HYPERSHIELD: Protecting the Hypercube MPC-in-the-Head Framework Against Differential Probing Adversaries without Masking

Linda Pirker [1,⋆], Quinten Norga [2], Suparna Kundu [2], Anindya Ganguly [3], Barry van Leeuwen [2], Angshuman Karmakar [3], and Ingrid Verbauwhede [2]

[1] Graz University of Technology, Graz, Austria
linda.pirker@student.tugraz.at
[2] COSIC, KU Leuven, Leuven, Belgium
quinten.norga,suparna.kundu,barry.vanleeuwen,ingrid.verbauwhede@esat.kuleuven.be
[3] Indian Institute of Technology Kanpur, Kanpur, India
anindyag,angshuman@cse.iitk.ac.in

Abstract. Post-quantum secure digital signatures based on the MPC-in-the-Head (MPCitH) paradigm, a zero-knowledge (ZK) proof-based construction, are becoming increasingly popular due to their small public key size. However, the development of techniques for protecting MPCitH-based schemes against side-channel attacks remains slow, despite them being critical for real-world deployment.

In this work, we adapt the Hypercube-MPCitH framework and exploit its native use of additive secret sharing to enable inherent protection against first- and high-order differential power analysis (DPA). We first perform a sensitivity analysis of the Hypercube Syndrome Decoding in the Head (SDitH) digital signature scheme with respect to both simple and differential power analysis. Based on this insight, we propose a *tweak* to the signature scheme to increase its *inherent* resistance against DPAs *by design*, eliminating the need to explicitly mask large parts of the signing procedure. More specifically, this is achieved through the novel $(k+1)$-Hypercube ZK Protocol: the proposed tweak increases the number of hidden shares an adversary must probe to recover the secret key from one to $k+1$, thus achieving *inherent* masking order k. Typically, increasing the amount of hidden shares results in a degradation of soundness in the zero-knowledge proof and as a result increases the signature size to a point where the scheme becomes of limited practical interest. To address this, we propose a technique to select the *hidden* shares in a more structured and optimal fashion, by exploiting the GGM tree structure in the Hypercube-MPCitH framework. As a result, the amount of revealed seeds is reduced, resulting in a smaller signature size.

Finally, we estimate the performance of the proposed Hypercube-SDitH signature scheme and compare it against the cost of traditional masking. For 1st-order protection, our tweaked signature scheme only incurs

⋆ Part of this work was completed while the author was at COSIC, KU Leuven.

a $15 - 50\%$ overhead in computational cost, compared to an estimated overhead of 100% for a fully masked implementation, while the overhead in signature size is around $60 - 95\%$. Overall, we demonstrate that the proposed $(k + 1)$-Hypercube ZK Protocol can be used to construct efficient, DPA-resistant MPCitH-based digital signatures.

Keywords: Side-Channel Attacks · MPC-in-the-Head · Post-Quantum Cryptography · Digital Signatures.

1 Introduction

The constant progress in the development of large-scale quantum computers poses a serious threat to the security of widely used public-key cryptosystems, such as RSA [41] and elliptic curve systems [30,34]. Acknowledging this urgency, NIST launched a competition to standardize quantum-resistant public-key cryptography, including digital signature schemes [38,2]. The first standardization effort concluded with the selection (and standardization) of multiple signature schemes: ML-DSA (f.k.a. CRYSTALS-Dilithium) [16,36], FN-DSA (f.k.a. Falcon) [21] and SLH-DSA (f.k.a. SPHINCS+) [11,37]. To increase diversity in standardized schemes, NIST launched an additional call for digital signatures schemes, explicitly focused on standardizing schemes based on different mathematical problems [35].

Signature schemes based on the MPC-in-the-Head (MPCitH) framework are a promising class of candidates in this ongoing competition [3], with SDitH [17], MQOM [10], and FAEST [6] advancing to Round 3. In MPCitH-based digital signature schemes, a zero-knowledge proof is constructed using secret sharing techniques from secure multi-party computation (MPC) [26]. More specifically, the prover uses a zero-knowledge proof to prove knowledge of a secret key to the verifier. This proof is split into multiple partial proofs on (a typically large amount of) shares of a (secret) key-derived witness and the execution of those partial proofs is emulated locally ("*in-the-Head*") by the signer. Eventually, a number of shares lower than the privacy threshold of the used secret sharing scheme (e.g. additive sharing) is revealed in the signature, allowing the verifier to recompute and verify parts of the proof without being able to recover the secret key. After the initial proposal of the MPCitH framework, several optimized versions like Threshold-Computation-in-the-Head [19], Hypercube-MPCitH [1] or VOLE-in-the-Head [6] have been developed, which reduce the amount of party computations that need to be performed or the signature size. Hypercube-MPCitH, used by SDitH [33], is particularly interesting due to the small signature sizes and its simple and scalable structure, utilizing the generally cheap and easy-to-implement additive secret sharing.

Power analysis attacks, most famously Differential Power Analysis (DPA) [31], are a type of passive Side-Channel Attack (SCA) which poses a serious threat to cryptographic implementations, especially on embedded devices [32,29,9]. These

power-based SCAs enable attackers to recover secret information, by measuring and exploiting statistical dependencies of the processed data with the device's power consumption. Masking is a well-studied countermeasure to construct d-probing secure implementations, resistant against d-order DPAs [27]. This countermeasure transforms the cryptographic functions to operate on $d+1$ randomized shares of sensitive data and thereby breaks dependencies between power consumption and secret data.

However, the application of masking to MPCitH-based digital signature schemes is expensive due to the nature of its zero-knowledge proof framework [42,20]. More specifically, to achieve d-order protection, each of the n party computations of a share must be masked, which essentially amounts to further creating d shares of each party's share of the secret, thus $n \cdot (d+1)$ in total [4]. As the cost of masking Hypercube-MPCitH schemes is implicitly high, the question arises *whether the underlying Hypercube-MPCitH framework and its secret sharing can be tweaked to efficiently achieve inherent side-channel protection?*

Related Works Side-channel protection of MPCitH signature schemes, especially recent NIST candidates, is a almost unexplored topic. Picnic [15], which was submitted to the first NIST standardization competition, uses secret sharing as an integral part of its structure. Yet it has been shown to be vulnerable to DPA [22]. In [42], the authors show that certain modifications to the scheme allow to utilize some inherent protection provided by the secret sharing at a loss of soundness of the zero-knowledge proof, which can be accounted for by adjusting the protocol parameters. However, the proposed strategy is not directly applicable to Hypercube-MPCitH signature schemes, due to different structural properties of Picnic's specific MPC-in-the-head instantiation. More recent work by Feneuil et al. [20] is directed towards the masking-friendliness of the related Threshold-Computation-in-the-Head (TCitH) framework [19], reducing the number of calls to masked symmetric primitives. Additionally, the authors show that the tweak initially proposed for Picnic can also be applied to the TCitH framework, however leading to a large overhead in signature which limits practical applicability.

Our Contributions Previous work on tweaking other variants of MPCitH signature scheme to achieve implicit side-channel protection has yielded mixed results, mainly due to relatively large overheads in signature size and computational cost. In this work, we present a tweaked Hypercube-MPCitH framework that achieves inherent side-channel resistance without relying on masking, demonstrating that this protection can be integrated into Hypercube-MPCitH with moderate additional cost. More specifically:

1. We provide a detailed analysis of side-channel attack paths in the SDitH signature scheme, which is constructed using the Hypercube-MPCitH framework. We identify operations and subroutines that are vulnerable to SPAs and DPAs. (Section 3)
2. We propose the novel $(k+1)$-Hypercube ZK Protocol: a tweaked hypercube-based zero-knowledge protocol that achieves inherent high-order side-channel

security against DPAs, *by design*. This is achieved by introducing slack in the open parties, thus revealing fewer shares than the reconstruction threshold, increasing the number of shares an adversary needs to probe to recover the secret. Further, we propose an optimized approach for selecting hidden shares in GGM trees, further reducing the signature size. We also provide a proof of completeness, soundness and the zero-knowledge property of our tweaked protocol. (Section 4)

3. We demonstrate how the proposed tweak can be used to construct a *hardened Hypercube-SDitH signature scheme* that achieves inherent k-order probing security for a significant part of the signing procedure at a lower overhead than masking. We carefully analyze the impact of all parameter choices on bandwidth and performance cost, and propose a set of optimal (short & fast) parameter sets for our implementation in C.[4] Our results show that probing security can be achieved at practical performance: for a slack of $k = 3$, the fast parameter set reaches 165.2 ms signing time (vs. 28.26 ms unprotected), while the small parameter set yields signatures of 23592 kB (vs. 6784 kB). Finally, we analyze its overhead compared to a fully masked implementation and the recently proposed masking-friendly TCitH scheme, exploring the range of trade-offs between speed and signature size. (Section 5)

2 Preliminaries

We denote by $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_n)$ the sharing of a value x. The k-th share is denoted $[\![x]\!]_k$. To prevent confusion between the usual notation of N as the number of parties or shares in literature and the hypercube parameter N for the side of the hypercube, we always refer to the number of parties with n.

2.1 Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs) are interactive protocols in which a prover $\mathcal{P}$ proves a statement to a verifier $\mathcal{V}$, typically the knowledge of a witness for some relation. The messages exchanged during the protocol are called transcripts of the execution. A ZKP must fulfill the following properties: (i) *Completeness*, an honest $\mathcal{P}$ will always succeed in convincing $\mathcal{V}$; (ii) *Soundness*, no cheating $\tilde{\mathcal{P}}$ can succeed in convincing $\mathcal{V}$, except with a small probability ε, also known as the soundness error; and (iii) *Zero-Knowledge*, the transcript of an execution does not reveal any information about the witness.

2.2 (Additive) Secret Sharing

Secret sharing is a technique from multi-party computation (MPC) to distribute a secret (split into shares) between multiple parties, ensuring only a *qualified*

[4] Our implementation will be made publicly available after publication.

subset can reconstruct the secret. Additive sharings are n-out-of-n secret sharings, which means that all n shares have to be known to reconstruct the secret. An additive secret sharing $[\![x]\!]_{k\in[1,n]}$ of the variable x between n parties is defined such that $x = \sum_{k=1}^{n} [\![x]\!]_k$. In practice, they are usually constructed by choosing the first $n-1$ shares at random and choosing the last share, also known as *auxiliary share*, such that $[\![x]\!]_n = x - \sum_{k=1}^{n-1} [\![x]\!]_k$.

2.3 MPC-in-the-Head Framework

The MPCitH paradigm is based on the work of Ishai et al. [26], who first described a general construction of a zero-knowledge proof using a MPC protocol combined with a commitment protocol. The framework allows a prover to prove the knowledge of a secret x to a verifier by executing an MPC protocol among n parties. The parties jointly evaluate a proof Π that either outputs ACCEPT for a valid secret x or REJECT for an invalid secret. In more detail, the following steps are performed:

- The prover commits to n shares $[\![x]\!]$ of the secret x. She simulates MPC locally ("*in-the-Head*") to compute proofs and commits to the views of the parties. Further, she sends the n output shares $[\![\Pi(x)]\!]$ of the parties to the verifier.
- The verifier chooses a subset of $t \leq n-1$ parties and asks the prover to reveal the view of the parties. The zero-knowledge property is preserved, as up to $n-1$ views do not reveal any information about the secret x.
- After receiving the views of the revealed parties, the verifier checks the consistency of the commitments by recomputing the t party computations. Additionally it verifies that the recombined output shares $[\![\Pi(x)]\!]$ are evaluated to ACCEPT.

A verifier without knowledge of the secret x can either generate a signature that evaluates to $\Pi(x) = \text{ACCEPT}$ with a false positive probability of p or cheat by corrupting one party computation, which will be undetected with a probability of $\frac{n-t}{n}$. This results in a total soundness error of $p + (1-p)\frac{n-t}{n}$ of the zero-knowledge protocol. To increase the soundness of the proof, τ parallel iterations of the MPC protocol are performed.

