Short Signatures from Regular Syndrome Decoding, Revisited

Dung Bui[1], Eliana Carozza[1], Geoffroy Couteau[2], Dahmun Goudarzi[3], and Antoine Joux[4]

[1] IRIF, Université Paris Cité, Paris, France.
lastname@irif.fr
[2] CNRS, IRIF, Université Paris Cité, Paris, France.
couteau@irif.fr
[3] Quarkslab.
dahmun.goudarzi@gmail.com
[4] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany.
joux@cispa.de

Abstract. We revisit the construction of signature scheme using the MPC-in-the-head paradigm, and focus in particular on constructions from the regular syndrome decoding assumption, a well-known variant of the syndrome decoding assumption. We obtain two main contributions:

– We observe that previous signatures in the MPC-in-the-head paradigm must rely on a salted version of the GGM puncturable pseudorandom function (PPRF) to avoid collision attacks. We design a new efficient PPRF construction provably secure in the multi-instance setting. The security analysis of our PPRF, in the ideal cipher model, is quite involved and forms a core technical contribution of our work. While previous constructions had to rely on a hash function, our construction uses only a fixed-key block cipher and is considerably more efficient as a result. Our improved PPRF can be used to speed up many MPC-in-the-head signatures, and illustrate it on two signatures: the recent SDitH (submitted to the NIST), and a new signature scheme that we introduce.
– We introduce a new signature scheme from the regular syndrome decoding assumption, based on a new protocol for the MPC-in-the-head paradigm, which significantly reduces communication compared to previous works. Our scheme is conceptually simple, though its security analysis requires a delicate and nontrivial combinatorial analysis.

1 Introduction

In this work, we revisit signature schemes using the MPC-in-the-head paradigm, and focus in particular on schemes based on the regular syndrome decoding problem that were introduced in [CCJ23]. We introduce a new construction making use of several new techniques to provide more flexible parameter choices and improved performances. At a high level, our construction follows the blueprint of [CCJ23], but significantly improves on it in several aspects. At its core, our contribution hinges on two vital ingredients.

Our first ingredient is an improved construction of puncturable pseudorandom function (PPRF) tailored to MPC-in-the-head signatures. In puncturable PRFs used in MPC-in-the-head signature schemes, the main efficiency issue comes from the need to avoid collisions between the values of the nodes of the underlying GGM trees (within a signature or across different signatures). The simplest approach is to double the number of bits of these nodes compared to the desired security parameters; however, this would greatly impact the size of signatures. To bypass this increase, existing schemes [BDK+21,AGH+23,FJR22,CCJ23,BBdSG+23] usually make use of an extra salt value which is used as a secondary input in each pseudo-random generation of a new node from its parent node. Unfortunately, this requires using derivation functions with more bits of input than output. Typically, any cryptographic hash function provides the desired interface. The main drawback is that such hash functions are vastly slower than block ciphers, especially with modern hardware. As a consequence, current implementations of salted variants of the GGM PPRF are about 50 times slower compared to the fastest implementations of *unsalted* GGM PPRFs, which typically rely on a fixed-key AES block cipher [GKWY20]. We propose and analyze (in the ideal cipher model) a salted-GGM tree based on AES, which achieves the same performances as the best *unsalted* GGM PPRF constructions, without suffering from collision-based attacks. Our analysis is nontrivial and forms a significant technical contribution to this work. We use this improved PPRF as a core component of a new signature scheme that we introduce, but note that it can also be used to speed up essentially all previous signatures using MPC-in-the-head. We demonstrate it on the signature scheme SDitH [FJR22, BBdSG$^+$23] recently submitted to the NIST call for additional post-quantum signatures. We further expect our new PPRF to find other applications. We sketch one application in Appendix A.

We now turn to our second main contribution, a new construction of MPCitH-based signature from the regular syndrome decoding assumption. To obtain improved performances compared to previous works, we encode the regular syndrome decoding instances using a sparse representation on top of the dense representation used in [CCJ23]. Encoding regular syndrome decoding instances in a sparse manner is quite natural and relies on the use of an indicator vector to locate the non-zero positions. However, such a representation is not compatible with the secret sharing techniques that are used to split the key between the virtual parties that are introduced by the MPC-in-the-head paradigm: in order to use sparse representations, we need to develop new conversion techniques involving both types of representations. Along the way, we rely on a mechanism to prevent cheating behavior in the conversion, which requires a highly non-trivial combinatorial analysis. Overall, our signature scheme is more than 30% shorter compared to the scheme of [CCJ23] and can use significantly more conservative parameter sets, for similar runtimes.

Results and comparison. We provide a full implementation of our signature scheme. Our implementation is a proof-of-concept implementation, and did not use any optimizations such as batching, vectorization, or bitslicing. We expect that our proof-of-concept implementation could be significantly more optimized, for example by using batching or by taking advantage of the AVX2 instruction set. Nevertheless, our implementation confirms that our scheme exhibits excellent performances, even when compared to fully-optimized schemes such as the NIST submission SDitH [MFG$^+$23] that make use of batching techniques advanced hardware instructions such as AVX2. We outline below a sample of parameter tradeoffs:

- (fast) signature size 7.1kB, signing time 3 ms
- (medium) signature size 5.7 kB, signing time 19 ms
- (compact) signature size 5.1 kB, signing time 141 ms.

We refer the reader to Table 2 for more details on our parameters and implementation. We also compared our scheme to SDitH [AGH$^+$23], the fastest known code-based signature scheme to date, by running both schemes on the same hardware and for comparable parameter sets. To better isolate the effect of our improved PPRF, we also benchmark SDitH with their PPRF replaced by our improved construction,[5] as well as our scheme using the hash-based PPRF of SDitH. We summarize our benchmarks on Table 1. Even when comparing our unoptimized implementation to the optimized implementation of [MFG$^+$23], we observe $2\times$ to $4\times$ runtime improvements for $D = 8$ (with shorter signatures). The comparison should especially favor our scheme on machines that do not have access to the AVX2 instruction set (e.g. Mac), though a future optimized implementation could potentially achieve a similar speedup on computers with AVX2. We also observe that plugging our new PPRF in SDitH yields a $14\% \sim 17\%$ runtime improvement.

Another advantage of our signature scheme is its simplicity: while [AGH$^+$23] requires fast polynomial operations over large fields, our signature uses only very simple operations on strings such as XORs and cyclic shifts. Eventually, we note that our work and [AGH$^+$23] use incomparable variants of syndrome decoding: we use regular syndrome decoding over $\mathbb{F}_2$, while [AGH$^+$23] uses syndrome decoding over $\mathbb{F}_{256}$. Both variants have received much less attention than the standard syndrome decoding assumption over $\mathbb{F}_2$ (though we note that RSD over $\mathbb{F}_2$ seems to have received significantly more attention than the variant of [AGH$^+$23] over the past two decades).

Concurrent work. A concurrent and independent work [CLY$^+$24] recently introduced another signature scheme based on the Regular Syndrome Decoding assumption. On a technical level, our approaches differ significantly: [CLY$^+$24] combines the VOLE-in-the-Head technique from [BBdSG$^+$23] with a sketching method of [BGI16] to reduce the check of the noise structure to a system of degree-2 equations, which are then proven using the Quicksilver VOLE-based zero-knowledge proof [YSWW21].

[5] In the conference version of their work, the construction of [AGH$^+$23] initially used an *unsalted* GGM tree (instantiated using AES), which we show in Section 3 to be insecure (with a concrete attack that breaks the scheme using 2^{40} signatures in time 2^{69}). The authors later fixed this issue in their NIST submission [MFG$^+$23], using a proper salted GGM tree instantiated with a hash function.

Table 1. Comparison of the new signature scheme with SDitH for $D = 8$ and $D = 12$, with and without our improved multi-instance puncturable pseudorandom function (denoted AES-PPRF and hash-PPRF respectively). All schemes were run on one core of an Intel Core i7 processor 14700KF.

	D	τ	$\|\sigma\|$	signing time
SDitH (hash-PPRF)	8	17	8.2 kB	3.07 ms
(with AVX2)	12	11	6.0 kB	29.5 ms
SDitH (hash-PPRF)	8	17	8.2 kB	6.82 ms
(without AVX2)	12	11	6.0 kB	46.8 ms
Our scheme (hash-PPRF)	8	17	7.7 kB	4.07 ms
(without AVX2)	12	11	5.7 kB	43.83 ms
SDitH (AES-PPRF)	8	17	8.2 kB	2.63 ms
(with AVX2)	12	11	6.0 kB	24.5 ms
SDitH (AES-PPRF)	8	17	8.2 kB	6.05 ms
(without AVX2)	12	11	6.0 kB	37.9 ms
Our scheme	8	17	7.7 kB	1.65 ms
(without AVX2)	12	11	5.7 kB	19.1 ms

We use the MPC-in-the-Head methodology with a dedicated share-conversion technique. The signatures of [CLY+24] are shorter than ours, e.g., 4 kB versus 5.4 kB for comparable runtimes. Since our techniques are incomparable, we nevertheless expect that they could prove useful in future improved constructions of RSD-based signature, and leave as future work the question of exploring whether our combinatorial techniques could be used to further improve the scheme of [CLY+24]. We note that our improved PPRF can be used as a drop-in replacement for the one used in [CLY+24] (though it uses VOLE-in-the-Head, the methodology still relies on a similar use of a GGM-style PPRF under the hood) and its use should improve the performances of [CLY+24].

