (Lemma 1). Then make L calls to $\tilde{\mathcal{P}}$, after which there is probability above $1/2$ of success. If successful, terminate, else return to Step 1.
4. The probability that r_h is bad is $1 - \alpha$. Thus, there is no guarantee on the probability of finding T_2 . Make L calls to $\tilde{\mathcal{P}}$ (because we do not yet know that r_h is bad), then when unsuccessful, return to Step 1.

Consequently, if a call in Step 1 to $\tilde{\mathcal{P}}$ does not yield T_1 , then repeat Step 1. If Step 1 is successful, giving T_1 , then we perform L further calls seeking to obtain T_2 , because we do not know a priori whether r_h is good or bad. If r_h is good (with probability α), then there is $1/2$ probability that we find T_2 and the algorithm terminates. With r_h good, there is also $1/2$ probability that we do not find T_2 . If r_h is bad (with probability $1 - \alpha$), there is no guarantee about finding T_2 so to provide an upper bound for the number of calls to $\tilde{\mathcal{P}}$ we say that this part is always unsuccessful at finding T_2 . Thus

$$\Pr[\text{no } T_2 | \text{succ}_{\tilde{\mathcal{P}}}] = \Pr[\text{no } T_2 | r_h \text{ good}] + \Pr[\text{no } T_2 | r_h \text{ bad}] = \alpha/2 + (1 - \alpha),$$

and in this case return to Step 1. So the expression for $\mathbb{E}(\tilde{\mathcal{P}})$ can be written

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq 1 + \underbrace{(1 - \Pr[\text{succ}_{\tilde{\mathcal{P}}})]}_{\text{Do not find } T_1} \mathbb{E}(\tilde{\mathcal{P}}) + \underbrace{\Pr[\text{succ}_{\tilde{\mathcal{P}}}] \left(L + \left(1 - \frac{\alpha}{2} \right) \mathbb{E}(\tilde{\mathcal{P}}) \right)}_{\text{Find } T_1}, \tag{14}$$

which reduces to

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq \frac{2}{\alpha\tilde{\epsilon}} \left(1 + \tilde{\epsilon}L\right) = \frac{2}{\alpha\tilde{\epsilon}} \left(1 + \frac{\tilde{\epsilon} \ln 2}{(1-\alpha)\tilde{\epsilon} - \epsilon}\right). \quad (15)$$

Define $(1-\alpha)\tilde{\epsilon} = \frac{1}{2}(\epsilon + \tilde{\epsilon})$, i.e. halfway between ϵ and $\tilde{\epsilon}$ in order to obtain a formula in terms of just ϵ and $\tilde{\epsilon}$. Then we arrive at the upper bound

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq \frac{4}{\tilde{\epsilon} - \epsilon} \left(1 + \frac{2\tilde{\epsilon} \ln 2}{\tilde{\epsilon} - \epsilon}\right). \quad (16)$$

□

We now prove that the protocol is zero-knowledge. The main intuition is that any prover who learns the challenge points \mathbf{r}, ϵ from Round 2 challenge before committing to the state on Round 1 can update c in the aux to force a false positive. Similarly any prover who learns the challenge coordinates i^* from Round 4 before committing to the MPC communications on Round 3 can alter the communication of the hidden party such that \mathbf{v} becomes the zero vector. The following simulator exploits the second option.

Theorem 3 (Honest-Verifier Zero Knowledge (HVZK)). *If the PRG of Algorithm 1 and commitment Com are indistinguishable from the uniform random distribution, then Algorithm 1 is Honest-Verifier Zero Knowledge.*

Proof of HVZK. To prove the HVZK property, we construct a simulator \mathcal{S} which outputs transcripts of Algorithm 1 which are computationally indistinguishable from real transcripts. For this we assume that the PRG of Algorithm 1 is $(t, \epsilon_{\text{PRG}})$ -secure and the commitment Com is $(t, \epsilon_{\text{Com}})$ -hiding. For ease of reading, in the following, we sometimes denote general leaf party indices $(i_{k_1}, \dots, i_{k_D})$ by i' , and the challenge party index (i_1^*, \dots, i_D^*) as simply i^* .

First consider a simulator, \mathcal{S} , described in Algorithm 4 which produces the transcript responses (COM, CH₁, RSP₁, CH₂, RSP₂):

Next we demonstrate that this simulator produces indistinguishable transcripts from the distribution of real transcripts by starting with a simulator that produces ‘true’ transcripts, and altering the outputs section-by-section until arriving at \mathcal{S} defined above. At each simulator alteration we argue why the distribution remains unchanged.

True transcripts (v0): This takes as input a witness \mathbf{x}_A as well as the honest verifier’s challenges $(\mathbf{r}, \epsilon, i^*)$. It then executes Algorithm 1 correctly, hence its output distribution is the ‘correct’ distribution.

Simulator v1: In this simulator, the only difference versus v0 is that randomness in leaf party i^* is replaced with true randomness. If $i^* = (N, \dots, N)$ then $[\mathbf{x}_A]_{N^D}$, $[\mathbf{Q}]_{N^D}$, and $[\mathbf{P}]_{N^D}$ are generated in the usual way. So the witness shares of all leaf parties still sum to give the input witness (and by extension, all parties for each MPCitH run in $[1, \dots, D]$), therefore only $[\mathbf{a}]_{N^D}$ and $[\mathbf{b}]_{N^D}$ are random (and by extension, so are the shares $[\mathbf{a}], [\mathbf{b}]$ for the D parties $[(1, N), \dots, (D, N)]$ which contain challenge leaf party $i^* = N^D$). We can see that the difficulty in distinguishing the output of Simulator v1 from the real distribution is equal to distinguishing ϵ_{PRG} from true randomness.

Simulator v2: Replace $[\mathbf{x}_A]_{N^D}$, $[\mathbf{Q}]_{N^D}$, $[\mathbf{P}]_{N^D}$, and $[\mathbf{c}]_{N^D}$ with true randomness (i.e. sample these shares randomly, and not via the protocol). This means that $[\mathbf{x}_A]$, $[\mathbf{Q}]$, and $[\mathbf{P}]$ are now independent of input witness, so the inputs to \mathcal{S} are reduced to the challenges (ch₁, ch₂).

For $i^* = N^D$ this means that only $[\alpha]_{i^*}, [\beta]_{i^*}$ are affected because in this scenario *aux* is not sent in RSP₂. These shares do not change in distribution from Simulator v1 to Simulator

Algorithm 4 HVZK simulator

Sample seed $\leftarrow \mathbb{S} \{0, 1\}^\lambda$.

Generate $(seed_{i'}, \rho_{i'})$ for all leaf parties via TreePRG(seed).

Step 1: Sample challenges

- CH1 = $\{\mathbf{r}, \epsilon\} \leftarrow \mathbb{F}_{\text{points}}^t \times \mathbb{F}_{\text{points}}^t$
- CH2 = $i^* \leftarrow [1, \dots, N^D]$

Step 2: generate N^D leaf party states

Expand root seed $seed_i$ recursively using TreePRG to obtain N^D leaf states and randomness $(seed_{i'}, \rho_{i'})$

Step 3: generate leaf party commitments and witness shares

for $i' \neq i^*$ **do**

 Compute $\mathbf{com}_{i'} = \text{Hash}(\text{state}_{i'}, \rho_{i'})$

if $i' \neq N^D$ **then**

 Expand the leaf party seeds into witness shares

else

 To generate aux for the last leaf party, $i' = N^D$, randomly draw $\llbracket \mathbf{x}_A \rrbracket_{N^D}$, $\llbracket \mathbf{Q} \rrbracket_{N^D}$, $\llbracket \mathbf{P} \rrbracket_{N^D}$, and $\llbracket \mathbf{c} \rrbracket_{N^D}$.

for $i' = i^*$ **do**

 Draw \mathbf{com}_{i^*} at random

Compute initial commitment $\text{COM} = \text{Hash}(\mathbf{com}_1, \dots, \mathbf{com}_{i^*}, \dots, \mathbf{com}_{N^D})$

Step 4: generate party communications

Draw $\llbracket \alpha \rrbracket_{i^*}$ and $\llbracket \beta \rrbracket_{i^*}$ uniformly at random from their respective domains.

for $k \in [1, \dots, D]$ **do**

for $i_k \neq i_k^*$ **do**

 Get communications $\{\llbracket \alpha \rrbracket_{i_k}, \llbracket \beta \rrbracket_{i_k}, \llbracket \mathbf{v} \rrbracket_{i_k}\}$ as stated in Algorithms 1, 2

for i_k^* **do**

 Compute party communication shares $\llbracket \alpha \rrbracket_{i_k^*}, \llbracket \beta \rrbracket_{i_k^*}, \llbracket \mathbf{v} \rrbracket_{i_k^*}$ by running Π on the sum of the witnesses of the $N - 1$ revealed leaf parties in their respective slices, as described in Algorithm 1, then add on $\llbracket \alpha \rrbracket_{i^*}$ and $\llbracket \beta \rrbracket_{i^*}$

 Set $\llbracket \mathbf{v} \rrbracket_{i^*} = -\sum_{i' \neq i^*} \llbracket \mathbf{v} \rrbracket_{i'}$.