2.4 Hypercube-MPCitH

The hypercube optimization can be applied to any MPCitH signature scheme that uses additive secret sharing between n parties, given that n can be expressed as $n = N^D$. The framework follows a geometrical approach arranging the N^D shares (also called leaf shares) in a hypercube of side N and dimension D to translate the original sharing between N^D parties to D sharings between N main parties. The main party shares for each of the D dimensions are constructed by grouping the original leaf shares into N slices of the hypercube along

the corresponding dimension. The additive secret sharing scheme allows to accumulate the shares within each slice, yielding D sets of sharings between N parties encoding the same original secret.

Within the hypercube, the leaf shares are indexed either using their absolute index s_i with $i \in [1, N^D]$ or by indexing them in the respective dimension $\mu \in [1, D]$ using the index $s_{\mathbf{i}}$ with $\mathbf{i} = (i_1, \ldots, i_D)$ and $i_\mu \in [1, N]$. The main shares are always indexed using the dimension μ and the index ν within this dimension $s_{\mu,\nu}$ with $\mu \in [1, D]$ and $\nu \in [1, N]$.

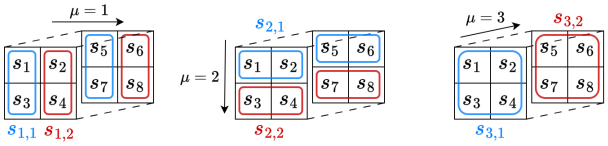


Fig. 1: Construction of main party shares on hypercube of $N = 2$ and $D = 3$

Figure 1 shows how a hypercube of side $N = 2$ and dimension $D = 3$ is used to construct 3 sets of 2 main shares from $2^3 = 8$ original shares/parties. The main shares for dimension $\mu = 1$ are $s_{1,1}$ and $s_{1,2}$ (left); along dimension $\mu = 2$ they are $s_{2,1}$ and $s_{2,2}$ (middle); dimension $\mu = 3$ yields $s_{3,1}$ and $s_{3,2}$ (right).

As all D main party sharings encode the same plain value, further optimization is possible. Performing a single computation using the plain value s allows to omit one party computation per dimension. Instead, this last share can be deduced from the difference of the first $N - 1$ parties computations and the plain computation, reducing the amount of required party computations from N^D to $1 + (N - 1) \cdot D$. For the example in Figure 1, the party computations are reduced from $2^3 = 8$ to $1 + (2 - 1) \cdot 3 = 4$.

2.5 Power Analysis Attacks

Power analysis attacks are categorized based on attacker capabilities: for each one of the classes, different countermeasures are suitable.

DPAs are characterized by the use of a large number of power traces to detect statistical dependencies on secret data [31]. Since power traces from multiple executions of the operation are required to mount a DPA, it can only be used to recover long-term secrets. Ephemeral secrets, which are randomly generated for each execution of the operation, cannot be probed using DPA. Protection against DPA is studied extensively and includes masking [13] as the most prominent countermeasure.

Contrary to DPA-type attacks, single-trace-type attacks can be performed using only a single power trace [14] allowing an attacker to probe ephemeral secrets

as well as long-term secrets. Classic Simple Power Analysis (SPA) as first proposed in [31] as well as template attacks [14] or more recent deep-learning based attacks [12] fall into this class of attacks. Masking is not considered to be an efficient countermeasure to protect against single-trace-type attacks and instead additional countermeasures like shuffling [25] have to be applied [8,39,5].

3 Sensitivity Analysis of Hypercube-SDitH

In the following section, we briefly introduce the Hypercube-SDitH signature scheme [17,33], as submitted to the first round of the NIST call for *Additional Digital Signature Schemes* [35]. Subsequently, we perform an analysis of the different power analysis attack paths possible in its signing procedure, based on the SDitH specification and reference implementation[5].

3.1 The Hypercube-SDitH Signature Scheme

Hypercube-SDitH uses the syndrome decoding (SD) problem and builds a signature scheme based on the MPCitH framework. The Fiat-Shamir heuristic is used to make the zero-knowledge proof non-interactive.

Construction of the Zero-Knowledge Proof The prover proves knowledge of a valid solution x to a public SD instance (H, y) such that $y = Hx$ and $\mathrm{wt}(x) = w$ where $\mathrm{wt}(x)$ is the Hamming weight i.e. the number of bits set in the variable x. To build a proof, three polynomials $Q(X)$, $S(X)$ and $P(X)$ are derived from the witness x and a public polynomial $F(X)$ is used such that $S(X) \cdot Q(X) = F(X) \cdot P(X)$. Verifying a valid solution amounts to verifying the polynomial $F(r) \cdot P(r) - S(r) \cdot Q(r) = 0$ for a series of challenge points r. This is done in a MPC setting, using a valid set of Beaver triples a, b, c such that $a \cdot b = c$, following the approach described in [7]:

$$[\![\alpha]\!]_i = \varepsilon \cdot [\![Q(r)]\!]_i + [\![a]\!]_i \qquad [\![\beta]\!]_i = [\![S(r)]\!]_i + [\![b]\!]_i \tag{1}$$

$$[\![v]\!]_i = \varepsilon \cdot [\![F(r) \cdot P(r)]\!]_i - [\![c]\!]_i + \alpha \cdot [\![b]\!]_i + \beta \cdot [\![a]\!]_i - \alpha \cdot \beta \tag{2}$$

Equation 1 shows the shares of the broadcast values, which are used to jointly compute the plain broadcast values $\alpha = \sum_{i=1}^{n} [\![\alpha]\!]_i$ and $\beta = \sum_{i=1}^{n} [\![\beta]\!]_i$ necessary for the computation of the parties output share $[\![v]\!]_i$ in Equation 2. If the shares recombine to $v = \sum_{i=1}^{n} [\![v]\!]_i = 0$, the proof is accepted. To reach the required level of soundness, the proof is repeated τ times.

[5] https://github.com/sdith/sdith

Signature Generation Below is a high-level description of the signature generation algorithm for the hypercube variant of the SDitH signature scheme. For the complete specification, we refer to its first round specification document [33].

- For each of the τ repetitions, $n = N^D$ leaf seeds are expanded from a root seed using a GGM tree [24][6]. The first $n-1$ leaf shares are directly expanded from the $n-1$ leaf seeds while the n-th leaf share is constructed additionally using the accumulated $n-1$ leaf shares as auxiliary value. The leaf shares are hashed to create the commitments.
- The $N-1$ main input shares of the D dimensions are built by accumulating the corresponding hypercube slices, while the main share containing the n-th leaf share (and thus the auxiliary value) is omitted.
- The MPC challenges (r, ε) for the τ parallel repetitions are expanded from the hashed commitments to the shares and are used to perform the plain broadcast computation as well as the party computations.
- The view challenges I^* for the τ parallel repetitions are expanded from the hashed MPC broadcast values. It corresponds to the index of the hidden leaf share $I^* = i^* \in [1, N^D]$ or equivalently indexed per dimension $\mu \in [1, D]$ in the hypercube the hidden main share $I^* = \mathbf{i}^* = (i_1^*, \ldots, i_D^*)$ where $i_\mu^* \in [1, N]$. All other leaf parties are included in the signature. The use of the GGM tree allows to do that efficiently, by revealing D sibling seeds.

3.2 Sensitivity Analysis

The signing algorithm takes the public key $(H', y,)$ and the key-derived witness (Q, S, P) as input and returns the signature $\sigma = (\mathtt{salt}, h_2, \mathtt{seed}_{\mathrm{sib}}, \mathtt{aux}, (\alpha, \beta), \mathtt{com}_{i \in I^*})$. We present the results of our sensitivity analysis of the signature generation in Figure 2. For simplicity, parallel repetitions ($\tau > 1$) are omitted. The seed expansion using the GGM tree produces N^D leaf seeds which are further expanded using an XOF (extendable output function) into N^D sets of $([\![\alpha]\!], [\![\beta]\!], [\![v]\!], [\![a]\!], [\![b]\!], [\![c]\!])$ (of which one is secret) and an auxiliary value $\mathtt{aux}$ (which may also be secret) to complete the sharing. Those leaf shares are hashed to create N^D commitments to the leaf shares, which are further hashed and expanded to the MPC challenges (r, ε) on which the plain broadcast computation Π_{plain} is performed using the secret plain (Q, S, P). The main shares are constructed for each dimension and the MPC computations Π are performed using them. Finally, the output is hashed to expand the view challenge I^*, which decides which exact main share per dimension is kept hidden i.e. remains secret.

In our analysis, we distinguish between i) secret variables such as secret-key derived witness-polynomials (Q, S, P), which are vulnerable to DPAs, ii) ephemeral secrets such as the seeds $\mathtt{seed}_i$, which are vulnerable to SPAs and iii) public values (see Section 2.5).

[6] A detailed description of the GGM tree construction is provided in Supplementary Material A.

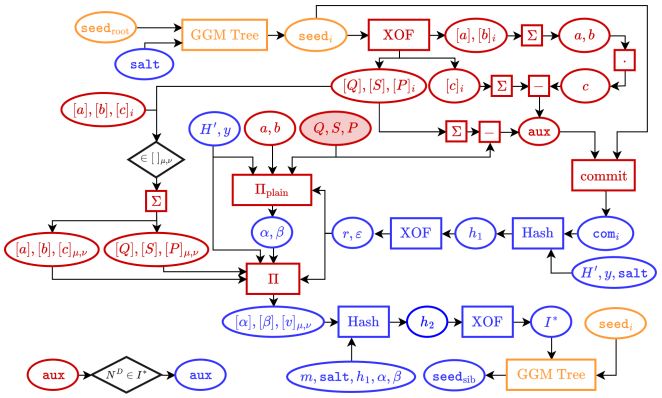


Fig. 2: Sensitivity analysis of the Hypercube-SDitH signing algorithm. Red filled: long-term secrets. Red: vulnerable to DPA. Orange: vulnerable to SPA. Blue: public (not vulnerable).

DPA Susceptibility Recovering the polynomial S will allow an attacker to recover the secret SD solution as shown in the attack on SDitH demonstrated in [23]. The recovery of polynomials P, Q might not allow full recovery of the secret key, but at least discloses information that decreases the key space. It further extends to shares of the witness-polynomials $([\![Q]\!], [\![S]\!], [\![P]\!])$ and the auxiliary value `aux`, which is constructed by accumulating those shares and plain values.

All but one of those shares are made public in the signature and therefore can be used by an attacker to recover the long-term secret, only using a single probe. More specifically, by using the revealed party shares (all-but-one) and guessing the encoded long-term secret, the power consumption (and thus value) of the hidden witness-polynomial share can be modeled. For example, consider a 3-party additive sharing of a secret variable x such that $x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3$. If shares $[\![x]\!]_1$ and $[\![x]\!]_2$ are public, an attacker could use DPA to verify a guess x' of the plain secret x by modeling the power consumption of the hidden share as $[\![x]\!]_3 = [\![x]\!]_1 + [\![x]\!]_2 + x'$. This applies to both the leaf sharing x_i (with $i \in [1, N^D]$) as well as the main sharings for one dimension μ with $x_{\mu,\nu}$ with $\nu \in [1, N]$.

Proposals for countermeasures against this type of attack include the application of masking to each party's share [4] or the introduction of a slack in revealed parties i.e. the use of an increased number of hidden parties to design a side-channel secure ZK protocol [42]. We rely on the latter to build an inherently DPA-resistant Hypercube ZK Protocol in Section 4.

Further, the Beaver triples (a, b, c) and their shares $([\![a]\!], [\![b]\!], [\![c]\!])$ are related to evaluations of the secret polynomials following Equation 1 and 2. As both the MPC challenge (r, ε) and the broadcast values (α, β, v) and their shares $([\![\alpha]\!], [\![\beta]\!], [\![v]\!])$ are public values, probing any of the Beaver triples allows an attacker to recover evaluations $Q(r)$, $S(r)$ or $P(r)$ and collecting $\deg(Q)$, $\deg(S)$ or

$\deg(P)$ such evaluations is enough to reconstruct the corresponding polynomial.