1.1 Organization

We introduce some preliminaries in Section 2, and provide a technical overview of our main two contributions in Section 3 (the improved GGM construction) and Section 4 (the new signature scheme) respectively. We then formally introduce and prove the security of our main results, in Section 5 and Section 6. This order of the presentation guarantees a better flow of the explanation because the security analysis of the signature scheme (and the statement of the theorem) relies on our new notion of multi-instance secure PPRF, which we, therefore, introduce first. Section 7 explains how to select parameters for our signature scheme, which requires some careful counting arguments.

2 Preliminaries

Notations. Given a set S, we write $s \leftarrow_r S$ to indicate that s is uniformly sampled from S. Given a probabilistic Turing machine $\mathcal{A}$ and an input x, we write $y \leftarrow_r \mathcal{A}(x)$ to indicate that y is sampled by running $\mathcal{A}$ on x with a uniform random tape, or $y \leftarrow \mathcal{A}(x; r)$ when we want to make the random coins explicit. Given an integer $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \cdots, n\}$. We use $\lambda = 128$ for the computational security parameter.

Vectors and matrices. We use bold lowercase for vectors and uppercase for matrices. We write $A||B$ to denote the horizontal concatenation of matrices A, B, and $A//B$ to denote their vertical concatenation. We denote by Id_n the $n \times n$ identity matrix. By default, we always view vectors as columns. Given a vector $\mathbf{v}$, we will often write $\mathbf{v} = (\mathbf{v}_1, \cdots, \mathbf{v}_n)$ to indicate that $\mathbf{v}$ is a (vertical) concatenation of n subvectors $\mathbf{v}_i$. We use this slight abuse of notation to avoid the (more precise, but cumbersome) notation $\mathbf{v} = (\mathbf{v}_1^\intercal, \cdots, \mathbf{v}_n^\intercal)^\intercal$. Given $\mathbf{u}, \mathbf{v} \in \{0,1\}^n$, we write $\mathbf{u} \oplus \mathbf{v}$ for the bitwise-XOR of $\mathbf{u}$ and $\mathbf{v}$, and $\mathsf{HW}(\mathbf{u})$ for the Hamming weight of $\mathbf{u}$, i.e., its number of nonzero entries.

Permutations. We let $\mathsf{Perm}(w)$ denote the set of all permutations of $[w]$. In this work, we typically use permutations over $[w]$ to shuffle the entries of a length-w vector, or even to shuffle the *blocks* of a vector which is the concatenation of w blocks. For example, given a vector $\mathbf{v} \in [\mathsf{bs}]^w$ and a permutation $\pi : [w] \mapsto [w]$, we write $\pi(\mathbf{v})$ to denote the vector $(v_{\pi(1)}, v_{\pi(2)}, \cdots, v_{\pi(w)})$. Given a vector $\mathbf{v}$ which is the concatenation of w subvectors $(\mathbf{v}_1, \cdots, \mathbf{v}_w)$, we write $\pi(\mathbf{v})$ to denote the vector $(\mathbf{v}_{\pi(1)}, \cdots, \mathbf{v}_{\pi(w)})$. We will typically apply this to vectors over $\mathbb{F}_2^K$, seen as the concatenation of w vectors over $\mathbb{F}_2^{\mathsf{bs}}$.

Code parameters. In this work, K always denotes the number of columns in the parity-check matrix H, and k denote the number of its rows. Equivalently, K is the codeword length, and $K - k$ is the dimension of the code. We let w denote the weight of the noise, which will always divide K. We let $\mathsf{bs} \leftarrow K/w$ denote the block size: a w-regular noise vector is sampled as a concatenation of w random unit vectors (the *blocks*) of length bs. We write Reg_w to denote the set of all length-K w-regular vectors.

Compact and expanded forms. Given an index $i \in [n]$, we let $\mathbf{e}_i \in \mathbb{F}_2^n$ denote the length-n unit vector over $\mathbb{F}_2$ whose i-th entry is 1. given w indices $(i_1, \cdots, i_w) \in [n]^w$, we extend the previous notation to $\mathbf{e}_\mathbf{i} = (\mathbf{e}_{i_1}, \cdots, \mathbf{e}_{i_w})$, the concatenation of w unit vectors. We typically manipulate noise vectors represented in *compact form*, *i.e.*, as elements $(i_1, \cdots, i_w)$ of $[\mathsf{bs}]^w$, where each entry $i_j \in [\mathsf{bs}]$ indicates the position of the 1 in the j-th length-bs unit vector. We let Expand denote the mapping which, given a noise vector $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_w) \in [\mathsf{bs}]^w$, outputs the vector $\mathbf{e}_\mathbf{x} = (\mathbf{e}_{\mathbf{x}_1}, \cdots, \mathbf{e}_{\mathbf{x}_w}) \in \mathbb{F}_2^K$. We call $\mathbf{e}_\mathbf{x}$ the *expanded form* of $\mathbf{x}$.

Cyclic shifts. Given a vector $\mathbf{u} \in \mathbb{F}_2^n$ and an integer $i \in [n]$, we write $\mathbf{u}{\downarrow}i$ to denote the vector $\mathbf{u}$ cyclically shifted by i steps (in other words, $\mathbf{u}{\downarrow}i$ is the convolution of $\mathbf{u}$ and $\mathbf{e}_i$). We also use notation $\mathsf{Shift}(\mathbf{u}, i)$ to denote $\mathbf{u}{\downarrow}i$. We extend the notation to a *block-by-block* cyclic shift of vectors: given a vector $\mathbf{u} \in \mathbb{F}_2^K$, viewed as a sequence of blocks $(\mathbf{u}_1, \cdots, \mathbf{u}_w) \in \mathbb{F}_2^{K/w} \times \cdots \times \mathbb{F}_2^{K/w}$, and a vector of shifts $\mathbf{x} \in [\mathsf{bs}]^w$, we write $\mathbf{u} \downarrow \mathbf{x}$ to denote the vector obtained by shifting the blocks of $\mathbf{u}$ according to $\mathbf{x}$. That is $\mathbf{u}{\downarrow}\mathbf{x} = (\mathbf{v}_1, \cdots, \mathbf{v}_w)$ where each $\mathbf{v}_i$ is the vector obtained by cyclically shifting (downward) the vector $\mathbf{u}_i$ by $\mathbf{x}_i$ steps. To avoid abusing parenthesis, we view $\downarrow$ as a "top priority" operator: by default, for any other operation op, $\mathbf{u}{\downarrow}\mathbf{x}\ \mathsf{op}\ \mathbf{v}$ means $(\mathbf{u}{\downarrow}\mathbf{x})\ \mathsf{op}\ \mathbf{v}$ and not $\mathbf{u}{\downarrow}(\mathbf{x}\ \mathsf{op}\ \mathbf{v})$.

Binary tree. Given a tree of size 2^D, for each leaf $i \in [2^D]$, we define $\mathsf{CoPath}(i)$ as co-path to i in the tree, *i.e.*, the set of intermediate nodes that can be used to recover all leaves except the $i-$th one. Denote bit-decompose i as $\sum_{j=1}^{D} 2^{j-1} \cdot i_j$ for $i_j \in \{0, 1\}$, the associated value of i-th leaf is defined as $X_i := X_{i_1,\ldots,i_D}$.

2.1 Basic Cryptographic Definitions

We cover a few additional standard preliminaries.