Step 5: Output transcript (COM, CH1, RSP1, CH2, RSP2):

$\text{RSP}_1 = h' = \text{Hash}(H_1, \dots, H_D)$ where $H_k \leftarrow \text{Alg. 2}(\llbracket \mathbf{x}_A \rrbracket, \llbracket \mathbf{Q} \rrbracket, \llbracket \mathbf{P} \rrbracket, \llbracket \mathbf{a} \rrbracket, \llbracket \mathbf{b} \rrbracket, \llbracket \mathbf{c} \rrbracket, \mathbf{r}, \epsilon)$

$\text{RSP}_2 = \mathbf{com}_{i^*}, \llbracket \alpha \rrbracket_{i^*}, \llbracket \beta \rrbracket_{i^*}, \{(\text{state}_{i_1, \dots, i_D}, \rho_{i_1, \dots, i_D}) \forall (i_1, \dots, i_D) \neq (i_1^*, \dots, i_D^*)\}$.

v2 because we already have in Simulator v1 the $\llbracket \alpha \rrbracket_{i^*}$ and $\llbracket \beta \rrbracket_{i^*}$ which appear to be uniformly distributed and are unaffected by the other parties, and $\llbracket \mathbf{v} \rrbracket_{i^*} = -\sum_{i \neq i^*} \llbracket \mathbf{v} \rrbracket_i$.

For $i^* \neq N^D$ only aux is affected in the transcript. In Simulator v1, aux is computed via the sum of true uniform randomness of leaf party i^* , and every other leaf party's pseudo-randomness, also generating aux via true uniform randomness does not alter the distribution between Simulators v1 and v2.

Simulator v3: In this version, the $\llbracket \alpha \rrbracket_{i^*}$ and $\llbracket \beta \rrbracket_{i^*}$ are also drawn using true randomness (affecting communications of party i^*). However these already appear to be uniformly distributed in Simulator v2, hence the output distribution does not change between Simulator v2 and v3. The outputs of Simulator v3 ($\text{RSP}_1, \text{RSP}_2$) are hence indistinguishable from those of an honest execution of Algorithm 1.

To obtain a global HVZK simulator we take the simulator described in Algorithm 4 and apply the hiding property of \mathbf{com}_{i^*} , with the final simulator performing as follows:

1. Generate random challenges ch_1, ch_2 .
2. Run Simulator v3 to get $\text{RSP}_1, \text{RSP}_2$.
3. For initial leaf party commitments $i' \neq i^*$ compute $\mathbf{com}_{i'} = \text{Com}(\text{state}_{i'}, \rho_{i'})$.

4. For leaf party i^* , draw \mathbf{com}_{i^*} at random.
5. Set initial commitment to $\text{Com} = \text{Hash}(\mathbf{com}_1, \dots, \mathbf{com}_{ND})$

The output of the global HVZK simulator is $(t, \epsilon_{\text{PRG}} + \epsilon_{\text{Com}})$ indistinguishable from the real distribution. \square

4 Signature based on Syndrome Decoding with hypercube MPCitH

We define the construction of an signature scheme based on the zero knowledge proofs outlined in Section 3. In order to do this, we describe the non-interactive transformation of the proof. The explicit construction for using many proofs in parallel to achieve sufficient soundness is also described in Algorithm 6, as well as a proof of security for the construction.

4.1 Description of the Signature scheme

A signature scheme is a tuple of algorithms (KeyGen, Sign, Verify). KeyGen generates a public key, secret key pair (pk, sk) . Sign(m, sk) takes as input a message and secret key, and returns a signature σ . Verify(m, pk, σ) takes as input a message, a public key, and a signature, and outputs *accept* deterministically if the message m has been signed with the secret key sk associated with pk , and *reject* otherwise. We apply the Fiat-Shamir transform to the 5 rounds of Algorithm 1 in order to make it non-interactive.

The use of the hypercube method for MPCitH enables one to obtain greater soundness in a single instantiation of the SDitH problem before the number of MPC parties makes the protocol computationally infeasible. This therefore means that fewer repetitions of the algorithm are required to reach the same security level.

Specifically, the expensive part of the communication is that of sending *aux* (that is the correction to the randomly generated polynomials) which ensures that the overall sum of the shares gives the correct witness. The signature that follows thus transmits fewer of the expensive parts (i.e., *aux*), allowing for smaller overall communications at a given security level.

4.2 A Non-Interactive Transformation

By application of the Fiat-Shamir transformation, we achieve non-interactivity in Rounds 1 to 5 of Algorithm 1 via the same methodology as described in [FJR22].

Explicitly, Challenges 1 and 2 are generated non-interactively by hashing the transcript up to that point, and then expanding the hash output into Challenge 1 and 2:

- A hash is generated from the initial commitments across all τ parallel repetitions:

$$h_2 = \text{Hash}_2(m, \text{salt}, \mathbf{com}^{[1]}, \dots, \mathbf{com}^{[\tau]}).$$

The challenge is then expanded via PRG(h_2) to give challenge points $\{\mathbf{r}^{[e]}, \epsilon^{[e]}\}_{e \in [\tau]}$ that make up Challenge 1.

- A hash is generated from the transcript information up to the end of Round 4:

$$h_3 = \text{Hash}_4(m, \text{salt}, h_2, \{H_1^{[e]}, \dots, H_D^{[e]}\}_{e \in [\tau]}),$$

and is subsequently expanded via PRG(h_4) to obtain challenge leaf parties $\{i^{*[e]}\}_{e \in [\tau]}$.

It is necessary at this point to consider the attack on Fiat-Shamir-transformed schemes [FS87; AAB⁺02]. The attack can be considered an improvement in the time to brute-force a transformed protocol. The key observation is that for protocols with more than one challenge of the verifier to be transformed, the prover often only needs to guess correctly one of the transformed challenges, so the security of each challenge can no longer be considered independently. This means that protocols with 5 passes or more are affected.

In the forgery attack presented by Kales and Zaverucha [KZ20], the forgery is achieved by breaking the two rounds of the protocol separately, therefore, resulting in an additive cost rather than the expected multiplicative cost. The cost associated with forging a transcript that passes the first 5 rounds of Algorithm 1 relies on finding an optimal τ' repetition rate where we guess the first challenge that results in the lowest cost:

$$\begin{aligned} \text{cost}_{\text{forge}} &:= \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + (N^D)^{\tau-\tau'} \right\} \\ &\leq \min_{0 \leq \tau' \leq \tau} \left\{ \left(\frac{1}{p} \right)^{\tau'} + (N^D)^{\tau-\tau'} \right\}, \end{aligned} \quad (17)$$

where p is the probability of a false positive \mathcal{F} given in Equation 9. The resulting $\text{cost}_{\text{forge}}$ is much lower than $1/\epsilon^\tau$, therefore the parameters are necessarily adjusted to ensure that the required security level is still met.