Single-Trace-Type Attack Susceptibility Ephemeral secrets are generally vulnerable to single-trace attacks. The seeds in the GGM trees are ephemeral, as they are expanded from a randomly sampled root seed and cannot be related to any long-term secret due to the one-way property of the hash function. Probing the root seed or any of the intermediate or leaf seeds potentially enables an attacker to recover secret hidden shares. Single-trace-type attacks are inherently more challenging and require complementary countermeasures such as shuffling or parallelism and are considered out-of-scope for this work.

GGM Seed Tree Expansion Although a probing attack on the root or intermediate seeds in the GGM tree poses a serious threat and thus requires countermeasures, such an attack can not be performed on the SDitH scheme using accumulated probes as is the case for DPA-type attacks. Hence, given that one-time expansion of the root seed can be guaranteed (by correctly sampling it from a proper source of randomness for each signature generation), any attack targeting the seed tree expansion is limited to a single-trace setting. In contrast to [20], we argue to *not* apply masking to the seed expansion procedure as only freshly sampled data is involved in this step, independent of any (potentially attacker-controlled) input.

4 Hardening the Hypercube ZK Protocol

The Hypercube zero-knowledge protocol reveals all-but-one of the secret main party shares used for the MPC computations for each dimension of the hypercube. As described in Section 3.2, this means an attacker can recover the (unshared) long-term secret through a single 'probe' on one of the hidden main or leaf shares. We now present a hardened version of the hypercube zero-knowledge protocol that introduces a *slack in revealed shares*, increasing the number of hidden shares in each main party sharing from one to $k+1$ to harden the respective sharing against k-order DPAs. The resulting hardened ZK protocol can be used to build MPCitH signature schemes utilizing the inherent side-channel protection provided by the secret sharing.

Slack in Revealed Parties The introduction of a slack in revealed parties has been proposed and discussed in the literature for several other variants of MPC-in-the-Head signature schemes [42,20]. For each dimension μ of the hypercube, instead of revealing all-but-one $(N-1)$ main party shares, an additional number of k shares stay hidden leading to a total of $k+1$ hidden shares in this sharing. Over all dimensions, this results in a number of $(k+1)^D$ hidden leaf shares. Each additional hidden share must be probed by an attacker in order to model the power consumption in a DPA attack. This provides k-order DPA protection for the set of main shares of each dimension (i.e. the respective party computations)

and DPA protection of order $(k+1)^D - 1$ for the set of leaf shares (i.e. the leaf share expansion and commitment).

4.1 Our Tweaked $(k+1)$-Hypercube ZK Protocol

Our tweaked $(k+1)$-Hypercube ZK Protocol described in Algorithm 1 largely conforms to the original Hypercube ZK Protocol. The N^D leaf shares are generated using a GGM tree and accumulated to form D sets of N main shares using which $(N-1) \cdot D$ party computations are performed. However, instead of revealing $N^D - 1$ leaf shares to the verifier by selecting a single hidden main share per dimension of the hypercube, $k+1$ hidden main shares per dimension are selected. The side N of the hypercube (i.e. the number of main shares per dimension) determines the maximum number of hidden main shares $k+1 < N$, as at least a single share per dimension must be revealed to contribute to the soundness of the proof.

The soundness error of verifying only the revealed $N-(k+1)$ main parties transcripts per dimension increases from $\frac{1}{N^D}$ for a single hidden leaf share to $\frac{(k+1)^D}{N^D}$ for $(k+1)^D$ hidden leaf shares, hence a larger number of parallel repetitions τ is needed to reach the target soundness level.

4.2 Optimal Selection of $k+1$ Hidden Main Shares in Hypercube MPCitH

We recall from Section 2.4 that the hypercube optimization enables the geometric re-arranging of the $n = N^D$ leaf shares to D sharings between N main parties, reducing the amount of party computations. As described in Section 3.1, leaf shares are expanded from a root seed using a GGM tree which allows for optimized revealing of the $N^D - (k+1)^D$ leaf shares that accumulate to the $N-k+1$ opened main party shares. When using a GGM tree to reveal all-but-$(k+1)^D$ leaf shares, instead of directly revealing $N^D - (k+1)^D$ leaf seeds, all sibling seeds along the $(k+1)^D$ hidden paths (i.e. the paths between each leaf seed and the root seed) are revealed when they are not part of any hidden path. Thus, the number of sibling seeds to reveal grows with both the number of hidden seeds and the level at which the hidden paths merge (in the worst case, at the root seed).

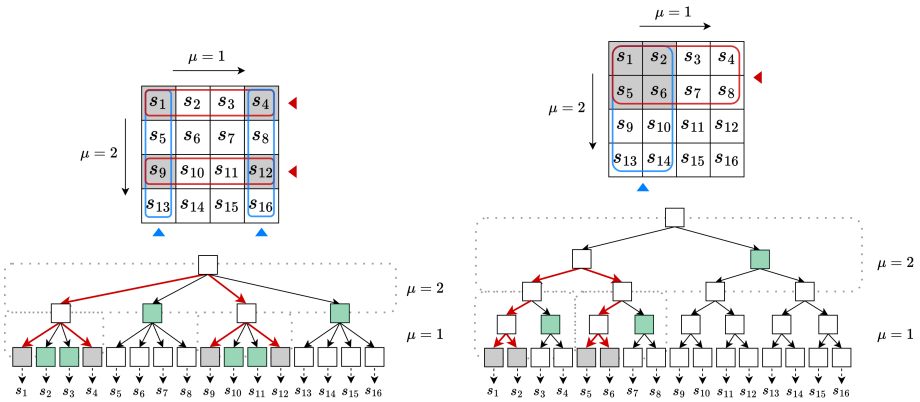
General Case In our tweaked $(k+1)$-Hypercube ZK Protocol, for each dimension of the hypercube $k+1$ main shares are randomly selected and remain hidden. This results in a total number of $(k+1)^D$ hidden leaf shares and yields an k-probing secure ZK protocol. The choice of slack k is restricted by the number of main shares to $k < N-1$. For a hypercube of e.g. $N = 4$ and $D = 2$, this method can be used to select $(k+1) = \{1, 2, 3\}$ hidden main shares. Figure 3a depicts this selection strategy for $(k+1) = 2$. In the N-ary GGM tree of D levels, the selection of $k+1$ hidden main shares corresponds to selecting $k+1$ hidden paths (red) and revealing the remaining $N-(k+1)$ sibling seeds on this level. This is

Algorithm 1: $(k+1)$-Hypercube ZK Protocol (SD-based)

Input: The prover knows the witness x and derives polynomials Q, P, S. Both parties have the public key (H, y). There exists a proof Π that outputs ACCEPT if the witness is valid.
Round 1: Prover generates shares and performs commitment.
Sample the leaf shares $[\![s]\!]_{i\in[1,N^D]}$, where $s = (P, Q, S, a, b, c)$.
Set the auxiliary aux $= s - \sum_{i=1}^{N^D-1} [\![s]\!]_i$.
Commit to all leaf shares $[\![s]\!]_i$.
Round 2: Verifier generates randomness $\mathrm{CH}_1 = (r, \varepsilon)$ for the MPC protocol.
Round 3: Prover emulates the MPC protocol.
Construct the main party shares $[\![s]\!]_{\mu,\nu}$ from the set of leaf shares with index in $\mathcal{I}_{\mu,\nu} = \{\mathbf{i} = (i_1, \ldots, i_D) \in [1, N]^D | (i_\mu = \nu)\}$ i.e. the leaf share's μ-th hypercube coordinate being ν as $[\![s]\!]_{\mu,\nu} = \sum_{i\in\mathcal{I}_{\mu,\nu}} [\![s]\!]_i$.
Perform the plain computation $(v, \alpha, \beta) = \Pi_{\mathrm{plain}}(s)$ using (r, ε).
Perform party computation $([\![v]\!], [\![\alpha]\!], [\![\beta]\!])_{\mu,\nu} = \Pi([\![s]\!]_{\mu,\nu})$ on each main share $[\![s]\!]_{\mu,\nu}$ using (r, ε).
Commit to v, α, β, $([\![v]\!], [\![\alpha]\!], [\![\beta]\!])_{\mu,\nu}$ and used randomness (r, ε).
Round 4: Verifier generates random view challenge $\mathrm{CH}_2 = I^*$.
Pick $(k+1)$ main party share indices $I^*_\mu = \{i^*_{\mu,1}, ..., i^*_{\mu,k+1}\}$ per dimension μ to remain hidden.
Round 5: Prover opens view of selected main parties.
Reveal $[\![s]\!]_i$ for $N^D - (k+1)^D$ non-hidden leaf parties.
Reveal α, β, v and $([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu}$.
Verification: Verifier reconstructs fully revealed main shares and recomputes $([\![v']\!], [\![\alpha']\!], [\![\beta']\!])_{\mu,\nu} = \Pi([\![s]\!]_{\mu,\nu})$.
Accepts if:
Revealed $N^D - (k+1)^D$ leaf party shares $[\![s]\!]_i$ are consistent with the committed shares.
Recomputed $([\![v']\!], [\![\alpha']\!], [\![\beta']\!])_{\mu,\nu}$ are consistent with committed $([\![v]\!], [\![\alpha]\!], [\![\beta]\!])_{\mu,\nu}$.
Revealed non-recomputable $([\![v]\!], [\![\alpha]\!], [\![\beta]\!])_{\mu,\nu}$ are consistent with committed $([\![v]\!], [\![\alpha]\!], [\![\beta]\!])_{\mu,\nu}$.
Proof Π performed by the prover outputs ACCEPT.

continued for each subtree below a hidden path and as a result, the number of sibling seeds to reveal will grow exponentially for each level of the tree (i.e. for each dimension of the hypercube): For dimension $\mu = 2$, $N - (k+1) = 2$ sibling nodes (green) are revealed and for dimension $\mu = 1$ for an additional number of $(k+1) \cdot (N - (k+1)) = 4$ must be revealed. This way, the naïve approach for selecting shares results in a total number of $n_{\mathrm{path}} = \sum_{\mu=1}^{D} (k+1)^{\mu-1} \cdot (N-(k+1))$ sibling seeds along the hidden paths that must be made public in the signature.

(Optimized) Selection of $k+1$ Main Shares from Binary Subtree We observe that this communication cost can be reduced for $N = 2^m$ and $(k+1) = 2^\ell$, by using a binary GGM tree structure and hiding ℓ-level subtrees for each dimension



(a) General case for selection of $k+1=2$ main shares. Hiding $2^2=4$ shares (gray) requires revealing 6 seeds (green).

(b) Optimized Selection of $k+1=2^\ell=2$ main shares from binary subtree of height $\ell=1$. Hiding $2^2=4$ shares (gray) requires revealing only 3 seeds (green).

Fig. 3: Selection of $k+1$ hidden main shares in hypercube of $N=4$ and $D=2$

such that for each dimension the $k+1$ hidden paths are merged until the lowest possible level. For a hypercube of side $N=4$ and dimension $D=2$, this method can be used to select $(k+1)=\{1,2\}$ hidden leaf shares (as shown in Figure 3b for $k+1=2$). Each dimension now corresponds to $m=\log_2(N)=2$ levels of a binary tree (dotted lines), were a hidden subtree of height $\ell=\log_2(k+1)=1$ requires revealing only $m-\ell=1$ sibling seeds (green) along the hidden paths (red) on each level. Using this optimized approach, a much smaller total number of $n_{\text{path}}=\sum_{\mu=1}^{D}(k+1)^{\mu-1}\cdot(\log_2(N)-\log_2(k+1))$ must be made public in the signature.

4.3 Security Proof

We now provide a formal security proof for completeness, soundness and the zero-knowledge property of the hardened $(k+1)$-Hypercube ZK Protocol. Our proofs are structured similarly to the initial SDitH proposal [17] and its hypercube variant [1]. We first show the (perfect) completeness of Algorithm 1, i.e. any honest prover performing the ZKP will be accepted by an honest verifier with certainty. Contrary to that, we show by proving the soundness that any prover without knowledge of the secret key will only be accepted with a probability lower than $\varepsilon\approx\frac{(k+1)^D}{N^D}$. Finally, we prove that the ZKP is zero-knowledge against an honest verifier that follows the protocol.