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

Definition 2 ((t, ϵ)-secure PRG). *Let $G : \{0,1\}^* \to \{0,1\}^*$ and let $l(.)$ be a polynomial such that for any input $s \in \{0,1\}^\lambda$ we have $G(s) \in \{0,1\}^{l(\lambda)}$. Then, G is a (t, ϵ)-secure pseudorandom generator if*

- *Expansion: $l(\lambda) > \lambda$;*
- *The distributions $\{G(s) | s \leftarrow \{0,1\}^\lambda\}$ and $\{r | r \leftarrow \{0,1\}^{l(\lambda)}\}$ are (t, ϵ)-indistinguishable.*

Definition 3 (Collision-Resistant Hash Functions). *A family of functions $\mathsf{Hash}_k : \{0,1\}^* \to \{0,1\}^{l(\lambda)}$; $k \in \{0,1\}^{\kappa(\lambda)}$ indexed by a security parameter λ is collision-resistant if there exists a negligible function v such that, for any PPT algorithm $\mathcal{A}$, we have:*

$$\Pr\left[\begin{array}{c} x \neq x' \\ \cap \mathsf{Hash}_k(x) = \mathsf{Hash}_k(x') \end{array} \middle| \begin{array}{c} k \in \{0,1\}^{\kappa(\lambda)} \\ (x, x') \leftarrow \mathcal{A}(k) \end{array}\right] \leq v(\lambda)$$

2.2 Regular Syndrome Decoding Problem

Given a weight parameter w, the syndrome decoding problem asks to find a solution of Hamming weight w (under the promise that it exists) to a random system of linear equations over $\mathbb{F}_2$. Formally, let $\mathcal{S}_w^K$ denote the set of all vectors of Hamming weight w over $\mathbb{F}_2^K$. Then:

Definition 4 (Syndrome Decoding Problem). *Let K, k, w be three integers, with $K > k > w$. The syndrome decoding problem with parameters (K, k, w) is defined as follows:*

- *(Problem generation) Sample $H \leftarrow_r \mathbb{F}_2^{k\times K}$ and $\mathbf{x} \leftarrow_r \mathcal{S}_w^K$. Set $\mathbf{y} \leftarrow H \cdot \mathbf{x}$. Output $(H, \mathbf{y})$.*
- *(Goal) Given $(H, \mathbf{y})$, find $\mathbf{x} \in \mathcal{S}_w^K$ such that $H \cdot \mathbf{x} = \mathbf{y}$.*

A pair $(H, \mathbf{y})$ is called an *instance* of the syndrome decoding problem. There exist several well-established variants of the syndrome decoding problem, with different matrix distributions, underlying fields, or noise distributions. In this work, we focus on a relatively well-studied variant known as the *regular syndrome decoding* (RSD) problem, introduced in 2003 in [AFS03] as the assumption underlying the FSB candidate to the NIST hash function competition.

The RSD problem is defined as the syndrome decoding problem, except that $\mathbf{x}$ is sampled randomly from the set Reg_w of w-regular vectors (*i.e.*, $\mathbf{x}$ is a concatenation of w unit vectors of length K/w). This variant has been used (and analyzed) quite often in the literature [AFS03,FGS07,MDCYA11,BLPS11, HOSS18, BCGI18, BCG+19b, BCG+19a, BCG+20b, YWL+20, WYKW21, RS21, CRR21, BCG+22].

Definition 5 (Regular Syndrome Decoding Problem). *Let K, k, w be three integers, with $K > k > w$. The syndrome decoding problem with parameters (K, k, w) is defined as follows:*

- *(Problem generation) Sample $H \leftarrow_r \mathbb{F}_2^{k\times K}$ and $\mathbf{x} \leftarrow_r \mathsf{Reg}_w$. Set $\mathbf{y} \leftarrow H \cdot \mathbf{x}$. Output $(H, \mathbf{y})$.*
- *(Goal) Given $(H, \mathbf{y})$, find $\mathbf{x} \in \mathsf{Reg}_w$ such that $H \cdot \mathbf{x} = \mathbf{y}$.*

2.3 The MPC-in-the-Head Paradigm

The MPC-in-the-head paradigm was initiated by the work of Ishai et al [IKOS07] and provided a compiler that can build honest-verifier zero-knowledge (HVZK) proofs for arbitrary circuits from secure MPC protocols. Assume we have an MPC protocol with the following properties:

- N parties $(P_1, \cdots, P_N)$ securely and jointly evaluate a function $f : \{0,1\}^* \to \{0,1\}$ on $\mathbf{x}$ while each party possess an additive share $[\![\mathbf{x}]\!]_i$ of input $\mathbf{x}$,
- Secure against passive corruption of $N-1$ parties i.e any $(N-1)$ parties can not recover any information about the secret $\mathbf{x}$.

Then the HVZK proof of knowledge of $\mathbf{x}$ such that $f(\mathbf{x}) = 1$ is constructed as:

- Prover generates the additively shares of the witness $\mathbf{x}$ into $([\![\mathbf{x}_1]\!], \cdots, [\![\mathbf{x}_N)]\!])$ among N virtual parties $(P_1, \cdots, P_N)$ and emulate the MPC protocol "in-the-head".
- Prover commits to the view of each party and sends commitments to the verifier.
- Verifier chooses randomly $(N-1)$ parties and asks the prover to reveal the view of these parties except one. Verifier later accepts if all the views are consistent with an honest execution of MPC protocol with output 1 and agrees with the commitments.

Security of MPC protocol implies that the verifier learns nothing about the input $\mathbf{x}$ from the $N-1$ shares, and MPC correctness guarantees that the Prover can only cheat with probability $1/N$. Security can then be amplified with parallel repetitions.

3 Technical Overview: Optimized GGM Trees for Faster MPCitH Signatures

Our starting point is the GGM puncturable pseudorandom function [KPTZ13,BW13,BGI14,GGM86], which is used in all modern MPC-in-the-head signatures. At a high level, all MPC-in-the-head protocols start by letting the prover generate shares of the witness, possibly together with shares of some appropriate preprocessing material, to be distributed among the n virtual parties. Then, in the last round, the prover will reveal $n-1$ out of n shares to the verifier. Since each share appears random, the share of each party P_i can be locally stretched from a short seed seed_i. To maintain correctness, an auxiliary "correction word" aux_n is appended to the seed seed_n of the last party P_n (e.g. to guarantee that the stretched shares XOR to the correct witness).

Puncturable PRFs allow us to significantly optimize this step. A *puncturable pseudorandom function* (PPRF) is a PRF F such that given an input x, and a PRF key k, one can generate a *punctured* key, denoted $k\{x\}$, which allows evaluating F at every point except for x, and does not reveal any information about the value $F.\mathsf{Eval}(k, x)$. Using a PPRF, the prover can define all seeds seed_i as outputs of the PRF, using a master seed seed^* as the PRF key. Then, revealing the key seed^* punctured at a point i suffices to succinctly reveal all seeds $(\mathsf{seed}_j)_{j \neq i}$ while hiding seed_i. Concretely, using the GGM PPRF [GGM86], the prover generates n seeds $\mathsf{seed}_1, \cdots, \mathsf{seed}_n$ as the leaves of a binary tree of depth $\lceil \log_2 n \rceil$, where the two children of each node are computed using length-doubling pseudorandom generators. This way, revealing all seeds except seed_i requires only sending the seeds on the nodes along the co-path from the root to the i-th leave, which reduces the communication from $\lambda \cdot (n-1)$ to $\lambda \cdot \lceil \log_2 n \rceil$.

3.1 On the use of salt to avoid collisions

It is well-known that MPC-in-the-head can suffer from collision attacks if the GGM PPRF is used *as is*: after about $2^{\lambda/2}$ signature queries, the adversary is likely to observe two signatures computed with the same master seed seed^*, an event which leaks the secret signing key. To circumvent this issue, previous works have relied in one way or another on a random 2λ-bit salt. However, the use of salt within the GGM PPRF is inconsistent across existing works. As a result, some constructions are either poorly specified or even insecure. Specifically:

- In Banquet [BDK+21], and in the more recent work of [AGH+23], the seeds $(\mathsf{seed}_1, \cdots, \mathsf{seed}_n)$ are generated from an *unsalted* GGM PPRF, and the salt is only used at the leaves, when stretching the share of each party P_i from its seed as $\mathsf{PRG}(\mathsf{seed}_i, \mathsf{salt})$.
- In [FJR22] and [CCJ23], the signature description loosely states that $(\mathsf{seed}_1, \cdots, \mathsf{seed}_n)$ are generated in a tree-based fashion using the master seed seed^* and the salt salt. However, the way the salt is used within the GGM construction is not specified precisely. In particular, the definition of the GGM tree in [FJR22] does not include the salt, and their implementation results only mention that it has been implemented "using AES in counter mode". The work of [CCJ23] does not have an implementation.

We observe that using the salt only at the leaves, as in [BDK+21, AGH+23], does not shield the signature from collision attacks. The attack is relatively simple:

- The attacker queries m signatures. Each signature will contain some number τ of $\lceil \log_2 n \rceil$-tuples of intermediate PRG evaluations (corresponding to the seeds on co-path to the unopened leave; τ corresponds to the number of repetitions of the underlying identification scheme). Let $(\mathsf{seed}^1, \cdots, \mathsf{seed}^k)$ denote all seeds received this way, where $k = m \cdot \tau \cdot \lceil \log_2 n \rceil$.
- The attacker locally samples random seeds seed and evaluates its two children $(\mathsf{seed}_0, \mathsf{seed}_1) \leftarrow_r \mathsf{PRG}(\mathsf{seed})$, until one of the seed_b collides with one of the seed^i.
- Since it knows the preimage of seed_b, it recovers the parent seed of seed^i, from which it can compute the seed associated with the unopened leave in one of the signatures.
- Given this seed, and using the salt salt associated to the signature (which is public), the attacker reconstructs all virtual parties' shares, and reconstructs the secret witness (the AES secret key in [BDK+21], the syndrome decoding solution in [AGH+23]. Using the witness, the attacker can now forge arbitrary signatures.