4.3 Key Generation

The hard problem for this signature scheme is a random syndrome decoding instance (\mathbf{H}, \mathbf{y}) which has a known solution \mathbf{x} of small weight, such that $\mathbf{H}\mathbf{x} = \mathbf{y}$ and $wt(\mathbf{x}) \leq w$. Generation of \mathbf{H} is done via expansion of a pseudorandom λ -bit seed (for security level λ). The same seed generates some random \mathbf{x} of small weight, and then \mathbf{y} is computed as the syndrome of \mathbf{x} with respect to \mathbf{H} .

Algorithm 5 Key Generation.

Input: seed $s \in \{0, 1\}^\lambda$
 $seed_H \leftarrow \text{PRG}(s)$
 $\mathbf{x} \leftarrow \text{PRG}(s)$
 $\mathbf{H} \leftarrow \text{PRG}(seed_H)$
 $\mathbf{y} = \mathbf{H}\mathbf{x}$
 $pk = (seed_H, \mathbf{y}), sk = s$
Return: (pk, sk)

For simplicity, one can write $pk = (\mathbf{H}, \mathbf{y})$ and $sk = (\mathbf{H}, \mathbf{y}, \mathbf{x})$, removing the extra seed expansion steps from subsequent descriptions.

Algorithm 6 SD-based signature MPCitH on a hypercube**Input:** Secret key $sk = (H, y, x)$, message $m \in \{0, 1\}^*$ Sample $salt \leftarrow \mathbb{F}_{2^\lambda}$ **Part 1:** Build proof witness

1. Choose $E \subset [m]$ such that $|E| = w$ and the non-zero coordinates of x are contained in $|E|$.
2. Compute $Q(X) = \prod_{i \in E} (X - \gamma_i) \in \mathbb{F}_{poly}(X)$
3. Compute $S(X) \in \mathbb{F}_{poly}(X)$ by interpolation over the coordinates of x

Part 1.1: Construct leaf party MPCitH inputs (for each repetition $e \in [\tau]$)

1. Sample a root seed: $seed \leftarrow \{0, 1\}^\lambda$
2. Generate N^D leaf seeds

Expand root seed $seed_i$ recursively using TreePRG to obtain N^D leaf states and randomness $(seed_{i'}, \rho_{i'})$ Initialize each main party share to zero: The index of a party is $(k, j) \in [1, \dots, D] \times [1, \dots, N]$ and contains all leaf parties whose k -th coordinate is j **for** each party $(k, j) \in [1, \dots, D] \times [1, \dots, N]$ **do**Set $\llbracket \mathbf{x}_A \rrbracket_{(k,j)}$, $\llbracket \mathbf{Q} \rrbracket_{(k,j)}$, $\llbracket \mathbf{P} \rrbracket_{(k,j)}$, $\llbracket \mathbf{a} \rrbracket_{(k,j)}$, $\llbracket \mathbf{b} \rrbracket_{(k,j)}$, and $\llbracket \mathbf{c} \rrbracket_{(k,j)}$ to zero.

Generate polynomial shares (at leaf level):

for each leaf $i' \in [1, \dots, N^D]$ **do****if** $i' \neq N^D$ **then**

$$\{\llbracket \mathbf{a} \rrbracket_{i'}, \llbracket \mathbf{b} \rrbracket_{i'}, \llbracket \mathbf{c} \rrbracket_{i'}\} \leftarrow \text{PRG}(salt, seed_{i'})$$

$$\{\llbracket \mathbf{x}_A \rrbracket_{i'}, \llbracket \mathbf{Q} \rrbracket_{i'}, \llbracket \mathbf{P} \rrbracket_{i'}\} \leftarrow \text{PRG}(salt, seed_{i'})$$

$$state_{i'} = seed_{i'}$$
else

$$\llbracket \mathbf{a} \rrbracket_{N^D}, \llbracket \mathbf{b} \rrbracket_{N^D} \leftarrow \text{PRG}(salt, seed_{N^D})$$

$$\llbracket \mathbf{c} \rrbracket_{N^D} = \langle \mathbf{a}, \mathbf{b} \rangle - \sum_{i' \neq N^D} \llbracket \mathbf{c} \rrbracket_{i'}$$

$$\llbracket \mathbf{x}_A \rrbracket_{N^D} = x_A - \sum_{i' \neq N^D} \llbracket x_A \rrbracket_{i'}$$

$$\llbracket \mathbf{Q} \rrbracket_{N^D} = \mathbf{Q} - \sum_{i' \neq N^D} \llbracket \mathbf{Q} \rrbracket_{i'}$$

$$\llbracket \mathbf{P} \rrbracket_{N^D} = \mathbf{P} - \sum_{i' \neq N^D} \llbracket \mathbf{P} \rrbracket_{i'}$$

$$aux = (\llbracket \mathbf{x}_A \rrbracket_{N^D}, \llbracket \mathbf{Q} \rrbracket_{N^D}, \llbracket \mathbf{P} \rrbracket_{N^D}, \llbracket \mathbf{c} \rrbracket_{N^D})$$

$$state_{N^D} = seed_{N^D} || aux$$

add the leaf party's shares to the corresponding main party share:

for each main party index p in $\{(1, i_1), (2, i_2), \dots, (D, i_D)\}$ **do**

$$\llbracket \mathbf{x}_A \rrbracket_p += \llbracket \mathbf{x}_A \rrbracket_{i'}, \llbracket \mathbf{Q} \rrbracket_p += \llbracket \mathbf{Q} \rrbracket_{i'}, \text{ and } \llbracket \mathbf{P} \rrbracket_p += \llbracket \mathbf{P} \rrbracket_{i'}$$

$$\llbracket \mathbf{a} \rrbracket_p += \llbracket \mathbf{a} \rrbracket_{i'}, \llbracket \mathbf{b} \rrbracket_p += \llbracket \mathbf{b} \rrbracket_{i'}, \text{ and } \llbracket \mathbf{c} \rrbracket_p += \llbracket \mathbf{c} \rrbracket_{i'}$$
Leaf parties commit to their state: $\mathbf{com}_{i'}^{[e]} = \text{Hash}_0(salt, e, i', state_{i'}^{[e]})$

Commit to the full problem instance state:

for $e \in [\tau]$ **do** $\mathbf{com}^{[e]} = \text{Hash}_1(salt, e, \mathbf{com}_1^{[e]}, \dots, \mathbf{com}_{N^D}^{[e]})$ **Part 2:** Compute Challenge 1Compute $h_2 = \text{Hash}_2(m, salt, \mathbf{com}^{[1]}, \dots, \mathbf{com}^{[\tau]})$.Extend hash $\{\mathbf{r}^{[e]}, \mathbf{e}^{[e]}\}_{e \in [\tau]} \leftarrow \text{PRG}(h_2)$, where $(\mathbf{r}^{[e]}, \mathbf{e}^{[e]}) \in \mathbb{F}_{points}^t \times \mathbb{F}_{points}^t$.**Part 3:** On all main parties $(1, 1), \dots, (D, N)$, obtain communications:*Note:* we obtain a different split of the $\{\llbracket \boldsymbol{\alpha}^{[e]} \rrbracket, \llbracket \boldsymbol{\beta}^{[e]} \rrbracket, \llbracket \mathbf{v}^{[e]} \rrbracket\}$ for each D **for** $e \in [\tau]$ **do****for** $k \in [1, \dots, D]$ **do**

$$\text{Algorithm 2} \rightarrow \{\llbracket \boldsymbol{\alpha}^{[e]} \rrbracket, \llbracket \boldsymbol{\beta}^{[e]} \rrbracket, \llbracket \mathbf{v}^{[e]} \rrbracket\}_k$$

$$\rightarrow H_k^{[e]} = \text{Hash}_3(salt, e, \{\llbracket \boldsymbol{\alpha}^{[e]} \rrbracket, \llbracket \boldsymbol{\beta}^{[e]} \rrbracket, \llbracket \mathbf{v}^{[e]} \rrbracket\}_k)$$