Definition 1 (Proof of Knowledge [18]). *Let x be a statement of language L in NP, and $W(x)$ the set of witnesses for x such that the following relation*

holds

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

Then a proof of knowledge for relation $\mathcal{R}$ with soundness error ε is a two-party protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with the following properties:

Completeness: If $(x, w) \in \mathcal{R}$, then an honest prover succeeds in convincing an honest verifier $\mathcal{V}$ of his knowledge, i.e.

$$\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x) \rangle \to 1] = 1$$

Soundness: If thee exists a PPT prover $\tilde{\mathcal{P}}$ such that

$$\tilde{\varepsilon} = \Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x) \rangle \to 1] > \varepsilon$$

then there exists an extractor, $\mathcal{E}$, which, given a rewindable black-box access to $\mathcal{P}$ outputs a witness w' for x in polynomial time with at least $1/2$.

Consistency with respect to the broadcast-based MPC protocol Π means that the opened views define a valid partial execution of Π, i.e. all local computations follow the protocol specification and all recomputed broadcast values agree with the public broadcast transcript. This consistency can be verified by reconstructing the execution of Π for the opened views and checking agreement with the broadcast values. Consistency with respect to the public input x means that the opened shares correspond to a valid secret sharing of x, which can be verified by checking consistency with the associated commitments.

Theorem 1 (Completeness). *An honest prover $\mathcal{P}$ with the knowledge of the witness w, who follows the protocol defined in Algorithm 1 will be accepted by an honest verifier $\mathcal{V}$ with a probability of 1.*

Proof. An honest verifier $\mathcal{V}$ will accept a transcript T of an execution of the protocol defined in Algorithm 1 if the views are consistent. In an honest execution a valid secret sharing of a witness w such that $\sum_{i=1}^{n} w_i = w$ is used and the MPC protocol Π is followed by all parties. Therefore it will always produce a transcript with consistent views which will be accepted by $\mathcal{V}$ with a probability of 1. □

Lemma 1. *A prover $\tilde{\mathcal{P}}$ that commits to a bad witness $S \cdot Q \neq P \cdot F$ and cannot find a hash collision has a probability of*

$$\mathsf{error} = p + (1 - p) \cdot \varepsilon_{\mathsf{cheat}} = \frac{(k+1)^D}{N^D}$$

of being accepted by the verifier $\mathcal{V}$.

Proof. A malicious prover $\tilde{\mathcal{P}}$ without knowledge of a valid set of polynomials such that $S \cdot Q = P \cdot F$ can convince an honest verifier $\mathcal{V}$ to accept if either: (i) with a false positive probability of p, for all challenge points the sharing $[\![v]\!]$

encodes the zero vector; or (ii) $\tilde{\mathcal{P}}$ cheats on the communications, so it appears that $[\![v]\!]$ encodes the zero vector.

During the verification, $\mathcal{V}$ checks the consistency of the views of the main parties. To detect cheating by $\tilde{\mathcal{P}}$ at any point in the protocol, it is sufficient for $\mathcal{V}$ to detect cheating in one of the dimensions. As $\mathcal{V}$ chooses a number of $k+1$ main shares to hide, such that it receives the views of the remaining main shares in the protocol, independently for each dimension consider the case where $D = 1$. With a probability of $(1-p)$, $\tilde{\mathcal{P}}$ has to cheat on the communications $[\![\alpha]\!]$, $[\![\beta]\!]$ or $[\![v]\!]$ of at least one of the main parties to make them encode the zero vector ($v = 0$). Note that the underlying shares for each of the MPCitH protocols are identical between the dimensions, therefore the main shares of each dimension encode the same plain value. Hence for $\tilde{\mathcal{P}}$ to cheat it is sufficient to consider that they cheat on at least one of the leaf shares, s_i.

Consider the case where $\tilde{\mathcal{P}}$ cheats on δ leaf shares in total. Since $\mathcal{V}$ chooses $k+1$ main shares to hide for each dimension, the probability that $\mathcal{V}$ finds at least one of the hidden shares among the $N-(k+1)$ views of main shares, therefore detecting cheating, is higher if the δ corrupted shares are distributed among more of the N main shares. Therefore, $\tilde{\mathcal{P}}$ will always attempt to consolidate the corrupted shares in as few main shares as possible. Hence, for the worst case analysis, we may assume that all δ shares lie in 1 main share.

Moreover, $\tilde{\mathcal{P}}$ is not bound to the leaf shares it has corrupted for each execution of the MPCitH protocol, i.e. each dimension, hence for the optimal chance of convincing the verifier, $\tilde{\mathcal{P}}$ would corrupt shares in each of the dimension such that the above consolidation is obtained. Therefore, we may assume that all δ lie in 1 main share for all dimensions.

General case. As $\mathcal{V}$ chooses $k+1$ shares uniformly randomly, and $\tilde{\mathcal{P}}$ has corrupted 1 main share, the probability that $\tilde{\mathcal{P}}$ is not detected is given by the probability that of all $k+1$ out of N shares $\mathcal{V}$ chooses, $\mathcal{V}$ does not choose the one share $\tilde{\mathcal{P}}$ has corrupted. Hence,

$$\varepsilon_{\mathsf{cheat,dim}} = \frac{\binom{N-1}{k+1-1}}{\binom{N}{k+1}} = \frac{k+1}{N}$$

$\mathcal{V}$ chooses $k+1$ main shares for each dimension $\mu \in [1, D]$ such that the total number of hidden leaf shares is $(k+1)^D$, hence for all dimensions, the probability that $\tilde{\mathcal{P}}$ is not detected is

$$\varepsilon_{\mathsf{cheat}} = \prod_{\mu=1}^{D} \frac{k+1}{N} = \frac{(k+1)^D}{N^D}$$

(Optimized) Selection of $k+1$ Main Shares from Binary Subtrees. For $N = 2^m$ and $k+1 = 2^\ell$, consider a binary tree with $2^l \geq |\{s_i\}_i|$ leaves, where $m = \log(N)$ levels are used to choose the hidden main shares for one dimension of a hypercube of side N. The height of the hidden subtree is given by $\ell = \log(k+1)$. For each

level of the subtree, $\mathcal{V}$ chooses half of the shares to see, halving the number of hidden shares in this dimension with each level. Hence,

$$\varepsilon_{\mathsf{cheat,dim}} = \prod_{\iota=1}^{\log(N)-\ell} \frac{1}{2} = \frac{1}{2^{\log(N)-\ell}} = \frac{2^{\ell}}{2^{\log(N)}} = \frac{k+1}{N}$$

This selection happens for each dimensions subtree, thus resulting in a soundness error of

$$\varepsilon_{\mathsf{cheat}} = \prod_{\mu=1}^{D} \frac{k+1}{N} = \frac{(k+1)^D}{N^D}$$

This results in an upper bound of

$$\mathsf{error} = p + (1-p) \cdot \varepsilon_{\mathsf{cheat}} = \frac{(k+1)^D}{N^D}$$

□

Lemma 2 (Splitting Lemma [40], as in [17,1]). *Suppose $S \subset \mathcal{U} \times \mathcal{V}$ satisfies $\Pr[(u,v) \in S] \geq \varepsilon$. Define $T := \{(u,v) \in \mathcal{U} \times \mathcal{V} \mid \Pr_{v' \in \mathcal{V}}[(u,v') \in \mathcal{V}]\} \geq (1-\alpha) \cdot \varepsilon$; for any $\alpha \in [0,1)$. Then the following hold: $\Pr[T] \geq \alpha \cdot \varepsilon$ and $\Pr[T|S] \geq \alpha$.*

Theorem 2 (Soundness). *If a prover $\tilde{\mathcal{P}}$ without the knowledge of a witness is capable of convincing a verifier $\mathcal{V}$ with a probability of $\tilde{\varepsilon} > \varepsilon$ there exists an extraction function E, where the average number of calls to $\tilde{\mathcal{P}}$ is upper bounded by an expression in poly$(\lambda, (\tilde{\varepsilon} - \varepsilon)^{-1})$.*

In the following, we prove that if a prover $\tilde{\mathcal{P}}$ can convince a verifier $\mathcal{V}$ with a probability greater than the soundness error ε, an extractor E can efficiently extract the witness which in turn is equivalent to $\tilde{\mathcal{P}}$ knowing the witness. Cheating with a probability smaller than the soundness error ε is not an issue for the soundness of the ZKP given ε is reasonably small.

Extraction condition. As defined in the original security proof for the hypercube framework [1], the extraction of the witness is possible given the opened state and communication for all parties under the additional condition that each party has to be verified in at least one accepting transcript. In our hardened hypercube, this condition is fulfilled whenever all main party shares for one dimension are opened in at least one transcript. Given a first challenge $I^* = (i_1^*, \ldots, i_D^*)$ with $i_\mu^* = \{i_{\mu,1}^*, \ldots, i_{\mu,k+1}^*\}$ consisting of $k+1$ hidden main shares per dimension μ this would require a maximum number of $\min(k+1)$ additional accepting transcripts. As we however want to relate the probability of extraction to the soundness error ε in the course of our proof, we pick a slightly stricter extraction condition which implies the one previously defined. For a first transcript with the challenge I^* and a total of $(k+1)^D$ hidden leaf shares, we require a maximum

number of $(k+1)^D$ additional accepting transcripts. For each hidden leaf share in I^* at least one transcript must be found, where this leaf share is opened. This means that successful extraction requires collecting $(k+1)^D$ additional successful transcripts, where each transcript must fulfill the described *partial extraction condition*.

Proof. In the following we assume the commitment scheme to be perfectly binding (instead of computationally binding), meaning the same commitments $\texttt{com}$ can only be produced by the same shares. For accepting transcripts which share their initial commitments to the leaf shares $h = \text{hash}(\texttt{com}_1, ..., \texttt{com}_{N^D})$ but differ in the reveal challenge that means either $[\![Q]\!]$, $[\![S]\!]$ and $[\![P]\!]$ are not consistent with the revealed shares and we found a hash collision or the revealed shares are consistent (i.e. the openings are unique) and $[\![Q]\!]$, $[\![S]\!]$ and $[\![P]\!]$ are uniquely defined. In the following we only focus on the extractor finding consistent shares. A witness ($[\![Q]\!]$, $[\![S]\!]$, $[\![P]\!]$) is defined to be a *good witness* if $Q \cdot S = P \cdot F$.

We define an extractor E that runs the prover $\tilde{\mathcal{P}}$ using a randomness R_h for the generation of leaf shares in the initial commitment. r_h is a possible realization of R_h. The extractor E performs the following steps:

1. Run $\tilde{\mathcal{P}}$ with an honest verifier $\mathcal{V}$ and any R_h until a successful transcript T_0 is found using a specific randomness r_h. This transcript has a challenge $I^* = \{i^*_1, ..., i^*_{(k+1)^D}\}$ with a total of $(k+1)^D$ hidden leaf shares.

2. Rewind $\tilde{\mathcal{P}}$ using the same randomness r_h until a successful transcript T_κ with a reveal challenge $J^* = \{j^*_1, ..., j^*_{(k+1)^D}\}$ is found, such that the κ-th hidden leaf share in transcript T_0 is not hidden in T_κ. Repeat this for all $(k+1)^D$ hidden leaf shares in T_0.

3. Extract the witness, if the witness is *bad* start again with step 1.

We now estimate the average number of calls to $\tilde{\mathcal{P}}$ the extractor E has to perform to extract the witness successfully. In the following, $\text{succ}^{T_\kappa}_{\tilde{\mathcal{P}}}$ denotes the event in which the prover $\tilde{\mathcal{P}}$ successfully convinces $\mathcal{V}$ in transcript T_κ. We also use $\text{succ}_{\tilde{\mathcal{P}}}$ for the general event of the prover succeeding, independent of any specific transcript.