As should be clear from the above description, we note that adding salt to the leaves has absolutely *no* effect on the security of the signature against this collision attack. Efficiency-wise, after receiving m signatures, the attacker finds a collision in time $2^\lambda/(m \cdot \tau \cdot \lceil \log_2 n \rceil)$. For example, using $\lambda = 128$, $n = 2^{16}$, and $\tau = 9$ (this is a parameter set from [AGH+23]), after seeing only $m = 2^{40}$ signatures, the attacker can break the scheme in time $\approx 2^{69}$.

3.2 On the efficiency of salted GGM trees

We believe that the attack pointed above is mostly an issue of the presentation in the respective papers, and that the authors are generally aware of this issue. For example, we observe that the implementation of Banquet[6] correctly fixes the issue, by adding salt within all intermediate computations of the tree. As for [AGH+23], while their original implementation suffers from the attack above, the authors recently included their scheme in a NIST submission, whose implementation[7] properly deals with this issue. However, the state of affairs remains quite unsatisfying on the *efficiency* front: unsalted GGM trees can be instantiated very efficiently using fixed-key AES [GKWY20], which enables the use of Intel's AES-NI instruction set. Unfortunately, the fixed block size of AES makes it hard to add salt. And indeed, existing implementations such as Picnic [CDG+20], BBQ [DDOS19], Banquet [BDK+21], and the recent NIST submissions based on [AGH+23], all implement the PRG using a hash function (such as SHAKE), as follows: $\mathsf{seed}_b \leftarrow H(\mathsf{seed}\|i\|j\|b\|\mathsf{salt})$, where i is the index of the parent node, and $j \leq \tau$ is a counter for the repetitions of the identification scheme. Unfortunately, because of the hardware support for AES, replacing AES with a hash function is up to $50\times$ slower. This is especially problematic in recent protocols that use the hypercube technique [AGH+23], where the cost of generating the tree dominates the signing time.

3.3 A fast salted GGM tree in the ideal cipher model

We now turn to our contribution: we introduce a new construction of salted GGM tree which matches the efficiency of the fastest *unsalted* GGM trees, but which yields much stronger security guarantees. We provide formal security proof that our new construction is a *multi-instance secure* PPRF, a notion that we introduce. Multi-instance PPRFs can be used as a drop-in replacement for PPRFs in MPCitH signatures. In contrast with standard PPRFs, whose use incurs a security loss proportional to the number of signature queries (as illustrated by our attack), the unforgeability of MPCitH signatures tightly reduces to the multi-instance security of the PPRF.

In essence, our multi-instance PPRF is based on a very simple idea: use the previous top-performing GGM construction from a fixed-key block-cipher, and *use the cipher key as the salt.* While the intuition is very natural, formally proving security is actually quite challenging. Our full proof of security, in the ideal cipher model, is one of the core technical contributions of this work. It relies on the H-coefficient technique of Patarin [Pat09, CS14] and combines it with a balls-and-bins analysis to measure the number of seed and cipher key collisions, and tightly estimate their impact on security.

Starting Point: a PPRF in the Random Permutation Model. Our starting point is a PPRF construction from [GKWY20]. The construction of [GKWY20] is a tweak on the original GGM construction, where the PRG is instantiated with the following "Davies-Meyer" function:

$$G : x \to (\pi_0(x) \oplus x, \pi_1(x) \oplus x)\,.$$

In this construction, (π_0, π_1) are two fixed pseudorandom permutations. Using this PRG, the construction of PPRF proceeds in a tree-based fashion: sample a PPRF key $K \leftarrow_r \{0,1\}^\lambda$. On input $x = (x_1, \cdots, x_n)$, the PPRF F_K returns $G_{x_n}(G_{x_{n-1}}(\cdots G_{x_1}(K)\cdots))$, where G_0, G_1 denote the left and right half of the output of G, respectively. Puncturing x is done by computing all values on the co-path to x in the tree, *i.e.*, the values $G_{\bar{x}_i}(G_{x_{i-1}}(\cdots G_{x_1}(K)\cdots)$ for $i = 2$ to n: knowing the values on the co-path allows reconstructing the entire tree except for $F_K(x)$, whose values are pseudorandom under the security of G. To prove the security of the construction, the authors of [GKWY20] rely on the *random permutation model*, where (π_0, π_1) are modeled as two independent random permutations.

In [GKWY20], the motivation for introducing the construction is that in practice, π_0, π_1 can be instantiated using the AES block cipher with two fixed keys (K_0, K_1). This allows to evaluate G using two calls to AES, which is extremely fast using the AES-NI hardware instruction set (encrypting with AES using AES-NI takes as little as *1.3 cycle per Byte* according to [MSY21]). Furthermore, the entire construction requires only two executions of the AES key schedule. This GGM construction is to

[6] `https://github.com/dkales/banquet`

[7] `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/SDitH-spec-web.pdf`

date, by a significant margin, the fastest known PPRF, and it has been featured extensively in recent works on functions secret sharing [GI14, BGI15, BGI16, BGI19, BCG+21], pseudorandom correlation generators [BCGI18, BCG+19b, BCG+19a, BCG+20b, YWL+20, WYKW21, CRR21, BCG+22], and many more. It is also the construction suggested in [AGH+23], though as we saw above it is insecure in the context of signatures.

Observing that this fast PPRF construction is typically instantiated using a block cipher suggests the following idea, which is very natural in retrospect: use the above construction, but instantiate (π_0, π_1) using a block cipher (such as AES) and *use the block cipher keys* (K_0, K_1) *as a random salt.* This means that in each instance, the pair (K_0, K_1) will be sampled at random. When using AES, this changes nothing to the efficiency of the construction, since in each instance, one still only has to execute the AES key schedule twice. Yet, now, there is some hope that the use of fresh cipher keys in distinct instances can prevent the collision attack.

Multi-instance PPRF and PRGs. To formalize this idea, we introduce the primitive of multi-instance PPRF. At a high level, we define an N-instance PPRF as a PPRF that additionally takes as input a random salt. In the N-instance security game, N keys $(k_1, \cdots, k_N)$, inputs $(x_1, \cdots, x_N)$, and salts $(\mathsf{salt}_1, \cdots, \mathsf{salt}_N)$ are sampled randomly. The game also samples a bit $b \leftarrow_r \{0,1\}$. Then, the adversary receives $((x_1, \mathsf{salt}_1), \cdots, (x_N, \mathsf{salt}_N))$ and the N punctured keys $(k_1\{x_1\}, \cdots, k_N\{x_N\})$. If $b = 0$, the adversary additionally receives $(F_k(x_1, \mathsf{salt}_1), \cdots, F_K(x_N, \mathsf{salt}_N))$; else, if $b = 1$, the adversary receives N random outputs $(y_1, \cdots, y_N)$ instead. The adversary outputs a guess b' and wins if $b' = b$. The PPRF is said to be N-instance (t, ε)-secure if the advantage of any t-time adversary in this game is at most ε. Since our constructions use τ parallel calls to a PPRF with the same salt, we generalize the notion to (N, τ)-instance security to capture the setting where N instances of τ repetitions of a PPRF are used, where the salt differ across instances, but not across internal repetitions.

As a first step toward proving the security of our construction, we also introduce the similar (but simpler) notion of (N, τ)-instance (t, ε)-secure PRG, which is a PRG $G : (\mathsf{seed}, \mathsf{salt}) \rightarrow (G_0(\mathsf{seed}, \mathsf{salt}), G_1(\mathsf{seed}, \mathsf{salt}))$ that additionally takes some random salt. In the N-instance security game, the adversary attempts to distinguish $(G_0(\mathsf{seed}_i, \mathsf{salt}_i), G_1(\mathsf{seed}_i, \mathsf{salt}_i))_{i \le N}$ from random given the salts $(\mathsf{salt}_1, \cdots, \mathsf{salt}_N)$ (the game extends to (N, τ)-instance security in a straightforward way, but the description is more tedious). We show that the standard GGM reduction extends to the multi-input setting: an (N, τ)-input (t, ε)-secure PRG implies an (N, τ)-input $(t, D \cdot \varepsilon)$-secure PPRF on input domain $[2^D]$ via a straightforward sequence of hybrids.