Part 4: Compute Challenge 2Compute $h_4 = \text{Hash}_4(m, salt, h_2, \{H_1^{[e]}, \dots, H_D^{[e]}\}_{e \in [\tau]})$ Expand hash $\{i^{*[e]}\}_{e \in [\tau]} \leftarrow \text{PRG}(h_4)$, with $i^* \in [N^D]$, or equivalently $i^* = (i_1^*, \dots, i_D^*) \in [1, \dots, N]^D$ **Part 5:** Prover sends $salt|h_2|h_4|(\text{state}_{i' \neq i^*}^{[e]} | \mathbf{com}_{i^*}^{[e]} | \{\llbracket \boldsymbol{\alpha}^{[e]} \rrbracket_{i^*[e]}, \llbracket \boldsymbol{\beta}^{[e]} \rrbracket_{i^*[e]} \rrbracket\}_{e \in [\tau]})$ to verifier

Algorithm 7 Signature verification

Input: Public key $pk = (H, y)$, signature σ , and message $m \in \{0, 1\}^*$

1. Read signature into component parts

$$\sigma = \text{salt} | h_2 | h_4 | \left(\text{state}_{i' \neq i^*}^{[e]} | \mathbf{com}_{i^*}^{[e]} | \{ [\boldsymbol{\alpha}^{[e]}]_{i^*[e]}, [\boldsymbol{\beta}^{[e]}]_{i^*[e]} \} \right)_{e \in [\tau]}$$

2. Generate Challenge 1: extend hash $\{\mathbf{r}^{[e]}, \boldsymbol{\epsilon}^{[e]}\}_{e \in [\tau]} \leftarrow \text{PRG}(h_2)$

3. Evaluate MPC on main parties:

for $e \in [\tau]$ **do**

for $i' \neq i^*$ **do**

$$\mathbf{com}_{i'}^{[e]} = \text{Hash}_0(\text{salt}, e, i', \text{state}_{i'}^{[e]})$$

$$\mathbf{com}^{[e]} = \text{Hash}_1(\text{salt}, e, \mathbf{com}_1^{[e]}, \dots, \mathbf{com}_{N^D}^{[e]})$$

 Simulate MPC protocol on main parties

for Dimension $k \in [1, \dots, D]$ **do**

 Get $[\boldsymbol{\alpha}], [\boldsymbol{\beta}], [\mathbf{v}]$, and generate the $H_k^{[e]'}$:

for Revealed main parties $i_k \neq i_k^*$ **do**

 Aggregate shares to get $[\mathbf{x}_A], [\mathbf{Q}], [\mathbf{P}], [\mathbf{a}], [\mathbf{b}], [\mathbf{c}]$ for main parties

 Run Alg 3. to get $\{[\boldsymbol{\alpha}], [\boldsymbol{\beta}], [\mathbf{v}]\}_k$, and $H_k^{[e]'}$

Compute: $h'_2 = \text{Hash}_2(m, \text{salt}, \mathbf{com}^{[1]}, \dots, \mathbf{com}^{[N^D]})$

Compute: $h'_4 = \text{Hash}_4(m, \text{salt}, h'_2, \{H_1^{[e]'}, \dots, H_D^{[e]'}\}_{e \in [\tau]})$

Output: Accept: if $h'_2 = h_2$ and $h'_4 = h_4$, **else:** Reject.

4.4 Proof of security

Theorem 4 (Security). *Let the signature be (t, ϵ_{PRG}) -secure and with adversary of advantage at most ϵ_{SD} against the underlying syndrome decoding problem. Let $\text{Hash}_0, \text{Hash}_1, \text{Hash}_2, \text{Hash}_3, \text{Hash}_4$ be random oracles with output length 2λ bits. The probability of such an adversary producing an existential forgery under chosen message attack (EU-CMA) is bounded from above by:*

$$\begin{aligned} \Pr(\text{forge}) \leq & \frac{3 \cdot (q + \tau N^D q_S)^2}{2 \cdot 2^\lambda} + \frac{q_S(q_S + 5q)}{2^\lambda} \\ & + q_S \cdot \tau \cdot \epsilon_{PRG} + \left(q_2 \cdot p^{\tau'} + q_4 \cdot \left(\frac{1}{N^D} \right)^{\tau - \tau'} \right) + \epsilon_{SD}, \end{aligned}$$

where the adversary makes $q_0, q_1, q_2, q_3, q_4, q_S$ queries to the random oracles, and signature scheme respectively, where $q = \max\{q_0, q_1, q_2, q_3, q_4\}$, and where ϵ is the soundness given in Theorem 2.

Our security proof is largely inspired by the proof for the SDitH signatures in [FJR22] which in turn was largely inspired by the proof for Picnic [ZCD⁺20]. The proof essentially proceeds in two steps: First, we argue that we can efficiently simulate a signature oracle to the CMA adversary using only the public key and a HVZK-simulator for the ZKP. In the second half, we argue that if an adversary succeeds with a success probability that is not explained by “vanilla cheating”, then we must be able to extract a solution to the SD problem from the forgery.

The first step requires us to argue that we can simulate the signing oracle by reprogramming the random oracles to match transcripts generated by the HVZK simulator. For this we have to argue that with overwhelming probability, no (output) collisions occur for any of the random oracles used (Game 2), and that the values on which we want to reprogram the random oracle have not been queried before (Game 3). Afterwards, one would usually just do a game hop using HVZK. As we are following [FJR22] this is slightly more complicated

as after the ZKP, we introduced a salt for the signature scheme. This salt is also used for the pseudorandom generation of values of the ZKP when used as a subroutine of the signature scheme. Hence, we have to argue that this is ok (Games 4-7) following exactly the same reasoning as [FJR22].

The second half of the proof is traditionally done using a forking lemma argument. However, like [FJR22] and [ZCD⁺20], we exploit that the commitments are implemented using random oracles. Hence, we can simulate these for the adversary and inspect their internal database after a forgery is submitted. If the adversary used a valid SD solution, this allows us to extract the solution. It then remains to argue that an adversary that succeeds with a success probability that is greater than what vanilla cheating allows must have committed to a valid SD solution. Vanilla cheating here refers to the tactic that exactly achieves the success probability possible according to the soundness error of the parallel-composed protocol. At this point we correct a minor imprecision of the SDitH proof, which used $q_4 \varepsilon^\tau$ as the search bound for vanilla cheating the τ -times parallel-composition of SDitH which has soundness error ε . The correct bound is $q_2 \cdot p^{\tau'} + q_4 \cdot (\frac{1}{ND})^{\tau-\tau'}$, with $p^{\tau'} = \sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i} \geq p^{\tau'}$. This takes into account that one has two different ways to cheat per instance, controlled by the different challenges, computed as outputs of Hash₂ and Hash₄.

Proof of Security. In this proof of security, we detail the occurrences in which the true distribution of the signature scheme varies from the modelled distribution in the HVZK simulator, and we thus quantify each of the events in which the two may differ. Giving an upper bound on the cumulative probability of these events is required to demonstrate zero knowledge. In general, these events correspond to collisions in the input to the hash functions, which in the simulator gives uniformly random output, but for random oracles will give the same output deterministically, thus differing from the HVZK distribution. For input queries to be repeated, it is necessary (but not sufficient) that the *salt* must be the same, therefore we eventually replace the probability of looking for input collisions with the probability of finding repeated *salt*, which is strictly larger and easier to analyse. For each event, we consider a new game, and denote the probability of producing a forgery in game i as $\Pr_i(\text{forge})$. Finally we demonstrate that if a signer has generated signatures with sufficiently low computation, then they must indeed know a valid witness. From here we use the same rewinding argument from the proof of soundness to extract the witness.

Game 1: The adversary \mathcal{A} interacts with the true signature. \mathcal{A} is able to make queries to the signing oracle in order to get a valid message, signature pair (m, σ) . For \mathcal{A} to generate a forgery, they must produce a valid message, signature pair for a message which has not yet been queried to the signing oracle. We now seek to upper bound $\Pr_1(\text{forge})$.