According to our hypothesis, the prover can find a successful transcript with a probability of $\Pr[\text{succ}_{\tilde{\mathcal{P}}}] = \tilde{\varepsilon}$ on average. To apply the Forking Lemma, we first have to show that the set of random variables for which $\tilde{\mathcal{P}}$ succeeds exists with a non-negligible probability. We therefore define r_h is *good* if $\Pr[\text{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \geq (1-\alpha) \cdot \tilde{\varepsilon}$ where $\alpha \in (0,1)$. Using the Splitting Lemma (Lemma 2) we can derive $\Pr[r_h \text{ good} \mid \text{succ}_{\tilde{\mathcal{P}}}] \geq \alpha$. That implies good randomness can be found after approximately $\frac{1}{\alpha}$ successful transcripts. Assuming r_h is good, by Lemma 1 we can say $\Pr[\text{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \geq (1-\alpha) \cdot \tilde{\varepsilon} > \varepsilon$ which implies the witness encoded is a good witness, as for a bad witness $\Pr[\text{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \leq \varepsilon$.

To proceed, we want to bound the number of required calls to $\tilde{\mathcal{P}}$ until a transcript T_κ is found that fulfills the partial extraction condition $\text{PExt}_\kappa \coloneqq \{i^*_\kappa \notin J^*\}$ i.e. the κ-th hidden leaf share in T_0 is not hidden in T_κ. We assume a good randomness r_h has been found and use the counter probability of not finding a transcript T_κ that fulfills PExt_κ because the specific hidden leaf share in T_0 is again hidden in T_κ. This probability is equivalent to the probability $\varepsilon_{\text{cheat}}$ described in Lemma 1 with which an initially corrupted leaf share is not detected by $\mathcal{V}$, because this specific leaf share is hidden. As a result, a lower bound for the probability of finding a transcript that fulfills PExt_κ in every step is found in Equation 3.

$$\begin{aligned}
\Pr[\text{succ}_P^{T_\kappa} \cap \text{ PExt}_\kappa \mid r_h \text{ good}] &= \Pr[\text{succ}_P^{T_\kappa} \mid r_h \text{ good}] \\
&\quad - \Pr[\text{succ}_P^{T_\kappa} \cap \neg\text{PExt}_\kappa \mid r_h \text{ good}] \\
&\geq (1-\alpha)\cdot\tilde{\varepsilon} - \Pr[\neg\text{PExt}_\kappa \mid r_h \text{ good}] \\
&= (1-\alpha)\cdot\tilde{\varepsilon} - \Pr[\neg\text{PExt}_\kappa] \\
&\geq (1-\alpha)\cdot\tilde{\varepsilon} - \varepsilon_{\text{cheat}} \geq (1-\alpha)\cdot\tilde{\varepsilon} - \varepsilon
\end{aligned} \tag{3}$$

With this probability and the same good randomness r_h as for T_0, the probability of obtaining a transcript T_κ that fulfills PExt_κ is greater $\frac{1}{2}$ after a number of N_κ calls to $\tilde{\mathcal{P}}$.

$$N_\kappa \approx \frac{\ln(\frac{1}{2})}{\ln(1-((1-\alpha)\cdot\tilde{\varepsilon}-\varepsilon))} = \frac{\ln(2)}{\ln(\frac{1}{1-((1-\alpha)\cdot\tilde{\varepsilon}-\varepsilon)})} \leq \frac{\ln(2)}{(1-\alpha)\cdot\tilde{\varepsilon}-\varepsilon} \tag{4}$$

Let C denote the number of calls to $\tilde{\mathcal{P}}$ made by the extractor. We denote the expected value of calls as $\mathbb{E}[C]$ in a recursive formula, considering the scenario of failing a step and starting a new iteration of the outer loop with certain probabilities. For one iteration:

- The extractor makes one call to $\tilde{\mathcal{P}}$ to obtain T_0.
- If T_0 is not successful, which occurs with a probability of $1 - \Pr[\text{succ}_{\tilde{\mathcal{P}}}]$:
  - The extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$.
- If T_0 is successful, which occurs with a probability of $\Pr[\text{succ}_{\tilde{\mathcal{P}}}]$:
  - The extractor tries to link $(k+1)^D$ transcripts for $\kappa = [1, (k+1)^D]$:
  - If r_h is good, which occurs with a probability of α, in total the extractor makes at most $(k+1)^D \cdot N_\kappa$ calls to $\tilde{\mathcal{P}}$ and returns a transcript T_κ that fulfills PExt_κ with a probability of $\frac{1}{2^{(k+1)^D}}$.
  - Otherwise, with a probability of at most $1 - \frac{\alpha}{2^{(k+1)^D}}$ the extractor makes at most $(k+1)^D \cdot N_\kappa$ calls to P and continues with a new iteration of the outer loop making an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$.

$$\mathbb{E}[C] \leq 1 + \underbrace{(1 - \Pr[\text{succ}_{\tilde{\mathcal{P}}}]) \cdot \mathbb{E}[C]}_{T_0 \text{ not found}} + \underbrace{\Pr[\text{succ}_{\tilde{\mathcal{P}}}] \cdot ((k+1)^D \cdot N_\kappa + \overbrace{(1 - \frac{\alpha}{2^{(k+1)^D}}) \cdot \mathbb{E}[C])}^{T_1 \dots T_\kappa \text{ not found}}}_{T_0 \text{ found}} \tag{5}$$

$$\begin{aligned}
\mathbb{E}[C] &\leq 1 + (1 - \tilde{\varepsilon}) \cdot \mathbb{E}[C] + \tilde{\varepsilon} \cdot ((k+1)^D \cdot N_\kappa + (1 - \frac{\alpha}{2^{(k+1)^D}}) \cdot \mathbb{E}[C]) \\
&\leq \frac{1 + \tilde{\varepsilon} \cdot ((k+1)^D) \cdot N_\kappa}{\tilde{\varepsilon} - \tilde{\varepsilon} \cdot (1 - \frac{\alpha}{2^{(k+1)^D}})} = \frac{2^{k+1}}{\tilde{\varepsilon} \cdot \alpha} \cdot (1 + \tilde{\varepsilon} \cdot (k+1)^D \cdot N_\kappa) \\
&\leq \frac{2^{(k+1)^D}}{\tilde{\varepsilon} \cdot \alpha} \cdot (1 + \tilde{\varepsilon} \cdot (k+1)^D \cdot \frac{\ln(2)}{(1-\alpha) \cdot \tilde{\varepsilon} - \varepsilon})
\end{aligned}$$

We choose α to be $(1-\alpha) \cdot \tilde{\varepsilon} = \frac{1}{2} \cdot (\varepsilon + \tilde{\varepsilon})$ and $\alpha = \frac{1}{2} \cdot (1 - \frac{\varepsilon}{\tilde{\varepsilon}})$ yielding the upper bound in Equation 6.

$$\mathbb{E}[C] \leq \frac{2^{k+2}}{\tilde{\varepsilon} - \varepsilon} \cdot (1 + \frac{2 \cdot \tilde{\varepsilon} \cdot (k+1)^D \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}) \tag{6}$$

□

Theorem 3 (Honest-Verifier Zero Knowledge). *Given a (t, ε_{PRG})-secure PRG and a (t, ε_{com})-hiding commitment scheme, there exists an efficient simulator $\mathcal{S}$, which gets as input the two challenges (r, ε) and $I^* = \{i_1^*, ..., i_{(k+1)^D}^*\}$ and outputs a transcript such that it is indistinguishable from a real transcript of the protocol defined in Algorithm 1.*

Proof. In the following, we demonstrate that the simulator $\mathcal{S}_{\text{v3}}$ (Alg. 2) produces transcripts with a distribution that is indistinguishable from the distribution of real transcripts. We start with a simulator $\mathcal{S}_{\text{v0}}$ that produces real transcripts and adapt the algorithm step-by-step until we arrive at the algorithm $\mathcal{S}_{\text{v3}}$. For each adaptation, we argue why the distribution of simulators transcripts is unchanged in the previous version.

Simulator $\mathcal{S}_{\text{v0}}$ (True transcripts): To simulate true transcripts, $\mathcal{S}_{\text{v0}}$ takes as input a witness x_A as well as the honest verifiers challenges $\text{CH}_1 = \{r, \varepsilon\}$ and $\text{CH}_2 = I^*$. It executes Algorithm 1 correctly and produces transcripts with a distribution identical to real transcripts.

Simulator $\mathcal{S}_{\text{v1}}$: The seed-derived randomness of leaf shares $([\![P]\!], [\![Q]\!], [\![S]\!], [\![a]\!], [\![b]\!], [\![c]\!])_{i \in I^*}$ is replaced by true randomness. If the challenge $I^* = \{i_1^*, \dots, i_{(k+1)^D}^*\}$ contains the last leaf share index ($N^D \in I^*$),

Algorithm 2: HVZK Simulator $\mathcal{S}_{\text{v3}}$

Randomly sample root seed rseed $\xleftarrow{\$} \{0,1\}^\lambda$.
Step 1 (Sample Challenges):
Sample $\text{CH}_1 = \{r, \varepsilon\} \xleftarrow{\$} \mathbb{F}^t_{\text{points}} \times \mathbb{F}^t_{\text{points}}$ and $\text{CH}_2 = I^* = (i^*_1, \ldots, i^*_D)$ with $i^*_\mu \xleftarrow{\$} \binom{[N]}{k+1}$.
Step 2 (Generate Leaf Party States):
Expand root seed rseed to get N^D leaf states seed_i.
Step 3 (Generate Leaf Party Commitments and Witness Shares):
for $i \notin I^*$ **do**
  if $i \neq N^D$ **then**
    Compute $\text{com}_i = \text{Hash}(\text{seed}_i)$.
    Expand seed_i into witness shares $[\![s]\!]_i$.
  else
    Generate aux for the last party $i = N^D$, randomly sample $([\![P]\!], [\![Q]\!], [\![S]\!], [\![a]\!], [\![b]\!], [\![c]\!])_{N^D}$.
    Compute $\text{com}_{N^D} = \text{Hash}(\text{seed}_i || \text{aux})$.
for $i \in I^*$ **do**
  Randomly sample com_i.
  Randomly sample $([\![P]\!], [\![Q]\!], [\![S]\!], [\![a]\!], [\![b]\!], [\![c]\!])_i$.
Compute $\text{COM} = \text{Hash}(\text{com}_1, \ldots, \text{com}_{N^D})$.
Step 4 (Generate Party Communications):
for $(\mu, \nu) \in [1, D] \times [1, N]$ **do**
  if $\nu \in i^*_\mu$ **then**
    Randomly sample $([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu}$, except for one $[\![v]\!]_{\mu,\nu'}$ per dimension which is set $[\![v]\!]_{\mu,\nu'} = -\sum_{\bar{\nu} \neq \nu'} [\![v]\!]_{\mu,\bar{\nu}}$ where $\bar{\nu} \in [1, N]$.
  else
    Compute $([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu}$ as stated in Algorithm 1.
  Compute $h_{\mu,\nu} \leftarrow \text{Hash}(([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu})$.
Step 5 (Output transcript):
$\text{RSP}_1 = \text{Hash}(h_{1,1}, \ldots, h_{D,N})$
$\text{RSP}_2 = (\text{com}_{i \in I^*}, ([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu \in i^*_\mu}, \{\text{state}_i\}_{i \notin I^*})$
Output: $(\text{COM}, \text{CH}_1, \text{RSP}_1, \text{CH}_2, \text{RSP}_2)$

$([\![P]\!], [\![Q]\!], [\![S]\!], [\![c]\!])_{N^D}$ are still generated in the usual way, so the witness main party shares still sum up to the input witness and only shares $[\![a]\!]_{N^D}$ and $[\![b]\!]_{N^D}$ are random. In any case, for a challenge $I^* = (i^*_1, \ldots, i^*_D)$ the values $([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu \in i^*_n}$ computed using a true random leaf share are random. Distinguishing the distribution of transcripts from simulator $\mathcal{S}_{\text{v1}}$ from $\mathcal{S}_{\text{v0}}$ in running time t is bounded by ε_{PRG}.