A multi-instance PRG in the ideal cipher model. The crux of the analysis is then to show that our PRG is indeed (N, τ)-instance (t, ε)-secure (for a suitable choice of N, τ, t, ε). Since the PRG now explicitly uses a block cipher, we cannot rely on the random permutation model anymore; instead, we prove security in the *ideal cipher model*, where each key $K \in \{0,1\}^\lambda$ defines a truly random permutation π_K, and all parties are given oracle access to π_K and π_K^{-1} for all K (we measure the running time t of the attacker as its number of queries q to the oracles). Using the H-coefficient technique of Patarin, we formally prove that our construction is an (N, τ)-instance (q, ε)-secure PRG for any N up to $2^{\lambda-1}$, with $\varepsilon \le \frac{4\tau \cdot \lambda}{\ln \lambda} \cdot \frac{q}{2^\lambda}$, where the term $4\tau\lambda / \ln \lambda$ can be replaced by 8τ when $N \le 2^{\lambda/2}$ (the above inequality is an approximation, see Theorem 11 for the formal inequality). Our analysis is non-trivial, and the bound stems from a careful analysis of the influence of the number of collisions among seeds on the adversarial advantage. We show that this number can be bounded using standard lemmas on the maximum load of a bin when $2N$ random balls are thrown randomly into 2^λ bins.

Concretely, this means that one can use our new multi-instance PPRF construction as a drop-in replacement for previous (much slower) hash-based construction, at the (small) cost of a security loss of $4\tau D\lambda / \ln \lambda$ (or simply $8\tau D$ when we bound the number of signature queries by $2^{\lambda/2}$). For $D = 16$, $\tau = 8$, and $\lambda = 128$, this translates to a loss of 14 bits of security (when the number of queries is up to 2^{127}) or 10 bits of security (for up to 2^{64} queries). Additionally, we introduce another optimization that converts (N, τ)-instance (t, ε)-secure PRG to $(\tau \cdot N, 1)$-instance (t, ε)-secure by using a pseudorandom generator to sample the τ salts $(\mathsf{salt}^{i,e})_{e \le \tau}$ in a given instance from a global salt salt_i for each $i \le N$. This shaves a factor τ from security loss, which is reduced to 7 bits for $D = 16, \tau = 8, \lambda = 128$.

We believe that this is a very reasonable tradeoff in exchange for the benefits of using a much faster AES-based construction.

4 Technical Overview: New Signature from Regular Syndrome Decoding

We now move to our second main contribution, a new signature scheme from the regular syndrome decoding assumption. We start with a brief high-level overview of the RSD-based signature scheme from [CCJ23], since it serves as a starting point for our scheme. Let $H \in \mathbb{F}_2^{k\times K}$ be a matrix and $\mathbf{x} \in \mathbb{F}_2^k$ be a w-regular vector (*i.e.*, a concatenation of w unit vectors). We let $\mathsf{bs} \leftarrow K/w$ denote the *block size* of $\mathbf{x}$. The signature builds upon an efficient n-party protocol which, on input shares of $\mathbf{x}$, checks that (1) $\mathbf{x}$ is a regular vector, and (2) $H \cdot \mathbf{x} = \mathbf{y}$. This n-party protocol is then compiled into a zero-knowledge proof via the MPC-in-the-head paradigm (which we sometimes abbreviate MPCitH), and the proof is further compiled into a signature scheme via Fiat-Shamir. The main idea underlying the protocol of [CCJ23] is that each of (1) and (2) above can be checked very efficiently, provided that the parties are given a suitable sharing of $\mathbf{x}$ in each case:

- Given (entry-wise) shares of $\mathbf{x}$ over $\mathbb{Z}_{\mathsf{bs}}$, checking that a block of coordinates $x_1, \cdots, x_{\mathsf{bs}}$ has weight 1 boils down to checking that $\sum_{i=1}^{\mathsf{bs}} x_i = 1 \bmod \mathsf{bs}$, which is a linear equation over $\mathbb{Z}_{\mathsf{bs}}$.
- Given shares of $\mathbf{x}$ over $\mathbb{F}_2$, checking $H \cdot \mathbf{x} = \mathbf{y}$ simply amounts to checking a linear equation over $\mathbb{F}_2$.

Since in the MPC-in-the-head paradigm, checking linear equations is for free, the task of building the protocol reduces to the task of designing a *sharing conversion* protocol, which converts $\mathbb{F}_2$-shares of $\mathbf{x}$ into $\mathbb{Z}_{\mathsf{bs}}$-shares. The next observation of [CCJ23] is that converting shares mod-2 of some value x into shares mod-bs can be done very efficiently given precomputed shares mod-2 and mod-bs of the same random bit r, which the prover can generate by themself. The only missing ingredient is a mechanism to ensure that the prover honestly computes mod-2 / mod-bs pairs of the *same* identical random bit. The last, and most involved, observation of [CCJ23] is that the verifier can completely dispense with the need to perform this check, by picking a random permutation π of $[K]$ and instructing the prover to shuffle the pairs according to π before running the protocol. Using a careful and non-trivial combinatorial analysis, [CCJ23] showed that whenever $\mathbf{x}$ is sufficiently far from being a regular vector (meaning that it has many non-unit blocks), a malicious prover using $\mathbf{x}$ has negligible success probability over the choice of π, even if they use incorrect mod-2 / mod-bs pairs. Of course, this does not prevent a malicious prover from using an incorrect but close-to-regular witness. However, by choosing the parameters (K, k, w) in a highly injective setting it can be guaranteed that the *only* close-to-regular solution to $H \cdot \mathbf{x} = \mathbf{y}$ is a regular vector.

4.1 An Alternative Share-Conversion Approach

The approach of [CCJ23] yields a competitive signature scheme, but has its shortcomings. Its main efficiency bottleneck stems from the use of shares over $\mathbb{Z}_{\mathsf{bs}}$: because of that, the signature includes several (one for each of the τ repetitions of the basic proof) length-K vectors over $\mathbb{Z}_{\mathsf{bs}}$ (using a CRT trick, this can be reduced to $\mathbb{Z}_{\mathsf{bs}/2}$ whenever $\mathsf{bs}/2$ is odd and ≥ 3). This yields a $O(K \cdot \mathsf{bs})$ communication cost, which is (by a significant margin) the dominant cost of their protocol. To mitigate this cost, the authors set the block size bs to be the smallest possible value $\mathsf{bs} = 6$ (such that $\mathsf{bs}/2 = 3$). In turn, this forces them to rely on RSD with very high weight $w = K/6$, which requires significantly increasing the parameters to compensate for the security loss.

Our first observation is that all of these shortcomings can be eliminated at once by relying on an alternative share conversion approach. Because $\mathbf{x}$ is w-regular, it admits a compressed representation as a list of w integers in $[\mathsf{bs}]$, which indicates the position of the nonzero entry in each of the w unit vectors. Now, observe that if the parties hold shares of w integers $(i_1, \cdots, i_w)$ modulo bs, these can always be interpreted as representing some regular vector $\mathbf{x}$; in other words, given such shares, condition (1) is satisfied by default. The crux of our protocol is a conversion procedure that turns shares of this compressed representation into shares modulo 2 of the "decompressed" regular vector (with which the parties can check the linear equation $H \cdot \mathbf{x} = \mathbf{y}$ for free). Furthermore, this share conversion can again be implemented very efficiently if the parties are given shares of pairs of the same random unit vector in compressed representation and in standard representation. Concretely, given an integer $r \in [\mathsf{bs}]$, let $\mathbf{e}_r$ denote the length-bs unit vector with a 1 at position r. Assume that the n parties, holding shares of some $i \in [\mathsf{bs}]$, are given shares of r modulo bs, and shares of $\mathbf{e}_r$ over $\mathbb{F}_2$. Consider the following simple protocol:

- All parties broadcast their shares of $z = i - r \bmod \mathsf{bs}$ and reconstruct z.
- All parties locally shift cyclically their share of $\mathbf{e}_r$ by z.