Game 2: Now consider the event of collisions in the output of random oracles Hash₀, Hash₁, or Hash₃. This changes the probability, respectively, of successfully cheating at either the leaf party layer, at the commitment to the overall problem instance, or at the commitment to main party communications. In the case of a collision of the outputs of Hash₀, Hash₁, or Hash₃, we abort. The number of queries to Hash₀, Hash₁, or Hash₃ is bounded from above by $(q + \tau \cdot N^D \cdot q_S)$, where $q = \max\{q_0, q_1, q_2, q_3, q_4\}$. This is due to q_i queries directly to Hash _{i} , plus the queries which are present from the queries directly to the signing oracle. A single query to the signing oracle contains $\tau \cdot N^D$ queries to Hash₀ and τ queries to Hash₁. It also contains a single query to Hash₂, $\tau \cdot D$ queries to Hash₃, and a single query to Hash₄. Thus the probability of a hash collision in Hash₀, Hash₁, or Hash₃ is bounded from above by

$$|\Pr_1(\text{forge}) - \Pr_2(\text{forge})| \leq \frac{3 \cdot (q + \tau N^D q_S)^2}{2 \cdot 2^\lambda}. \quad (18)$$

Game 3: Now consider collisions on input queries. We abort if on any signing query, the inputs to any of Hash_i have been queried previously, either in a prior signing query, or in a query to any of $\text{Hash}_0, \text{Hash}_1, \text{Hash}_2, \text{Hash}_3, \text{Hash}_4$. For an input query to be duplicated, we must have *at least* that the sampled *salt* must be the same. Therefore we can bound this probability by

$$|\Pr_2(\text{forge}) - \Pr_3(\text{forge})| \leq \frac{q_S(q_S + q_0 + q_1 + q_2 + q_3 + q_4)}{2^\lambda} \leq \frac{q_S(q_S + 5q)}{2^\lambda}, \quad (19)$$

as in each signing query, the sampled *salt* could collide with that of any of the queries to any of the Hash_i , or with any of the other queries to the signing oracle.

Game 4: Now replace the hashes h_2, h_4 with uniform randomness (thus compute the challenges CH_1, CH_2 via expansion of the random h_2, h_4). Then the difference in forgery probability versus Game 3 is due to collisions in the input to hashes, which happens only when *salt*'s collide, and so in these scenarios Game 3 aborts. Thus the forgery probability is the same

$$\Pr_4(\text{forge}) = \Pr_3(\text{forge}) \quad (20)$$

Game 5: In this game we replace $\text{com}_{i^*}^{[e]}$ with uniform randomness. This only differs from Game 4 if $\text{state}_{i^*}^{[e]}$ was queried before. But $\text{com}_{i^*}^{[e]} = \text{Hash}_0(\text{salt}, e, i^*, \text{state}_{i^*}^{[e]})$ includes the tuple of indices (e, i) which is unique within a query, and so cannot happen in the same signing query. If this has happened in a previous signing query, or in another Hash_0 query, then the game is aborted due to Game 3 criteria (of a duplicated *salt*), thus the probability of aborting Game 5 is the same as for Game 4

$$\Pr_5(\text{forge}) = \Pr_4(\text{forge}). \quad (21)$$

Game 6: Next replace $\text{com}^{[e]}$ with uniform randomness. The only difference versus Game 4 is if in a previous signing query, Hash_1 has received the same input $(\text{salt}, e, \text{com}_1^{[e]}, \dots, \text{com}_{N^D}^{[e]})$. It is not possible to duplicate this input in the same signing query, as the index e is unique. However in the event that the same input query is registered in a previous signing query, or Hash_1 query, there is a duplicated *salt* and so the game aborts due to Game 3, thus the probability of a forgery remains the same

$$\Pr_6(\text{forge}) = \Pr_5(\text{forge}). \quad (22)$$

Game 7: In this experiment the signer uses the Simulator **v3** (in the proof of HVZK) to generate views of the parties, over which \mathcal{A} has an advantage of at most ϵ_{PRG} . In Simulator **v3**, the states of the parties no longer sum up to give the correct witness. The probability of forgery versus Game 6 is

$$|\Pr_7(\text{forge}) - \Pr_6(\text{forge})| \leq \tau \cdot q_S \cdot \epsilon_{\text{PRG}}. \quad (23)$$

Game 8: An execution e^* of a query to Hash_4

$$h_4 = \text{Hash}_4(m, \text{salt}, h_2, \{H_1^{[e]}, \dots, H_D^{[e]}\}_{e \in [\tau]}), \quad (24)$$

is said to define a correct witness if the following criteria are satisfied:

- Each of the $H_k^{[e]}$ are the output of a query to Hash_3

$$H_k^{[e]} = \text{Hash}_3(\text{salt}, e, \{[\alpha]^{[e]}, [\beta]^{[e]}, [\mathbf{v}]^{[e]}\}_k)$$

- h_2 is the output of a query to Hash₂

$$h_2 = \text{Hash}_2(m, \text{salt}, \mathbf{com}^{[1]}, \dots, \mathbf{com}^{[\tau]}).$$

- each $\mathbf{com}^{[e]}$ input to h_2 was generated by a prior query to Hash₁

$$\mathbf{com}^{[e]} = \text{Hash}_1(\text{salt}, e, \mathbf{com}_1^{[e]}, \dots, \mathbf{com}_{N^D}^{[e]}).$$

- each $\mathbf{com}_{i'}^{[e]}$ input to an above instance of $\mathbf{com}^{[e]}$ was generated by a prior query to Hash₀

$$\mathbf{com}_{i'}^{[e]} = \text{Hash}_0(\text{salt}, e, i', \text{state}_{i'}^{[e]}).$$

- the vector \mathbf{x} defined by the leaf party states $\{\text{state}_i\}_{i \in [N^D]}$, has small weight $wt(\mathbf{x}) \leq w$ and syndrome \mathbf{y} , i.e. $\mathbf{H}\mathbf{x} = \mathbf{y}$.

In the event (we call solve) that such an execution exists, where the message m has not already been queried to the signing oracle, it is possible to extract the correct witness \mathbf{x} from $\{\text{state}_i^{[e]}\}_{i \in [N^D]}$, which implies solving the underlying hard problem, so $\Pr_7(\text{solve}) \leq \epsilon_{\text{SD}}$.

We claim that finding a forgery without solving the underlying problem means that Game 8 gives

$$\Pr_8(\text{forge} \wedge \overline{\text{solve}}) \leq q_2 \cdot p'^{\tau'} + q_4 \cdot \left(\frac{1}{N^D}\right)^{\tau - \tau'}, \quad (25)$$

with $p'^{\tau'} = \sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i} \geq p^{\tau'}$, and τ' being the optimal number of false positives to find to minimize the cost of forgery as described in Eq 17. In this case, solve does not happen, so there is no successful execution e^* which gives a correct witness. Then to have obtained a forgery via query to Hash₄, the adversary must have either:

- found a false positive polynomial and challenge point combination

$$\mathbf{S} \cdot \mathbf{Q} \neq \mathbf{F} \cdot \mathbf{P}, \text{ but } \mathbf{S} \cdot \mathbf{Q}(r_k) = \mathbf{F} \cdot \mathbf{P}(r_k), k \in [t]$$

at challenge points r_k , which happens with probability p .

- else (with probability $1-p$) successfully cheat on a leaf party i' which will be successfully be challenged with probability $1/N^D$,
- (also with probability $1-p$) or equivalently, cheat on one out of N main parties, independently in each of the D protocols, with probability $(1/N)^D$, which is equivalent to cheating on one of the N^D leaf parties.

and this has to have happened independently for each of the τ iterations of the protocol. \square

5 Performance and analysis

In this section we will analyse the protocol with respect to the communication cost. We first provide costs for the zero-knowledge protocol, in order to compare with the other protocols using syndrome decoding, and then provide parameters and costs for the signature scheme. In the original SDitH work, the authors present a variant of the underlying SD problem known as the d -split problem, and explain how their signature scheme can be adapted to be based on this variant of the SD problem. We do not present the same adaptations to this problem for our signature scheme. However, the difference presented by the d -split problem affects

only the underlying hardness assumptions, and so it is still instructive to present parameter sets for the d -split variants for comparison with the previous signature schemes.