Simulator $\mathcal{S}_{\text{v2}}$: In the next step $([\![S]\!], [\![Q]\!], [\![P]\!], [\![c]\!])_{N^D}$ are also replaced by true randomness. $\mathcal{S}_{\text{v2}}$ is now independent of the witness and only takes the challenges CH_1 and CH_2 as input. For $N^D \in I^*$ only $([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu \in i^*_\mu}$ are affected in the transcript (because aux is not send). In this case, the output distribution of $\mathcal{S}_{\text{v2}}$ is identical to $\mathcal{S}_{\text{v1}}$, because those shares are already randomly distributed

and independent of other parties in $\mathcal{S}_{\text{v1}}$. For $N^D \notin I^*$ also aux is affected in the transcript. In $\mathcal{S}_{\text{v1}}$, aux is computed by adding all true random leaf shares $[\![s]\!]_{i\in I^*}$ and all pseudo-random leaf party shares $[\![s]\!]_{i\notin I^*}$, therefore generating aux via true randomness does not alter the distribution between $\mathcal{S}_{\text{v1}}$ and $\mathcal{S}_{\text{v2}}$.

Simulator $\mathcal{S}_{\text{v3}}$: In the final step, all $([\![\alpha]\!], [\![\beta]\!], [\![v]\!])_{\mu,\nu\in i^*_\mu}$ except for one $[\![v]\!]_{\mu,\nu'\in i^*_\mu}$ per dimension μ are randomly sampled. The last $[\![v]\!]_{\mu,\nu'\in i^*_\mu}$ is set to $[\![v]\!]_{\mu,\nu'} = -\sum_{(\mu,\nu)\neq(\mu,\nu')}[\![v]\!]_{\mu,\nu}$. As these already are randomly distributed in $\mathcal{S}_{\text{v2}}$, the output distributions of $\mathcal{S}_{\text{v3}}$ does not change between $\mathcal{S}_{\text{v2}}$ and $\mathcal{S}_{\text{v3}}$. The outputs RSP_1 and RSP_2 of $\mathcal{S}_{\text{v3}}$ are thus indistinguishable from those of an honest execution of Algorithm 1. When applying the hiding property of the commitment scheme on $\text{com}_{(i_1,\ldots,i_D)\in I^*}$, we can conclude that the global HVZK simulator indeed outputs a transcript that is $(t, \varepsilon_{PRG}+\varepsilon_{com})$-indistinguishable from a real transcript of Algorithm 1. □

5 A Hardened SDitH Digital Signature Scheme

In the following, we present a d-probing secure Hypercube-SDitH signature scheme, that builds on the $(k+1)$-Hypercube ZK Protocol proposed in Section 4 with a slack of $k = d$ and binary GGM trees to efficiently reveal all-but-$(k+1)$ main shares per dimension from a subtree. The inherent probing security provided by our tweaked Hypercube-SDitH signature scheme applies to all parts of the scheme that operate on shares of the secret, i.e. the share expansion from the leaf seeds, the commitment of the leaf shares, the construction of the main shares and the MPC computations. Our tweak does not yield additional side-channel protection for the seed expansion via the GGM tree and the plain broadcast computation. The GGM seed expansion however, is inherently resistant to DPA, since no long-term secrets are present in this procedure. The plain broadcast computation operates on the plain secret and is not protected against DPA. This makes the application of masking with order $d_{\text{msk}} = d$ to this operation necessary to achieve DPA protection for the full scheme.

5.1 Parameter Selection: Speed vs. Bandwidth Trade-Off

In the original Hypercube-SDitH, the choice of hypercube dimension D allows for a trade-off between signature size and computational cost. Larger D requires larger GGM tree expansion, but also increases single-round soundness and thus allows to reduce the number of repetitions τ, leading to smaller signature sizes. In our hardened Hypercube-SDitH, the signature size grows exponentially for higher hypercube dimensions, preventing a reasonable trade-off using D. Instead, only the choice in side N of the hypercube allows for a trade-off: A large N and thus a larger number of leaf shares N^D provides large single-round soundness which allows for fewer parallel repetitions τ and smaller signature size. At the same time, the computational cost increases as a larger GGM tree is expanded and a larger number of $(N-1)$ MPC computations per dimension must be performed. A smaller N requires more repetitions τ and larger signatures, while

it decreases the computational cost for GGM expansion, commitments and MPC computations.

The additional side-channel protection provided by increasing the number of hidden shares comes at the cost of soundness. This is, because hidden shares are not verified and can be corrupted to cheat in the ZK protocol and increasing their number thus increases the chances of a cheating prover being accepted. The number of parallel repetitions τ is chosen to achieve a total soundness of the ZKP according to the required security level. Further, due to the use of Fiat-Shamir transforms to create a non-interactive signature scheme, the choice of parameters must also provide sufficient security against the forgery attack described by Kales et al. [28]. That requires a choice of τ such that the minimum cost of this attack as described in Equation 7 is greater than 2^{λ}, where p is the false positive probability of the used proof and ε is the soundness error of a single repetition of the MPC protocol (Section 5.1).

$$\text{cost}_{\text{forge}} := \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + (\frac{1}{\varepsilon})^{\tau-\tau'} \right\} \tag{7}$$

5.2 Performance Estimation and Analysis

In the following, we analyze and compare the performance (speed & signature size) of our novel Hypercube-SDitH signature scheme, which provides d-order hardening by design, against unprotected and masked Hypercube-SDitH.

Computational Overhead We first estimate the computational cost of the Hypercube-MPCitH signature scheme, which is mainly attributed to the following operations: the GGM tree expansion, the share generation and the commitments, all three due to their large use of symmetric primitives, and the MPC computations.

Unprotected SDitH The number of required MPC computations for the unprotected original Hypercube-SDitH signature scheme is equal to $((N-1) \cdot D+1) \cdot \tau$, and the number of share generations and commitments are both equal to $N^D \cdot \tau$.

Masked Hypercube-SDitH In SDitH, the MPC computations only involve linear operations, as can be seen in Eq. 1 - 2. For d_{msk}-order masking, that means each of the $(d_{\text{msk}}+1)$ shares can be processed independently requiring approximately $(d_{\text{msk}}+1)$-times the number of MPC computations during signing. The total estimated cost of the masked MPC computations during signature generation is:

$$\#\ \text{MPC}_{\text{sign, fully masked}} = ((N-1) \cdot D + 1) \cdot (d_{\text{msk}} + 1) \cdot \tau \tag{8}$$

Further, the share generation and commitment also require masking. To avoid the use of a masked hash function, which is potentially very expensive due to the

expected quadratic complexity, the authors in [20] propose the use of $(d_{\text{msk}}+1)$ parallel unmasked seed trees, randomness expansions and $(d_{\text{msk}}+1)$ parallel commitments of shares which comes with a linear overhead of $(d_{\text{msk}}+1)$. In total, the cost of share generations and commitments is $N^D \cdot (d_{\text{msk}}+1) \cdot \tau$.

Our New Hardened Hypercube-SDitH The proposed tweak, which introduces a slack in the revealed main parties, only provides inherent side-channel protection for parts of the signature generation (share expansion, commitment and party computations). In order to perform a fair comparison with the fully masked implementation, we also estimate the overhead of d-order masking the remaining operations during signing: the plain broadcast computation. This results in:

$$\#\ \text{MPC}_{\text{sign, PBC masked}} = [(N-1) \cdot D + (d+1)] \cdot \tau \tag{9}$$

The cost of share generation and commitments is calculated according to the unprotected case. Generally, for a slack of $k = 0$ the computational cost is identical to the unprotected scheme. For higher slack ($k > 1$), although the cost per round is identical, the overall cost is increased due to the higher number of parallel repetitions τ in order to account for the loss in soundness.

Bandwidth Cost Another important aspect of the performance of signature schemes is the signature size (in byte), as it directly impacts communication cost and required bandwidth.

Unprotected SDitH Each signature contains a salt, hash h_2 the auxiliary share in `view`, the broadcasted plain values (`broad_plain`), the released `path` in `view` and commitments. All contributions result in:

$$\text{size}_{\text{unprotected}} = \frac{\lambda}{2} + \tau \cdot [k_{\text{SD}} + 2w + (2d_{\text{SD}}+1) \cdot 4t + \frac{\lambda}{8} \cdot \log_2(N) + \frac{\lambda}{4}] \tag{10}$$

Masked Hypercube-SDitH The application of classical masking does not impact the signature size, as masked variables are unmasked before being made public in the signature. However, as proposed in [20], we assume the use of parallel seed trees and commitments to decrease computational overhead. This comes at the cost of a larger signature size as $(d_{\text{msk}}+1)$ seeds and commitments of the hidden share need to be included in the signature. In total, the signature size is:

$$\text{size}_{\text{msk}} = \frac{\lambda}{2} + \tau \cdot [k_{\text{SD}} + 2w + (2d_{\text{SD}}+1) \cdot 4t + D \cdot \frac{\lambda}{8} \cdot (d_{\text{msk}}+1) + \frac{\lambda}{4} \cdot (d_{\text{msk}}+1)] \tag{11}$$

Our New Hardened Hypercube-SDitH The signature size for our hardened Hypercube-SDitH signature scheme can be calculated as shown in Equation 12. Compared to the unprotected scheme, the number of sibling seeds increases to

$n_{\text{path}} = \sum_{\mu=1}^{D}(k+1)^{\mu-1} \cdot (\log_2(N) - \log_2(k+1)$. Signature size further increases by $(k+1)^D$ additional hidden leaf share commitments and $D \cdot k$ broadcast values.

$$\text{size} = \frac{\lambda}{2} + \tau \cdot [k_{\text{SD}} + 2w + (2d_{\text{SD}}+1) \cdot 4t \cdot (1 + D \cdot k) + \frac{\lambda}{8} \cdot n_{\text{path}} + \frac{\lambda}{4} \cdot (k+1)^D \quad (12)$$

Comparison Table 1 shows the comparison of computational cost and bandwidth between unprotected, fully masked and full slack Hypercube-SDitH instances, for different orders of protection k and proposed MPC parameters (N, D). The overhead of the full slack variants is mainly caused by the increase in parallel repetitions τ to account for the loss in soundness due to the larger number of $k+1$ hidden main shares per dimension. Hence full slack yields the best results for $N \gg k$ and thus higher soundness of a single round (at the cost of a higher baseline computational cost). Generally, as N grows the relative overhead of the hardened scheme for a fixed slack of k decreases, while the estimated overhead of masking is always linear. The choice of larger N however increases the baseline offline cost i.e. the MPC computations of the instance.

For larger N, the full slack variant substantially reduces the offline cost (i.e. share expansions and commitments) for all k compared to the masked variant. For smaller N, the lowest achievable offline cost for full slack is constrained by the requirement of N to be both a power of two and larger than $k+1$. Compared to the masked variant with $N = 2$, the number of MPC computations in the full slack instances (which require $N > 2$) is always higher, however for large GGM trees the calls to symmetric primitives dominate computational cost. However, especially for large slack, the signature size is significantly increased compared to the masked instances. This can be improved by combining a lower slack k with masking for higher probing security. For the D-th-root slack variant (detailed in Section 5.2), this allows to archive smaller signature sizes compared to masking for $k = 3$ for all instances (and further for $k = 7$ and $N = 64$) and lower offline cost at the expense of higher online cost due to the additional masking.

For $N = 64$ and $D = 2$ and 3rd-order DPA protection, the full slack instance increases the overhead in computational cost to around $41 - 45\%$ (in addition to $10\times$ higher baseline online cost due to the increased parameter N) compared to the estimated overhead of 300% for masking. Signature size is increased by 248% for full slack which is higher than the 119% overhead for the masked instance. The partial slack variant can improve this overhead to a lower overhead of 61% (10.9 kB), while it also improves the overhead in computational offline cost to 17% (but at the same time increases the overhead in online cost to 135%).