After this protocol, all parties end up with shares of the vector $\mathbf{e}_r$ shifted by z, which we denote $\mathbf{e}_r \downarrow z$ (we view vectors as columns, hence the shift by z is downward). Observe that $\mathbf{e}_r \downarrow z = \mathbf{e}_r \downarrow (i-r) = (\mathbf{e}_r \uparrow r) \downarrow i = \mathbf{e}_{\mathsf{bs}} \downarrow i = \mathbf{e}_i$. As in [CCJ23], we will let the prover generate w random pairs $(r, \mathbf{e}_r)$ and share them between the virtual parties. To dispense with the need to check that the pairs were honestly generated, we rely on the same strategy and let the verifier sample a random permutation π of $[w]$, and instruct the prover to shuffle the pairs according to π before using them in the protocol. The high-level structure of the MPCitH-compiled zero-knowledge proof (without optimizations) is below:

- **Parameters and input:** let (K, k, w) be parameters for the syndrome decoding problem, and let $\mathsf{bs} \leftarrow K/w$. The prover holds a w-regular witness $\mathbf{x} \in [\mathsf{bs}]^w$ (in compressed representation) for the relation $H \cdot \mathbf{x} = \mathbf{y}$, where $H \in \mathbb{F}_2^{k \times K}$ and $\mathbf{y} \in \mathbb{F}_2^k$ are public. Let n be the number of virtual parties.
- **Round 1:** the prover samples w pairs $(r_i, \mathbf{e}_{r_i})$ where $r_i \leftarrow_r [\mathsf{bs}]$. We denote $(\mathbf{r}, \mathbf{e}_\mathbf{r})$ the vector of pairs. The prover generates n shares of $\mathbf{e}_\mathbf{r}$ (over $\mathbb{F}_2$) and of $\mathbf{x}, \mathbf{r}$ (modulo bs) distributed between the virtual parties, and commits to the local state of each party.
- **Round 2:** the verifier samples and sends to the prover a random permutation $\pi \leftarrow_r \mathsf{Perm}(w)$. We write $\pi(\mathbf{r})$ (resp. $\pi(\mathbf{e}_\mathbf{r})$) for the vector $(r_{\pi(1)}, \cdots, r_{\pi(w)})$ (resp. $(\mathbf{e}_{r_{\pi(1)}}, \cdots, \mathbf{e}_{r_{\pi(w)}})$).
- **Round 3:** the prover runs in their head the following protocol and commits to the views of all parties:
  - All parties reconstruct $\mathbf{z} = \mathbf{x} - \pi(\mathbf{r})$ and shift their shares of $\pi(\mathbf{e}_\mathbf{r})$, getting shares of $\pi(\mathbf{e}_\mathbf{r}) \downarrow \mathbf{z}$ (the shifting is done blockwise: each $\mathbf{e}_{r_{\pi(i)}}$ is cyclically shifted by z_i). Note that $\pi(\mathbf{e}_\mathbf{r}) \downarrow \mathbf{z} = \mathbf{e}_\mathbf{x}$ (*i.e.* the "uncompressed" representation of the witness $\mathbf{x}$).
  - All parties compute a share of $H \cdot (\pi(\mathbf{e}_\mathbf{r}) \downarrow \mathbf{z})$ and broadcast them. All parties check that the shares reconstruct to $\mathbf{y}$.
- **Round 4:** the verifier picks $i \leftarrow_r [n]$ and challenges the prover to open the views of all parties except i.
- **Round 5:** the prover sends the $n-1$ openings to the verifier, who checks that the views are consistent with the commitments, with each other, and with the output of the protocol being $\mathbf{y}$.

The soundness of the scheme is $\varepsilon = \mathsf{p} + (1/n) \cdot (1 - \mathsf{p})$, where $\mathsf{p} = \mathsf{p}(K, k, w)$ is an upper bound on the probability (over the choice of the random permutation π) that a cheating prover, that commits in the first round to an incorrect witness (*i.e.* a compressed vector $\mathbf{x}^*$ such that $H \cdot \mathbf{e}_{\mathbf{x}^*} \neq \mathbf{y}$), manages to generate a valid MPC transcript (*i.e.* finds —possibly incorrect— pairs $(\mathbf{r}, \mathbf{u})$ such that $H \cdot (\pi(\mathbf{u}) \downarrow \mathbf{z}) = \mathbf{y}$, where $\mathbf{z} = \mathbf{x}^* - \pi(\mathbf{r})$). The crux of our analysis lies in computing a tight evaluation of p.

In our final signature, we incorporate multiple optimizations on top of this basic template, including the usual optimization of generating the shares in a tree-based fashion using the GGM puncturable pseudorandom function [KPTZ13, BW13, BGI14, GGM86], but also the more recent hypercube technique from [AGH+23], and a number of additional optimizations tailored to our scheme.

In terms of signature size, the dominant cost stems from the size of a share of $\mathbf{x}$ and of w pairs $(r, \mathbf{e}_r)$ (using standard optimizations, all shares except one can be compressed, hence the communication is dominated by the size of a single share, ignoring for now the number of repetitions of the identification scheme). The size of a share of $\mathbf{x}$ together with w pairs $(r, \mathbf{e}_r)$ is $2w \log \mathsf{bs} + K$ bits[8], whereas the size of $\mathbf{x}$ (now shared as a vector over $\mathbb{F}_2^K$) and of the pairs in [CCJ23] is $K \cdot (2 + \mathsf{bs}/2)$ bits. This directly incurs a significant reduction in the signature size. Furthermore, with this alternative conversion, using a very small block size is not advantageous anymore, which allows us to explore a much wider range of parameters, resulting in further savings.

[8] As in [CCJ23], this number is multiplied by a number τ of repetition, but since it is the same in both works, we ignore it in this discussion for simplicity.

4.2 Combinatorial Analysis

Although the high-level strategy —shuffling the random pairs— is the same as in [CCJ23], the security analysis is *entirely* different and forms a core technical contribution of our work. Shuffling the prover-generated correlated randomness is a highly non-generic technique, where each new protocol requires a new and dedicated combinatorial analysis.[9] The crux of the proof lies in bounding the success probability of a cheating adversary $\mathcal{A}$ in the following game:

- $\mathcal{A}$ holds a vector $\mathbf{x}^* \in [\mathsf{bs}]^w$ and chooses $\mathbf{r} \in [\mathsf{bs}]^w$ and $\mathbf{u} \in \mathbb{F}_2^K$, such that $\mathbf{u}$ is *not* a regular vector.
- A uniformly random permutation π is sampled from $\mathsf{Perm}(w)$.
- $\mathcal{A}$ wins iff $H \cdot (\pi(\mathbf{u}){\downarrow}(\mathbf{x}^* - \pi(\mathbf{r}) \bmod \mathsf{bs})) = \mathbf{y}$.

Given a bound on $\mathcal{A}$'s winning probability in this game, the rest of the proof follows in a relatively standard way and is similar to previous security proofs of code-based signatures schemes in the MPCitH paradigm, such as [CCJ23] (we still provide a full proof in the paper for completeness). Above, note that for any vector $\mathbf{s} \in [\mathsf{bs}]^w$, $\pi(\mathbf{u}) \downarrow \mathbf{s}$ is a regular vector if and only if $\mathbf{u}$ is a regular vector. Note also that whether $\mathbf{x}^*$ is actually a correct witness or not (*i.e.* whether $H \cdot \mathbf{e}_{\mathbf{x}^*}$) does not matter: as long as $\mathbf{u}$ is regular, if $\mathcal{A}$ wins the game above, then an extractor can recover a valid regular solution $\pi(\mathbf{u}){\downarrow}(\mathbf{x}^* + \mathbf{r} \bmod \mathsf{bs})$ to the syndrome decoding problem (hence $\mathcal{A}$ "knew" a solution to the problem in the first place). Eventually, note that

$$\pi(\mathbf{u}){\downarrow}(\mathbf{x}^* - \pi(\mathbf{r}) \bmod \mathsf{bs}) = \pi(\mathbf{u}{\uparrow}\mathbf{r}){\downarrow}\mathbf{x}^*,$$

hence, the game above simplifies to the following: $\mathcal{A}$ chooses $\mathbf{x}^* \in [\mathsf{bs}]^w$ and $\mathbf{u} \in \mathbb{F}_2^K \setminus \mathsf{Reg}_w$, and wins iff $H \cdot (\pi(\mathbf{u}){\downarrow}\mathbf{x}^*) = \mathbf{y}$ holds over the choice of a random permutation π.

Eliminating spurious solutions. An immediate issue with the above game is that an adversary *might* win with a very high probability, if the system of equations $H \cdot \mathbf{x} = \mathbf{y}$ admits solutions that are mostly invariant by blockwise permutation. Concretely, assume that there exists a vector $\mathbf{u}^*$ which satisfies $H \cdot \mathbf{u}^* = \mathbf{y}$, and such that $\mathbf{u}^*$ is not a regular vector, yet $\mathbf{v}^*$ is a concatenation of w *identical* vectors from $\mathbb{F}_2^{\mathsf{bs}}$. If this happens, then there is an easy winning strategy: $\mathcal{A}$ sets $\mathbf{u} \leftarrow \mathbf{u}^*$ and $\mathbf{x}^* \leftarrow 0^w$. Since $H \cdot (\pi(\mathbf{u}){\downarrow}\mathbf{x}^*) = H \cdot \pi(\mathbf{u}) = H \cdot \mathbf{u}^* = \mathbf{y}$, $\mathcal{A}$ is guaranteed to win. More generally, if $H \cdot \mathbf{x} = \mathbf{y}$ admits a solution $\mathbf{u}$ whose blocks are *mostly* identical, then the equation $H \cdot \pi(\mathbf{u}^*) = \mathbf{y}$ has a relatively large chance to hold simply because $\pi(\mathbf{u}^*)$ has a relatively large chance to be equal to $\mathbf{u}^*$.