There are a few points in the protocol which we do not include as their impact is arbitrarily small compared to the main communication cost, these being the challenges from the verifier. The communication cost is then calculated from the following:

- **Com**: the hash, h , of the N^D commitments.
- **Res₁**: the hash, h' , of the D hashes output from the MPC simulation.
- **Res₂**: the $(\text{state}_{i_1, \dots, i_D}, \rho_{i_1, \dots, i_D}) \forall (i_1, \dots, i_D) \neq (i_1^*, \dots, i_D^*)$, $\mathbf{com}_{(i_1^*, \dots, i_D^*)}$, $\llbracket \boldsymbol{\alpha} \rrbracket_{(i_1^*, \dots, i_D^*)}$, $\llbracket \boldsymbol{\beta} \rrbracket_{(i_1^*, \dots, i_D^*)}$.

If we consider each leaf ($i' = (i_1, \dots, i_D) \in \{1, \dots, N^D\}$) of the hypercube, for all but the final leaf ($i' \neq N^D$) the cost of each $\text{state}_{i'}$ is the size of a seed of λ bits. For the case of the final leaf ($i' = N^D$), the $\text{state}_{i'}$ consists of seed_{N^D} , as well as the auxiliary which consists of (i) the share $\llbracket \mathbf{x}_A \rrbracket_{N^D}$ of the plaintext, (ii) the shares $\llbracket \mathbf{Q} \rrbracket_{N^D}$ and $\llbracket \mathbf{P} \rrbracket_{N^D}$ being two polynomials of degree $w - 1$, and (iii) the shares $\llbracket \mathbf{c} \rrbracket_{N^D}$ of the t points of $\mathbb{F}_{\text{points}}$.

The only parts within the commitment and responses that are affected by the hypercube component, D , is the number of, and thus size of, the seed and commitment randomness. This in essence becomes a sibling path, of length D , from $(\text{state}_{i_1^*, \dots, i_D^*}, \rho_{i_1^*, \dots, i_D^*})$ to the tree root, which will cost at most $D \cdot \lambda \cdot \log_2(N)$ bits. For the remaining costs, we have the commitment $\mathbf{com}_{(i_1^*, \dots, i_D^*)}$ of 2λ bits and $\llbracket \boldsymbol{\alpha} \rrbracket_{(i_1^*, \dots, i_D^*)}$, $\llbracket \boldsymbol{\beta} \rrbracket_{(i_1^*, \dots, i_D^*)}$ are elements of $\mathbb{F}_{\text{points}}$. We then calculate the size of the communication cost (in bits) of a single round of the protocol as:

Total Size = 4λ	size of h and h' .
$+ k \cdot \log_2(\mathbb{F}_{\text{SD}})$	size of $\llbracket \mathbf{x}_A \rrbracket_{N^D}$.
$+ 2w \cdot \log_2(\mathbb{F}_{\text{poly}})$	sizes of $\llbracket \mathbf{Q} \rrbracket_{N^D}$ and $\llbracket \mathbf{P} \rrbracket_{N^D}$.
$+ (2 \cdot d + 1) \cdot t \cdot \log_2(\mathbb{F}_{\text{points}})$	sizes of $\llbracket \boldsymbol{\alpha} \rrbracket_{(i_1^*, \dots, i_D^*)}$, $\llbracket \boldsymbol{\beta} \rrbracket_{(i_1^*, \dots, i_D^*)}$, $\llbracket \mathbf{c} \rrbracket_{N^D}$.
$+ D \cdot \lambda \cdot \log_2(N)$	size of the seeds.
$+ 2\lambda$	size of $\mathbf{com}_{(i_1^*, \dots, i_D^*)}$.

In order to achieve the target security level and soundness, $2^{-\lambda}$, we can perform τ parallel repetitions of the protocol. Using the definition of the forgery cost in Equation 17 and using predefined values for false positivity, we can find the minimum number of repetitions, τ , that satisfies Equation 17. Additionally, we do not need to repeat this process for the entire communication costs, the values for h and h' can be merged for each τ . Thus, the total communication cost (in bits) of the overall protocol with τ repetitions is:

$$\begin{aligned} \text{Size} = & 4\lambda + \tau \cdot (k \cdot \log_2(|\mathbb{F}_{\text{SD}}|) + 2w \cdot \log_2(|\mathbb{F}_{\text{poly}}|) \\ & + (2d + 1) \cdot t \cdot \log_2(|\mathbb{F}_{\text{points}}|) + D \cdot \lambda \cdot \log_2(N) + 2\lambda). \end{aligned}$$

Using Equation 10 we have the obtained soundness error as $(p + (1 - p) \frac{1}{N^D})^\tau$.

5.1 Comparison of code-based zero-knowledge protocols

The SDitH protocol is not the first proposal for a zero-knowledge protocol using syndrome decoding. There have been other proposals for identity schemes and signature schemes, we

Table 1: Communication sizes of ZK protocols using syndrome decoding.

Protocol	Year	Instance 1	Instance 2	Proved Statement
Stern [Ste94]	1993	37.4 kB	46.1 kB	$y = Hx, \text{wt}(x) = w$
Véron [Vér97]	1997	31.7 kB	38.7 kB	<i>message decoding</i>
CVE11 [CVE11]	2010	-	37.4 kB	$y = Hx, \text{wt}(x) = w$
AGS11 [AGS11]	2011	24.8 kB	-	$y = Hx, \text{wt}(x) = w$
GPS22 [GPS22] (short)	2021	-	15.2 kB	$y = Hx, \text{wt}(x) = w$
GPS22 [GPS22] (fast)	2021	-	19.9 kB	$y = Hx, \text{wt}(x) = w$
FJR21 [FJR21] (short)	2021	12.9 kB	15.6 kB	$y = Hx, \text{wt}(x) = w$
FJR21 [FJR21] (fast)	2021	20.0 kB	24.7 kB	$y = Hx, \text{wt}(x) = w$
SDitH [FJR22] (short)	2022	9.7 kB	6.9 kB	$y = Hx, \text{wt}(x) \leq w$
SDitH [FJR22] (fast)	2022	14.4 kB	9.7 kB	$y = Hx, \text{wt}(x) \leq w$
Ours (shortest)	2022	6.0 kB	4.5 kB	$y = Hx, \text{wt}(x) \leq w$
Ours (shorter)	2022	7.5 kB	5.5 kB	$y = Hx, \text{wt}(x) \leq w$
Ours (short)	2022	9.7 kB	6.9 kB	$y = Hx, \text{wt}(x) \leq w$
Ours (fast)	2022	14.4 kB	9.7 kB	$y = Hx, \text{wt}(x) \leq w$

can compare these protocols on different instances of syndrome decoding for 128-bit security. Table 1 shows this comparison which is also given in [FJR22], which also provides further calculation costs and parameters. Each scheme in Table 1 utilizes the same parameters (m, k, w) ; either Instance 1 [FJR21] which is SD on \mathbb{F}_2 for $(1280, 640, 132)$ or Instance 2 [CVE11] which is SD on \mathbb{F}_{2^8} for $(208, 104, 78)$, for the given communication costs.

In order to directly compare with [FJR22], we utilize the same parameters for $(N, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t)$, which only differ in (N, τ) , in which our protocol optimizes. Our protocol also differs slightly in the calculation of the soundness error, ε , which affects the security level being attained; with SDitH using $(p + \frac{1}{N} - p \cdot \frac{1}{N})$ whereas we use $(p + \frac{1}{ND} - p \cdot \frac{1}{ND})$.