Partial slack As full slack Hypercube-MPCitH does not perform well for a hypercube of high dimension D (Section 5.1), we propose to combine a slack in revealed main shares with masking to enhance performance. The resulting probing security can be argued as following: A slack of k for D dimensions results

Table 1: Comparison of estimated cost for d-probing secure instances of Hypercube-SDitH (fast/balanced/short).

Scheme	**Param.** N	D	**d**	τ	**Offline Cost** # Exp.	# Com.	+%	**Online Cost** # MPC	+%	**Sig. Size** (byte)	+%
Hypercube-SDitH [33]	2	5	0	27	864	864	-	162	-	12 160	-
	2	8	0	17	4 352	4 352	-	153	-	8 496	-
	2	12	0	12	49 152	49 152	-	156	-	6 784	-
Masked Hypercube-SDitH	2	5	1	27	1 728	1 728	100	324	100	15 184	24.9
			3		3 456	3 456	300	648	300	21 232	74.6
			7		6 912	6 912	700	1 296	700	33 328	174.1
	2	8	1	17	8 704	8 704	100	306	100	11 216	32.0
			3		17 408	17 408	300	612	300	16 656	96.0
			7		34 816	34 816	700	1 224	700	27 536	224.1
	2	12	1	12	98 304	98 304	100	312	100	9 472	39.6
			3		196 608	196 608	300	624	300	14 848	118.9
			7		393 216	393 216	700	1 248	700	25 600	277.4
Full Slack Hypercube-SDitH	8	2	0	23	1 472	1 472	-	345	-	10 736	-
			1	33	2 112	2 112	43.5	528	53.0	20 920	94.9
			3	65	4 160	4 160	182.6	1 170	239.1	74 424	593.2
			7	∞	N/A	N/A	-	N/A	-	N/A	-
	16	2	0	17	4 352	4 352	-	527	-	8 496	-
			1	23	**5 888**	**5 888**	35.3	736	39.7	15 704	84.8
			3	33	**8 448**	**8 448**	94.1	1 122	112.9	40 456	376.2
			7	65	**16 640**	**16 640**	282.4	2 470	368.7	197 144	2220.4
	64	2	0	12	49 152	49 152	-	1 524	-	6 784	-
			1	14	**57 344**	**57 344**	16.7	1 792	17.6	10 928	61.1
			3	17	**69 632**	**69 632**	41.7	2 210	45.0	23 592	247.8
			7	23	**94 208**	**94 208**	91.7	3 082	102.2	76 424	1026.5
D-th-Root Slack Hypercube-SDitH	8	2	3	33	**2 112**	**2 112**	43.5	1 056	206.1	**20 920**	94.9
			7	65	**4 160**	**4 160**	182.6	2 340	578.3	74 424	593.2
	16	2	3	23	**5 888**	**5 888**	35.3	1 472	179.3	**15 704**	84.8
			7	33	**8 448**	**8 448**	94.1	2 244	325.8	40 456	376.2
	64	2	3	14	**57 344**	**57 344**	16.7	3 584	135.2	**10 928**	61.1
			7	17	**69 632**	**69 632**	41.7	4 420	190.0	**23 592**	247.8

in a number of $k+1$ hidden main shares for these dimensions and $(k+1)^D$ hidden leaf shares. Masking a set of N^D leaf shares or N main shares of dimension μ with a masking order d_{msk} or $d_{\text{msk},\mu}$ results in a number of $(d_{\text{msk}}+1)\cdot N^D$ leaf shares or $(d_{\text{msk},\mu}+1)\cdot N$ main shares. If masking and slack are combined, order-d_{msk} masking amplifies the probing security provided by the slack by a factor $(d_{\text{msk}}+1)$, as each hidden share is further split into $(d_{\text{msk}}+1)$ uniformly random shares a probing attacker must recover to probe one hidden share. The combined probing security d for the main shares of dimension μ amounts to $d_\mu = (d_{\text{msk},\mu}+1)\cdot(k+1)-1$ and $d = (d_{\text{msk}}+1)\cdot(k+1)^D-1$ for the leaf shares.

We propose the following variants of partial slack in the Hypercube-MPCitH framework:

1-Dimension Slack: The smallest signature sizes for a probing security of $d < N - 1$ can be archived if the slack tweak is only applied to dimension $\mu' = 1$ which is equivalent to the lowest levels of the GGM tree. This dimension μ' is fully protected by a slack of $k' = d$ (or $k' = 2^\ell$ for the subtree variant), while the other dimensions $\mu \in [2, D]$ must be protected by masking of order $d_{\text{msk},\mu} = d$. The combined probing security is $d = (k' + 1) - 1 = (d_{\text{msk},\mu} + 1) - 1$.

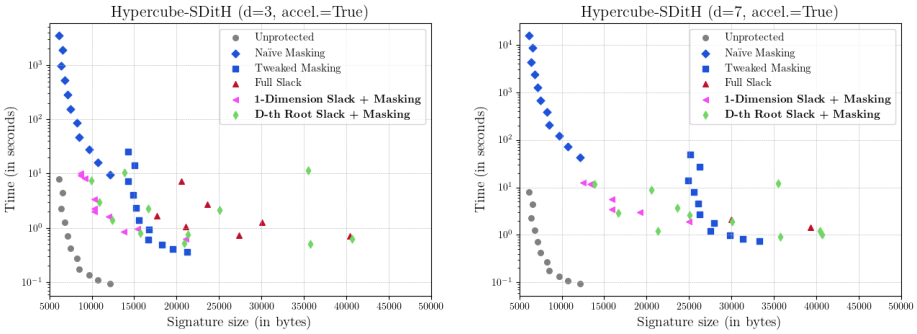
D-th-Root Slack: The best tradeoff between computational cost and signature size for d-probing security can be archived with full slack protection for the leaf sharing (and thus for the more expensive-to-mask share expansion and commitment) and a lower order of masking $d_{\text{msk},\mu}$ for the MPC computations to close the gap toward the lower inherent probing security for the main sharings such that $(d_{\text{msk},\mu} + 1) \cdot (k + 1) = (k + 1)^D = (d + 1)$. This is equivalent to a slack of $k = \lceil \sqrt[D]{d+1} \rceil - 1$ (or $k = 2^{\lceil \log_2(\sqrt[D]{d+1}) \rceil} - 1$ for the subtree variant) and a masking order of $d_{\text{msk},\mu} = \lceil \frac{d+1}{k+1} \rceil - 1$ for the MPC for all dimensions μ and $d_{\text{msk}} = 0$ for the rest of the scheme.

We integrate the Hypercube-SDitH scheme with different variants of partial slack in the framework from [20], to estimate its performance on a 32-bit RISC-V platform (at 250 MHz) with hardware acceleration for symmetric primitives. Figure 4 shows a comparison of different masking approaches (naive masking and the use of masking-friendly parallel seed trees as described in Section 5.2), full slack and the different variants of partial slack. All partial slack variants largely improve the performance over the full slack variant, to a point where interesting tradeoffs are achieved. Compared to the naïve masking approach all variants of partial slack can provide a significant reduction of computational cost at smaller signature sizes than the tweaked masking approach.

5.3 Implementation

We also implement the proposed $(k + 1)$-Hypercube ZK Protocol and hidden-subtree optimization (Section 4) in C and integrate it in the SDitH reference implementation to obtain a hardened Hypercube-SDitH signature scheme. Table 2 compares its performance to the original (unprotected) SDitH reference implementation, measured on an Intel Core i7-8550U (1.80 GHz).

We propose a *fast* instance that minimizes execution time by selecting the smallest possible value of N for a given slack parameter k. While this variant provides k-probing security with almost linear overhead for the offline phase (i.e. share expansion and commitment), its overall gains are limited by the comparably expensive online phase, which dominates the total runtime for this parameter set. Moreover, the resulting signature sizes increase significantly for large k. In contrast, full-slack Hypercube-SDitH is particularly effective for the *short* and *shortest* instances, where the offline phase becomes a significant or even dominant part of the signing time. To maintain compact signatures even for $k > 1$,



(a) Order of protection $d = 3$.

(b) Order of protection $d = 7$.

Fig. 4: Comparison between different variants of partial slack in Hypercube-SDitH ($n = 32$ to $n = 2^{15}$ for $N \in \{2, 4, 16, 32, 64, 128\}$) using the MPCitH estimator.

these instances use low-dimensional hypercubes ($D = 2$) together with large side ($N = 64$ or $N = 256$). Since $N \gg (k+1)$, the additional computational cost for increasing k grows sublinearly (after the initial large increase for $k = 1$ due to the increased N).

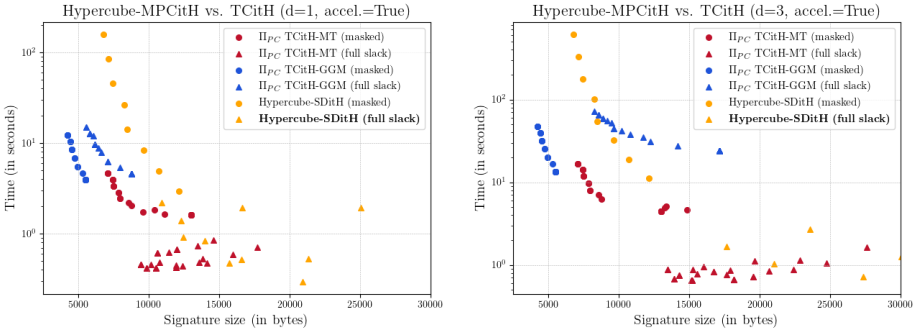
Table 2: Benchmarks (time and signature size) for the unprotected and hardened Hypercube-SDitH scheme on Intel Core i7-8550U at 1.80 GHz.

Scheme	**Param.** N	D	**k**	τ	**Sign time (in ms)** offline	online	total	**Verify time (in ms)**	**Sig. Size (in bytes)**
Hypercube-SDitH [33]	2	5	0	27	0.98	18.34	28.26	24.04	12 160
	2	8	0	17	4.78	17.13	32.05	29.29	8 496
	2	12	0	12	54.4	17.9	105.8	103.8	6 784
	2	16	0	9	668.4	18.67	1004.2	992.2	5 680
Full Slack Hypercube-SDitH (fast)	2	5	0	27	0.98	18.34	**28.27**	24.01	12 160
	4	3	1	44	3.10	49.31	**76.6**	48.31	35 792
	8	2	3	65	4.56	108.5	**165.2**	92.10	74 424
	16	2	7	65	17.77	222.3	350.83	193.7	197 144
Full Slack Hypercube-SDitH (short)	2	12	0	12	55.10	17.47	106.1	104.0	6 784
	64	2	1	14	60.79	195.2	374.2	365.1	**10 928**
	64	2	3	17	73.98	237.6	455.4	432.5	**23 592**
	64	2	7	23	100.3	322.0	617.2	556.5	76 424
Full Slack Hypercube-SDitH (shortest)	2	16	0	9	684.6	20.32	1030.4	1022.0	5 680
	256	2	1	11	778.0	620.4	2043.8	2017.2	**9 656**
	256	2	3	12	846.4	676.2	2225.3	2193.3	**18 592**
	256	2	7	14	996.6	799.6	2625.3	2565.3	50 576

5.4 Comparison to Hardened TCitH Signatures

We again use the framework from [20], to compare the performance of full slack Hypercube-SDitH to two full slack TCitH signature schemes proposed in the same work. Figure 5 shows estimated signature sizes and signing times for our hardened SDitH signature scheme (yellow) compared to the Π_{PC} TCitH scheme [19] for both its GGM variant (in blue) and MT variant (in red), at protection orders $k = 1$ and $k = 3$ and for different MPC parameters (N, D).

In comparison to the full slack TCitH-GGM scheme, the full slack Hypercube-SDitH offers significantly faster signing times at larger signature sizes. This is partly due to the fact that Feneuil et al. [20] recommend to apply masking to the seed expansion, despite it not being vulnerable to DPA (Section 3). Compared to the full slack TCitH-MT scheme, the full slack Hypercube-SDitH performs similar in terms of signature size with a slight increase in computational cost. In conclusion, full slack Hypercube-MPCitH archieves significantly better performance in computational cost compared to a fully masked implementation (unlike full slack TCitH-GGM which performes worse in computational cost and signature size), while the resulting signature sizes are comparable to those of full slack TCitH-MT.