Setting up some notations. Given a vector $\mathbf{u}$, we let $\mathsf{pn}(\mathbf{u})$ denote $|\{\pi(\mathbf{u}) \mid \pi \in \mathsf{Perm}([w])\}|$. That is, $\mathsf{pn}(\mathbf{u})$ is the number of distinct vectors in $\mathbb{F}_2^K$ which can be obtained by shuffling $\mathbf{u}$ blockwise; we call $\mathsf{pn}(\mathbf{u})$ the *permutation number* of $\mathbf{u}$. Then, given a bound B, we define $\mathsf{PN}_B = \{\mathbf{u} \mid \mathsf{pn}(\mathbf{u}) > B\}$, the set of vectors with a large permutation number. We let X denote the set $\{\mathbf{v} \in \mathbb{F}_2^K : \exists \mathbf{u} \in \mathbb{F}_2^K \setminus \mathsf{PN}_B, \exists \mathbf{x}^* \in [\mathsf{bs}]^w, \mathbf{v} = \mathbf{u} \downarrow \mathbf{x}^*\}$. The set X captures exactly the possible spurious solutions: it contains the vectors $\mathbf{v}$ such that there exists some choice of the shift $\mathbf{x}^*$ such that $\mathbf{v} \uparrow \mathbf{x}^*$ has a small permutation number ($\mathsf{pn}(\mathbf{v} \uparrow \mathbf{x}^*) \leq B$). Denoting $\mathsf{Ker}(H) \oplus \mathbf{y}$ the solutions to $H \cdot \mathbf{x} = \mathbf{y}$, if there is a vector $\mathbf{v} \in X \cap \mathsf{Ker}(H) \oplus \mathbf{y}$, then $\mathcal{A}$ can pick $\mathbf{u}, \mathbf{x}^*$ such that $\mathbf{v} = \mathbf{u} \downarrow \mathbf{x}^*$ with $\mathsf{pn}(\mathbf{u}) \leq B$. This guarantees that with probability at least $1/B$, a random permutation π will satisfy $\pi(\mathbf{u}) = \mathbf{u}$, hence $H \cdot (\pi(\mathbf{u}){\downarrow}\mathbf{x}^*) = H \cdot (\mathbf{u}{\downarrow}\mathbf{x}^*) = H \cdot \mathbf{v} = \mathbf{y}$.

Sampling highly-injective instances. Fix some bound B. To eliminate spurious solutions in X, which an adversary could use to win with probability at least $1/B$, we choose parameters (K, k, w) such that when sampling the regular syndrome decoding instance $(H, \mathbf{y} = H \cdot \mathbf{x})$ (for some $\mathbf{x} \in \mathsf{Reg}_w$), it holds with probability $1 - 1/2^\lambda$, the *only* element of X that also belongs to $\mathsf{Ker}(H) \oplus \mathbf{y}$ is the w-regular solution $\mathbf{x}$. It follows from a standard analysis that this is the case as soon as $k \geq \log_2 |X| + \lambda$. To select k, we therefore compute a tight upper bound on $|X|$ (see Lemma 26). Counting the number of elements of X is not entirely straightforward due to the fact that we count "up to some blockwise shift", but a closed formula can be established using known bounds for counting k-necklaces (*i.e.* bitstrings counted up to cyclic shifts) by leveraging Pólya's enumeration theorem [Red27]. Given the formula, we use a short Python program to compute explicitly the bound on $|X|$ and select a suitable

[9] To give a sense of how specific the analysis of [CCJ23] was, not only does it work only for their type of pairs: it works exclusively for $\mathsf{bs} = 6$, corresponding to pairs of bits shared modulo 2 and modulo 3.

parameter k (for a fixed choice of K, w). This also faces some challenges: the formula of Lemma 26 requires summing binomial coefficients over all *integer partitions* of the weight parameter w (*i.e.*, the number of tuples of distinct positive integers that sum to w). Because w is around 120, its number of integer partitions is too large to simply enumerate. With some careful considerations, we observe that many of these partitions can be eliminated from the counting procedure and leverage this observation to reduce the runtime of the program.

Bounding the success probability. We now turn to the crux of the analysis: showing that if $\mathcal{A}$ picks $(\mathbf{u}, \mathbf{x}^*)$ where $\mathsf{pn}(\mathbf{u}) > B$, then their probability of winning the game is at most $O(1/B)$ over the choice of the permutation π. What makes the analysis challenging is that in principle, it could be that some vector $\mathbf{u}$ has a high permutation number, yet *many of its permutations belong to* $\mathsf{Ker}(H) \oplus \mathbf{y}$. The core technical component of the analysis is a proof that with very high probability over the choice of a random syndrome decoding instance $(H, \mathbf{y})$, it will simultaneously hold for *all* vectors $\mathbf{u}$ with $\mathsf{pn}(\mathbf{u}) > B$ that for any choice of shift $\mathbf{x}^*$, $\Pr_\pi[H \cdot (\pi(\mathbf{u}){\downarrow}\mathbf{x}^*) = \mathbf{y}] \leq 4/B$. To state the result formally, we define "good" syndrome decoding instances below:

Definition 6 (GOOD_B). *Given a bound B, GOOD_B is defined as the set of syndrome decoding instances $(H, \mathbf{y}) \in \mathbb{F}_2^{k \times K} \times \mathbb{F}_2^k$ such that for every $\mathbf{u} \in \mathsf{PN}_B \setminus \mathsf{Reg}_w$ and for all $\mathbf{x}^* \in [\mathsf{bs}]^w$,*

$$\Pr_{\pi \leftarrow_r \mathsf{Perm}_w}[H \cdot (\pi(\mathbf{u}){\downarrow}\mathbf{x}^*) = \mathbf{y}] \leq 4/B.$$

Our main technical result of the analysis is stated below:

Lemma 7 (Most syndrome decoding instances are good).

$$\Pr_{H, \mathbf{y}}[(H, \mathbf{y}) \in \mathsf{GOOD}_B] > 1 - \binom{2B}{5} \cdot \frac{2^{K+1}}{B \cdot 2^{3k}} \cdot \left(10 + \frac{(K/w)^w}{2^k}\right).$$

To parse the above, the reader can consider that $(K/w)^w \ll 2^k$ will hold for our selection of parameters, hence the probability that $(H, \mathbf{y}) \in \mathsf{GOOD}_B$ is of the order of $1 - B^4 \cdot 2^{K - 3 \cdot k}$. For concreteness, the reader can think of $\log_2 K$ as being around 1550, $\log_2 k$ as being around 820, w being around 200, and $\log B$ as being around 70, resulting in the above being around $1 - 2^{-630}$.

Key intuition. We outline the main idea of the proof. Given a vector $\mathbf{u}$ with $\mathsf{pn}(\mathbf{u}) = N$, fix some ordering $\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(N)}$ of its distinct blockwise permutations, and let $\mathbf{x}^* \in [\mathsf{bs}]^w$ denote some shift. Sample a random matrix $H \leftarrow_r \mathbb{F}_2^{k \times K}$, a random regular vector $\mathbf{x} \leftarrow_r \mathsf{Reg}_w$, and set $\mathbf{y} \leftarrow H \cdot \mathbf{x}$. Let $(\mathbf{v}_1, \cdots, \mathbf{v}_N) \leftarrow ((\mathbf{u}^{(1)}{\downarrow}\mathbf{x}^*) \oplus \mathbf{x}, \cdots, (\mathbf{u}^{(N)}{\downarrow}\mathbf{x}^*) \oplus \mathbf{x})$ (note that $H \cdot \mathbf{v}_i = 0$ iff $H \cdot (\mathbf{u}^{(i)}{\downarrow}\mathbf{x}^*) = \mathbf{y}$). Observe that the $\mathbf{v}_i$ are random variables, but they are set independently of H (since $\mathbf{x}$ is sampled independently from H). Then, for any subset S of t *linearly independent* vectors $\mathbf{v}_i$, it holds that

$$\Pr_{H \leftarrow_r \mathbb{F}_2^{k \times K}}[H \cdot \mathbf{v}_i = 0 \text{ for all } i \in S] = 2^{-k \cdot t}.$$

In other words, whenever the $\mathbf{v}_i$'s are linearly independent, the binary random variables X_i equal to 1 if $H \cdot \mathbf{v}_i = 0$ are independent. Building upon this observation, we will show the following: fix an arbitrary subset S of five indices. Then

- S contains a size-3 linearly independent subset with probability 1, and
- S contains a size-4 linearly independent subset, except with probability at most $10 \cdot (K/w)^{-w}$.