SDitH ZKP parameters:

Instance 1:

Short: $(256, 16, 2^{11}, 2^{22}, 2)$; $\varepsilon^\tau = 2^{-128}$

Fast: $(32, 26, 2^{11}, 2^{22}, 1)$; $\varepsilon^\tau = 2^{-129.6}$

Instance 2:

Short: $(256, 16, 2^8, 2^{24}, 2)$; $\varepsilon^\tau = 2^{-128}$

Fast: $(32, 26, 2^8, 2^{24}, 1)$; $\varepsilon^\tau = 2^{-130.0}$

Our ZKP parameters:

Instance 1:

Shorter: $(2^{12}, 11, 2^{11}, 2^{22}, 2)$; $\varepsilon^\tau = 2^{-132}$

Shortest: $(2^{16}, 8, 2^{11}, 2^{22}, 2)$; $\varepsilon^\tau = 2^{-128}$

Instance 2:

Shorter: $(2^{12}, 11, 2^8, 2^{24}, 2)$; $\varepsilon^\tau = 2^{-132}$

Shortest: $(2^{16}, 8, 2^8, 2^{24}, 2)$; $\varepsilon^\tau = 2^{-128}$

For Instance 1 and Instance 2, and using a target soundness of 2^{-128} , Table 1 provides the corresponding communication costs for the different zero-knowledge protocols using syndrome decoding. We reuse the parameters used in SDitH for the Short and Fast variants, thus we achieve similar costs for these. We also extended these parameters for a large number of simulated parties to achieve Shorter and Shortest variants. Details on the communication costs for the other protocols can be found in the full version of [FJR22, Appendix B]. Also, it is worth noting that there are some differences between the proved statements; i.e. either proving the equality or inequality for the Hamming weight of w .

Table 2: The SD and MPC parameters used in our protocol, originally from [FJR22].

Scheme	SD Parameters					MPC Parameters			
	q	m	k	w	d	$ \mathbb{F}_{\text{poly}} $	$ \mathbb{F}_{\text{points}} $	t	p
Variant 1	2	1280	640	132	1	2^{11}	2^{22}	6	$\approx 2^{-69}$
Variant 2	2	1536	888	120	6	2^8	2^{24}	5	$\approx 2^{-79}$
Variant 3	2^8	256	128	80	1	2^8	2^{24}	5	$\approx 2^{-78}$

5.2 Parameter Selection

Here we derive the parameters we use for our proposed signature scheme. Due to similarities with SDitH we utilize the same values for many of the parameters; this also makes it simpler to compare the two protocols in terms of efficiency and communication costs. As with SDitH, the parameters chosen are for attaining at least 128 bits of security.

5.2.1 Syndrome Decoding and MPC Parameters

To estimate the security levels of cryptographic schemes based on the hardness of solving a syndrome decoding instance for a random linear code over \mathbb{F}_2 , we use algorithms which perform the best practical attacks. Currently this is a version of the Information-Set Decoding (ISD) algorithm [MMT11], based on previous work by Finiasz and Sendrier [FS09]. Recently an argument was made that the lower bound cost of the attack can be calculated by considering the cost of its topmost recursion step [FJR21].

The details of the algorithm will be omitted since the SD parameters will be reused from SDitH, however we provide a description of each parameter set (or variant) and their differences below. Each variant listed will have associated parameters for (q, m, k, w, d) which define its hardness.

- Variant 1: based on the standard binary syndrome decoding problem with some parameters used from [FJR21].
- Variant 2: based on the d -split binary syndrome decoding problem, where d is chosen such that $m/d \leq 2^8$, meaning that $\mathbb{F}_{\text{poly}} = \mathbb{F}_{2^8}$.
- Variant 3: based on the syndrome decoding problem defined over \mathbb{F}_{2^8} with some parameters used from [CVE11].

The choice of the MPC parameters will also follow from the suggestions in SDitH. The MPC parameters are chosen such that the resulting communication cost is small, thus the smallest possible field for \mathbb{F}_{poly} is used as the communication includes polynomials in this field. The SD and MPC parameters for the three variants are provided in Table 2.

5.2.2 Signature Scheme Parameters

Now with SD and MPC parameters we can propose parameters for our signature scheme and provide costs. The parameters of the signature scheme that primarily contribute to the communication cost are $(N, D, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t)$. Again, we fix many of these parameters for comparison reasons, these being the SD and MPC parameters shown in Table 2.

Table 3 shows the parameters proposed for SDitH. The parameters are derived using the three different variations, as well as having two different values for the party size, N , with the aim of producing a fast computation version, for $N = 32$, and a short communication cost version, for $N = 256$. Once the party size is defined, the number of repetitions, τ can

Table 3: SDitH [FJR22] parameters (N, τ) with key and signature sizes in bytes.

λ	Scheme	Aim	Parameters		Sizes		
			N	τ	pk	sk	Sign
128	Variant 1	Fast	32	27	96	16	16 422
128		Short	256	17	96	16	11 193
128	Variant 2	Fast	32	27	97	16	17 866
128		Short	256	17	97	16	12 102
128	Variant 3	Fast	32	27	144	16	12 115
128		Short	256	17	144	16	8 481

thus be calculated such that they gain the target security level, which in this work is at least 128 bits of security.

The parameters in which our protocol primarily optimizes over SDitH are the party size, being N or in our case N^D , and the resulting repetitions required, τ . A large part of the signature scheme in SDitH is the auxiliary, being made up of $(\llbracket \mathbf{x}_A \rrbracket_N, \llbracket \mathbf{Q} \rrbracket_N, \llbracket \mathbf{P} \rrbracket_N, \llbracket \mathbf{c} \rrbracket_N)$, which is then repeated for each τ . Being able to significantly reduce τ means we drastically reduce this cost. In Figure 3, we show the relationship between τ and D and how this affects the size of the signature.

In our parameter selection, we decided to fix the value for $N = 2$ and adapt for different dimension sizes, D . It is possible for parameters to become equivalent, for example $(N = 2^{16}, \tau = 9)$ produces the same communication costs and computations as $(N = 256^2, \tau = 9)$, however the former parameters require significantly less (potentially expensive) MPC computations and in turn require (probably less expensive) hash calculations. This quality in the flexibility we gain with parameters is particularly coveted when its applications on a variety of hardware is considered; which can range from CPUs with dedicated instructions for field arithmetic, to mid-range devices with AES-NI and SHA extension support, to low-end constrained devices with limited ISA support for cryptographic operations.

A complete summary of the parameters of our protocol are given in Table 4. Similarly to SDitH we provide parameters for the three SD and MPC variants, and those parameters with the aim of having short communication costs (for $N = 2^{16}$ and $N = 2^{12}$) and fast computations ($N = 2^8$ and $N = 2^5$). The associated public-key and secret-key values are unchanged compared to SDitH parameters, the major differences are seen in the signature sizes and computation costs. We use similar nomenclature to SDitH, but due to the savings we make in performance, we can ‘upgrade’ their previous parameters from Fast and Short, to Faster and Fast, respectively. The latter parameters we propose increase the dimension size, thus the party size in the MPC protocol, which finally results in Short and Shorter parameters.

5.3 Implementation

In the previous sections we provided parameter sets which, as well as providing key and signature sizes, can give estimates on runtime performance; based on the number of computations required and then the expected performances of randomness generation and hash functions. However, it is still important to realize exact performance figures and thus we describe this information here and compare this to the current state-of-the-art.