(a) Order of protection $d = k = 1$.

(b) Order of protection $d = k = 3$.

Fig. 5: Comparison between full slack TCitH [20] ($N = 32$ to $N = 256$) and our full slack Hypercube-SDitH ($n = 32$ to $n = 4096$ for $N \in \{2, 4, 16, 32, 64, 128\}$) using the MPCitH estimator.

6 Conclusion

In this work, we first performed an analysis of the side-channel security of an SD-based Hypercube-MPCitH signature scheme, and identified DPA- and SPA targets. We showed that none of the known attacks on MPCitH signature schemes can be applied to mount a DPA on the fully ephemeral parts of the signature

scheme, i.e. the GGM tree. Operations on the shares of the secret however, were identified to be vulnerable to DPA attacks. To address this, we introduced a hardened Hypercube-MPCitH zero-knowledge protocol that provides k-th-order DPA resistance without the linear overhead inherent to masking. We further proposed a structured selection of $k+1$ hidden main shares for reduced signature size and discuss the combination of partial slack protection with masking to further improve performance. Applied to SDitH, for $N = 64$ and $D = 2$, the hardened signature scheme can archive high-order DPA-protection with an overhead of around $41 - 45\%$ in computational cost and 248% in signature size for 3rd-order protection using full slack, which can be improved to 16.7% overhead in offline cost and 61% in signature size for partial slack with an increased overhead of 135% in online cost due to masking.

Acknowledgments

This work has been partially supported by the Horizon 2020 ERC Advanced Grant (101020005 Belfort), the Cybersecurity Research Program Flanders (CRPF) with reference number VOEWICS02, Intel Corporation, the Research-I foundation from Infosys, the Initiation grant from IIT Kanpur, Prime Minister Early Career Research Grant from Anusandhan National Research Foundation, the Google India research fellowship, the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085 and by the FWO under an Odysseus project GOH9718N. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, the US Government, Cyber Security Research Flanders, or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes, notwithstanding any copyright annotation therein.

References

1. Aguilar-Melchor, C., Gama, N., Howe, J., Hülsing, A., Joseph, D., Yue, D.: The return of the sdith. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 564–596. Springer (2023)
2. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D.: Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. NIST Internal Report (IR) NIST IR 8413-upd1, National Institute of Standards and Technology, Gaithersburg, MD (2022). https://doi.org/10.6028/NIST.IR.8413-upd1
3. Alagic, G., Bros, M., Ciadoux, P., Dang, Q., Dang, T.H., Kelsy, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., Waller, N.: Status Report on the Second Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process. NIST Internal Report (IR) NIST IR 8610, National Institute of Standards and Technology, Gaithersburg, MD (2026). https://doi.org/10.6028/NIST.IR.8610

4. Aranha, D.F., Berndt, S., Eisenbarth, T., Seker, O., Takahashi, A., Wilke, L., Zaverucha, G.: Side-channel protections for picnic signatures. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 239–282 (2021)
5. Assael, G., Elbaz-Vincent, P., Reymond, G.: improving single-trace attacks on the number-theoretic transform for cortex-m4. In: 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 111–121. IEEE (2023)
6. Baum, C., Braun, L., de Saint Guilhem, C.D., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. In: Annual international cryptology conference. pp. 581–615. Springer (2023)
7. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: IACR International Conference on Public-Key Cryptography. pp. 495–526. Springer (2020)
8. Belaïd, S., Grosso, V., Standaert, F.X.: Masking and leakage-resilient primitives: One, the other (s) or both? Cryptography and Communications **7**(1), 163–184 (2015)
9. Belenky, Y., Dushar, I., Teper, V., Bugaenko, V., Karavaev, O., Azriel, L., Kreimer, Y.: Carry-based differential power analysis (cdpa) and its application to attacking hmac-sha-2. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(3), 1–29 (2023)
10. Benadjila, R., Feneuil, T., Rivain, M.: Mq on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. In: 2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P). pp. 468–485. IEEE (2024)
11. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The sphincs+ signature framework. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 2129–2146 (2019)
12. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without preprocessing. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 45–68. Springer (2017)
13. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Annual International Cryptology Conference. pp. 398–412. Springer (1999)
14. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International workshop on cryptographic hardware and embedded systems. pp. 13–28. Springer (2002)
15. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 acm sigsac conference on computer and communications security. pp. 1825–1842 (2017)
16. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals–dilithium: Digital signatures from module lattices (2018)
17. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Annual International Cryptology Conference. pp. 541–572. Springer (2022)
18. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 541–572. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_19
19. Feneuil, T., Rivain, M.: Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. Journal of Cryptology **38**(3), 28 (2025)

20. Feneuil, T., Rivain, M., Warmé-Janville, A.: Masking-friendly post-quantum signatures in the threshold-computation-in-the-head framework. Cryptology ePrint Archive (2025)
21. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z., et al.: Falcon: Fast-fourier lattice-based compact signatures over ntru. Submission to the NIST's post-quantum cryptography standardization process **36**(5), 1–75 (2018)
22. Gellersen, T., Seker, O., Eisenbarth, T.: Differential power analysis of the picnic signature scheme. In: International Conference on Post-Quantum Cryptography. pp. 177–194. Springer (2021)
23. Godard, J., Aragon, N., Gaborit, P., Loiseau, A., Maillard, J.: Single trace side-channel attack on the mpc-in-the-head framework. In: International Conference on Post-Quantum Cryptography. pp. 267–293. Springer (2025)
24. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions, p. 241–264. Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3335741.3335752
25. Herbst, C., Oswald, E., Mangard, S.: An aes smart card implementation resistant to power analysis attacks. In: International conference on applied cryptography and network security. pp. 239–252. Springer (2006)
26. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge proofs from secure multiparty computation. SIAM Journal on Computing **39**(3), 1121–1152 (2009)
27. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23. pp. 463–481. Springer (2003)
28. Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. In: International Conference on Cryptology and Network Security. pp. 3–22. Springer (2020)
29. Kannwischer, M.J., Genêt, A., Butin, D., Krämer, J., Buchmann, J.: Differential power analysis of xmss and sphincs. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 168–188. Springer (2018)
30. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation **48**(177), 203–209 (1987)
31. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual international cryptology conference. pp. 388–397. Springer (1999)
32. Lo, O., Buchanan, W.J., Carson, D.: Power analysis attacks on the aes-128 s-box using differential power analysis (dpa) and correlation power analysis (cpa). Journal of cyber security technology **1**(2), 88–107 (2017)
33. Melchor, C.A., Feneuil, T., Gama, N., Gueron, S., Howe, J., Joseph, D., Joux, A., Persichetti, E., Randrianarisoa, T.H., Rivain, M., et al.: The syndrome decoding in the head (sd-in-the-head) signature scheme. NIST Round 1 submission to the Additional Call for Signature Schemes (2023)
34. Miller, V.S.: Use of elliptic curves in cryptography. In: Conference on the theory and application of cryptographic techniques. pp. 417–426. Springer (1985)
35. National Institute of Standards and Technology: Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process (2022), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf, accessed: 2024-10-30

36. National Institute of Standards and Technology: Module-Lattice-Based Digital Signature Standard. Federal Information Processing Standards Publication (FIPS) NIST FIPS 204, Department of Commerce, Washington, DC (2024). https://doi.org/10.6028/NIST.FIPS.204
37. National Institute of Standards and Technology: Stateless Hash-Based Digital Signature Standard. Federal Information Processing Standards Publication (FIPS) NIST FIPS 205, Department of Commerce, Washington, DC (2024). https://doi.org/10.6028/NIST.FIPS.205
38. National Institute of Standards and Technology: PQC Standards. `https://csrc.nist.gov/projects/post-quantum-cryptography` (2025)
39. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: International Conference on Cryptology and Information Security in Latin America. pp. 130–149. Springer (2019)
40. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**(3), 361–396 (2000)
41. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
42. Seker, O., Berndt, S., Wilke, L., Eisenbarth, T.: Sni-in-the-head: Protecting mpc-in-the-head protocols against side-channel analysis. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1033–1049 (2020)

A Supplementary Material: GGM Tree

The GGM tree, introduced by Oded Goldreich, Shafi Goldwasser, and Silvio Micali in the seminal GGM construction [24]. It provides a standard method for constructing a pseudorandom function (PRF) from a pseudorandom generator (PRG). Starting from a master seed $\mathsf{seed}_{\mathrm{root}}$, the construction recursively expands seeds along a complete binary tree using a length-doubling PRG $G : \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ (here, λ is the security parameter). Let $G_0(s)$ and $G_1(s)$ denote the left and right halves of the output of $G(s)$, respectively. For a binary input string $x = x_1 x_2 \cdots x_l \in \{0,1\}^l$, the GGM PRF is defined by traversing the tree according to the bits of x, namely

$$F_{\mathsf{seed}_{\mathrm{root}}}(x) = G_{x_l}\big(G_{x_{l-1}}(\cdots G_{x_1}(\mathsf{seed}_{\mathrm{root}}))\big).$$

Equivalently, the root node is labeled with $\mathsf{seed}_{\mathrm{root}}$, and each internal node labeled by s generates two child nodes labeled by $G_0(s)$ and $G_1(s)$. The value associated with a leaf reached by the path x defines the PRF output on input x. The security of the GGM construction follows from the pseudorandomness of the underlying PRG: if G is a secure PRG, then the resulting function family F is computationally indistinguishable from a truly random function.

Figure 6 presents an example of the GGM tree expansion for a binary tree of depth three. Starting from the root seed $\mathsf{seed}_{\mathrm{root}}$, the recursive application of the PRG generates the depth-one nodes $G_0(\mathsf{seed}_{\mathrm{root}})$ and $G_1(\mathsf{seed}_{\mathrm{root}})$, the depth-two nodes G_{00}, G_{01}, G_{10}, and G_{11}, and ultimately the leaf seeds $\mathsf{seed}_0, \ldots, \mathsf{seed}_7$. Each leaf is associated with a unique binary path from the root.

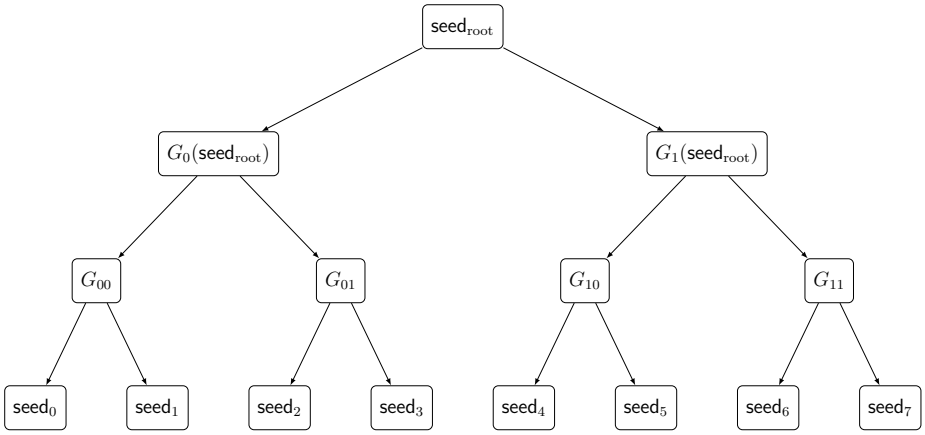


Fig. 6: GGM tree-based recursive seed expansion from the master seed $\mathsf{seed}_{\mathrm{root}}$ to the leaf seeds seed_i. Depth-two nodes are denoted as $G_{00} = G_0(G_0(\mathsf{seed}_{\mathrm{root}}))$, $G_{01} = G_1(G_0(\mathsf{seed}_{\mathrm{root}}))$, $G_{10} = G_0(G_1(\mathsf{seed}_{\mathrm{root}}))$, and $G_{11} = G_1(G_1(\mathsf{seed}_{\mathrm{root}}))$.