We focus on the implementation of the Variant 3 parameter sets, since these are the most interesting as they provide the fastest and smallest signatures. Moreover, our optimizations

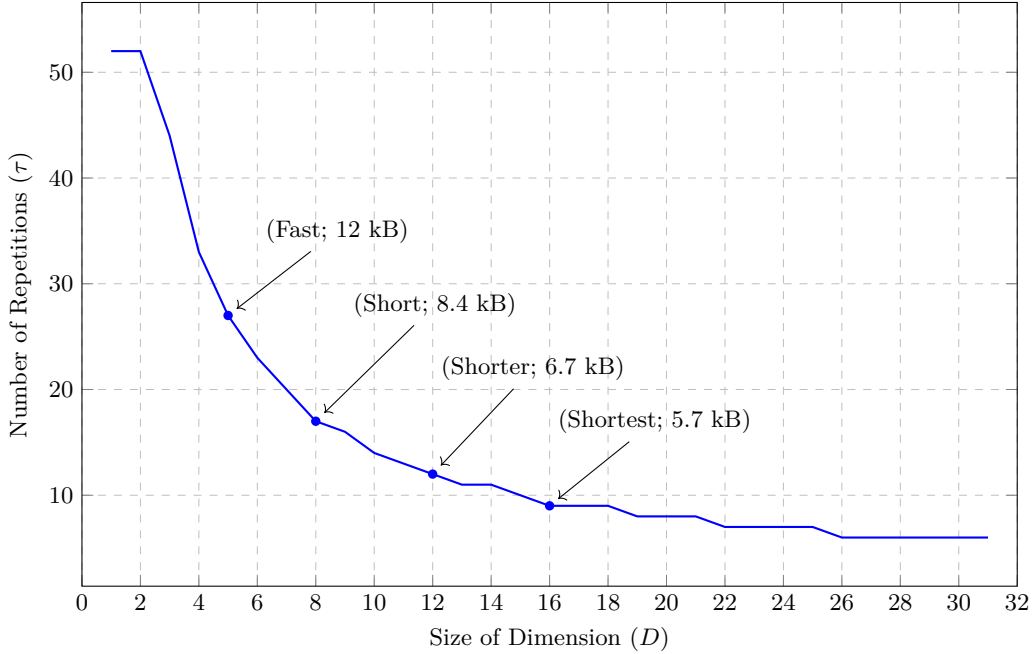


Fig. 3: The relationship between the size of the dimension, D , and the number of repetitions within the signature scheme, τ , using $N = 2$. Parameters and signature sizes provided for Variant 3.

do not have an effect on key generation since the secret and public keys are identical, both seeded and expanded. We also expect verification to have the same performance as signing, thus we omit these from the results. We provide the benchmark results of our signature runtime as well as SDitH in Table 5. In order to simplify and fairly benchmark on the same processor the authors of SDitH kindly shared their code used for the experiments in this section upon request. For comparison, we also ran the SDitH implementation for the Shorter parameter set, however the Shortest parameters gave issues and have thus been omitted from the table.

In both implementations, the offline phase uses the AES native instructions for seed expansion and SHAKE for hash and commitments purposes. Both implementations also rely on a fast gf256 library³, which utilizes AVX2 instructions. Our processor does not support the newer Galois Field New Instructions (GFNI) opcodes. For the same number of leaf shares, N for SDitH and N^D for our protocol, the performance of both signature schemes in the offline phase are more or less the same as the one in their implementation, which confirms our expectations, and highlights that both software implementations are equivalent in performance, the performance differences observed come from the protocol differences. Our online phase however is largely accelerated compared to the reference implementation, which confirms the expected $N^D \rightarrow N \cdot D$ algorithmic speedup. Again, we can verify that the gain is roughly $N \cdot D / N^D$ as we would expect from comparable implementations. In fact, our online costs are more-or-less constant for a given security level as they are in $N \cdot D \cdot \tau$ and the security is roughly in $\log_2 N \cdot D \cdot \tau$ (and N is constant). Besides being roughly constant, they are also very small, around 1 ms, and can probably be further optimized.

³ <https://github.com/catid/gf256>.

Table 4: Our parameters with related key and signature sizes in bytes.

λ	Scheme	Aims	Parameters			Sizes		
			N	D	τ	pk	sk	sign
128	Variant 1	Fast	2	5	27	96	16	16 386
128		Short	2	8	17	96	16	11 157
128		Shorter	2	12	12	96	16	8 662
128		Shortest	2	16	9	96	16	7 089
128	Variant 2	Fast	2	5	27	97	16	17 830
128		Short	2	8	17	97	16	12 066
128		Shorter	2	12	12	97	16	9 304
128		Shortest	2	16	9	97	16	7 570
128	Variant 3	Fast	2	5	27	144	16	12 079
128		Short	2	8	17	144	16	8 445
128		Shorter	2	12	12	144	16	6 748
128		Shortest	2	16	9	144	16	5 653

Table 5: Benchmarks of reference implementations of our scheme and SDitH [FJR22]. Both schemes were run using a single CPU core of a 3.1 GHz Intel Core i9-9990K.

λ	Scheme	Aim	Parameters			Sign Time (ms)		
			N	D	τ	Offline	Online	Total
128	SDitH [FJR22] (Variant 3)	Fast	32	-	27	0.87	5.03	5.96
128		Short	256	-	17	4.33	18.95	23.56
128		Shorter	2^{12}	-	12	59.24	251.14	313.70
128		Shortest	2^{16}	-	9	-	-	-
128	Ours (Variant 3)	Fast	2	5	27	1.14	1.62	3.78
128		Short	2	8	17	5.06	1.23	7.17
128		Shorter	2	12	12	59.68	1.22	60.63
128		Shortest	2	16	9	734.82	1.10	736.85

References

- [AAB⁺02] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: minimizing assumptions for security and forward-security. In L. R. Knudsen, editor, *EUROCRYPT 2002*, pages 418–433. Springer, Heidelberg, 2002 (cited on page 21).
- [AGS11] C. Aguilar, P. Gaborit, and J. Schrek. A new zero-knowledge code based identification scheme with reduced communication. In *2011 IEEE Information Theory Workshop*, pages 648–652. IEEE, 2011 (cited on page 28).
- [Bea92] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *CRYPTO’91*, pages 420–432. Springer, Heidelberg, 1992 (cited on page 6).
- [CVE11] P.-L. Cayrel, P. Véron, and S. M. El Yousfi Alaoui. A zero-knowledge identification scheme based on the q-ary syndrome decoding problem. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *SAC 2010*, pages 171–186. Springer, Heidelberg, 2011 (cited on pages 28, 29).

- [DGV⁺16] Ö. Dagdelen, D. Galindo, P. Véron, S. M. El Yousfi Alaoui, and P.-L. Cayrel. Extended security arguments for signature schemes. *Designs, Codes and Cryptography*, (2):441–461, 2016 (cited on page 3).
- [FJR21] T. Feneuil, A. Joux, and M. Rivain. Shared permutation for syndrome decoding: new zero-knowledge protocol and code-based signature. Cryptology ePrint Archive, Report 2021/1576, 2021. <https://eprint.iacr.org/2021/1576> (cited on pages 28, 29).
- [FJR22] T. Feneuil, A. Joux, and M. Rivain. Syndrome decoding in the head: shorter signatures from zero-knowledge proofs. In Y. Dodis and T. Shrimpton, editors, *CRYPTO 2022, Part II*, pages 541–572. Springer, Heidelberg, 2022. Full version at <https://eprint.iacr.org/2022/188> (cited on pages 2, 7, 10, 12, 14–16, 20, 23, 24, 28–30, 32).
- [FS09] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *ASIACRYPT 2009*, pages 88–105. Springer, Heidelberg, 2009 (cited on page 29).
- [FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO’86*, pages 186–194. Springer, Heidelberg, 1987 (cited on page 21).
- [GGM84] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, 1984 (cited on page 3).
- [GPS22] S. Gueron, E. Persichetti, and P. Santini. Designing a practical code-based signature scheme from zero-knowledge proofs with trusted setup. *Cryptography*, (1):5, 2022 (cited on page 28).
- [IKO⁺07] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In D. S. Johnson and U. Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, 2007 (cited on pages 1, 6).
- [KZ20] D. Kales and G. Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In S. Krenn, H. Shulman, and S. Vaudenay, editors, *CANS 20*, pages 3–22. Springer, Heidelberg, 2020 (cited on page 21).
- [MMT11] A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, pages 107–124. Springer, Heidelberg, 2011 (cited on page 29).
- [PS00] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, (3):361–396, 2000 (cited on page 3).
- [Ste94] J. Stern. Designing identification schemes with keys of short size. In Y. Desmedt, editor, *CRYPTO’94*, pages 164–173. Springer, Heidelberg, 1994 (cited on pages 2, 28).
- [Vér97] P. Véron. Improved identification schemes based on error-correcting codes. *Applicable Algebra in Engineering, Communication and Computing*, (1):57–69, 1997 (cited on page 28).
- [ZCD⁺20] G. Zaverucha, M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, J. Katz, X. Wang, V. Kolesnikov, and D. Kales. Picnic. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions> (cited on pages 2, 23, 24).