Together with the previous bound on the probability that $H \cdot \mathbf{v}_i = 0$ for linearly independent vectors, this yields a probability bound of $10 \cdot (K/w)^{-w}/2^{3 \cdot k} + 1/2^{4 \cdot k}$ that $H \cdot \mathbf{v}_i = 0$ for all $i \in S$. To see why this bound holds, observe that:

- The $\mathbf{v}_i$ are pairwise distinct and nonzero by construction (because $\mathbf{u}$ is assumed to be nonregular, so $\pi(\mathbf{u}){\downarrow}\mathbf{x}^*$ is never $\mathbf{0}$, and the $\mathbf{u}^{(i)}$ are distinct by definition).
- If e.g. $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ are linearly dependent, they therefore need to satisfy $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3 = \mathbf{0}$. But then, $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_4 \neq \mathbf{0}$ (otherwise, we would have $\mathbf{v}_3 = \mathbf{v}_4$, contradicting the fact that the vectors are pairwise distinct). Hence, we are guaranteed to find a size-3 independent subset of vectors in S.

– By the same reasoning, S contains necessarily a 4-tuple of $\mathbf{v}_i$'s that does not XOR to 0, say, $(\mathbf{v}_1, \cdots, \mathbf{v}_4)$ (since if both $(\mathbf{v}_1, \cdots, \mathbf{v}_4)$ and $(\mathbf{v}_1, \cdots, \mathbf{v}_3, \mathbf{v}_5)$ XOR to 0, then $\mathbf{v}_4 = \mathbf{v}_5$). Then, either $(\mathbf{v}_1, \cdots, \mathbf{v}_4)$ is linearly independent (in which case we are done, since we found a 4-independent subset), or it must contain a size-3 subset that XORs to 0.
– For any subset of 3 $\mathbf{v}_i$'s, the probability that they XOR to 0 is at most $(K/w)^{-w}$. This follows from the fact that the $\mathbf{v}_i$'s are equal to $(\mathbf{a} \oplus \mathbf{x}, \mathbf{b} \oplus \mathbf{x}, \mathbf{c} \oplus \mathbf{x})$ for some *fixed* vectors $(\mathbf{a}, \mathbf{b}, \mathbf{c})$, and a uniformly random regular vector $\mathbf{x} \in [\mathsf{bs}]^w$. But then, $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3 = 0$ rewrites to $\mathbf{a} \oplus \mathbf{b} \oplus \mathbf{c} = \mathbf{x}$, which happens with probability at most $\mathsf{bs}^{-w} = (K/w)^{-w}$ over the random choice of $\mathbf{x}$.

Since there are 10 size-3 subsets of S, the bound follows. To summarize, we fixed a vector $\mathbf{u}$ with $\mathsf{pn}(\mathbf{u}) = N > B$ and a shift $\mathbf{x}^*$, and showed that for every size-5 subset S of $[N]$, the probability that $H \cdot (\mathbf{u}^{(i)} \downarrow \mathbf{x}^*) = \mathbf{y}$ holds simultaneously for all $i \in S$ is at most $10 \cdot (K/w)^{-w}/2^{3 \cdot k} + 1/2^{4 \cdot k}$.

A careful union bound. To finish the proof of Lemma 7, it remains to compute a union bound over all possible vectors $\mathbf{u}$, shifts $\mathbf{x}^*$, and size-5 subsets S. However, a quick calculation shows that a naive union bound does not suffice: first, the number of subsets is $\binom{N}{5}$, but since we only know that $N > B$ is the permutation number of $\mathbf{u}$, we can only bound it by $w!$, which is way too large. Second, the number of vectors $\mathbf{u}$ is 2^K, which is also too large for the union bound to yield a nontrivial result.

We overcome this issue by providing a more careful union bound. First, we divide the distinct blockwise permutations of $\mathbf{u}$, $(\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(N)})$, into size-$B$ blocks of vectors. We apply the previous bound to all size-5 subsets inside each *block* of vectors, which reduces the factor resulting from the union bound to $(N/B) \cdot \binom{B}{5}$. This suffices to guarantee that in each size-B block, at most 4 vectors $\mathbf{v}_i$ can simultaneously satisfy $H \cdot \mathbf{v}_i = 0$, hence guaranteeing a success probability for $\mathcal{A}$ of at most $4/B$ over the random choice of π. Second, instead of enumerating over all vectors $\mathbf{u}$, we enumerate over all *equivalence classes* of vectors $\mathbf{u}$ which generate the same list $(\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(N)})$. Each equivalence class contains exactly N vectors, and all equivalence classes are disjoint, and we shave a factor N this way from the union bound. Eventually, we finish the union bound by summing over all possible values of $N = \mathsf{pn}(\mathbf{u})$ from $B + 1$ to $w!$. This finishes the proof of Lemma 7.

5 Multi-Instance PPRFs in the Ideal Cipher Model

In this section, we introduce the notion of *multi-instance* puncturable pseudorandom function. We describe an efficient construction from a block cipher, and formally prove its security in the ideal cipher model.

5.1 Defining Multi-Instance Puncturable PRF

Pseudorandom functions [GGM86], are families of keyed functions F_k such that no adversary can distinguish between a black-box access to F_k for a random key k and access to a truly random function. A puncturable pseudorandom function (PPRF) [KPTZ13, BW13, BGI14] is a PRF F such that given an input x, and a PRF key k, one can generate a *punctured* key, denoted $k\{x\} = F.\mathsf{Punc}(K, x)$, which allows evaluating F at every point except for x (*i.e.*, there is an algorithm $F.\mathsf{Eval}$ such that $F.\mathsf{Eval}(k\{x\}, x') = F_K(x')$ for all $x' \neq x$), and such that $F_k(x)$ is indistinguishable from random given $k\{x\}$. Then,

Definition 8 ((N, τ)-instance (t, ϵ)-secure PPRF). *A function family $F = \{F_K\}$ with input domain $[2^D]$, salt domain $\{0,1\}^s$, and output domain $\{0,1\}^\lambda$, is an (N, τ)-instance (t, ϵ)-secure PPRF if it is a PPRF which additionally takes as input a salt* salt, *and for every non-uniform PPT distinguisher $\mathcal{D}$ running in time at most t, it holds that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{PPRF}}(\mathcal{D}) = |\Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}pprf}}(\lambda) = 1] - \Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}pprf}}(\lambda) = 1]| \leq \epsilon(\lambda)$$

where the experiments $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}pprf}}(\lambda)$ and $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}pprf}}(\lambda)$ are defined below.

$\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}pprf}}(\lambda)$:	$\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}pprf}}(\lambda)$:
– $((K_{j,e})_{j\leq N, e\leq\tau} \leftarrow_r (\{0,1\}^\lambda)^{N\cdot\tau}$ – $\mathsf{salt} := (\mathsf{salt}_1, \dots, \mathsf{salt}_N) \leftarrow_r \{0,1\}^s$ – $\mathbf{i} := ((i_{1,e})_{e\leq\tau}, \dots, (i_{1,e})_{e\leq\tau}) \leftarrow_r [2^D]^{N\cdot\tau}$ – $\forall j \leq N, e \leq \tau : K_{j,e}^{i_{j,e}} \leftarrow F.\mathsf{Punc}(K_{j,e}, i_{j,e})$ – $(y_{j,e})_{j\leq N, e\leq\tau} \leftarrow (F_{K_{j,e}}(i_{j,e}, \mathsf{salt}_i))_{j\leq N, e\leq\tau}$ **Output** $b \leftarrow \mathcal{D}(\mathsf{salt}, \mathbf{i}, (K_{j,e}^{i_{j,e}}, y_{j,e})_{j\leq N, e\leq\tau})$	– $((K_{j,e})_{j\leq N, e\leq\tau} \leftarrow_r (\{0,1\}^\lambda)^{N\cdot\tau}$ – $\mathsf{salt} := (\mathsf{salt}_1, \dots, \mathsf{salt}_N) \leftarrow_r \{0,1\}^s$ – $\mathbf{i} := ((i_{1,e})_{e\leq\tau}, \dots, (i_{1,e})_{e\leq\tau}) \leftarrow_r [2^D]^{N\cdot\tau}$ – $\forall j \leq N, e \leq \tau : K_{j,e}^{i_{j,e}} \leftarrow F.\mathsf{Punc}(K_{j,e}, i_{j,e})$ – $(y_{j,e})_{j\leq N, e\leq\tau} \leftarrow_r (\{0,1\}^\lambda)^{N\cdot\tau}$ **Output** $b \leftarrow \mathcal{D}(\mathsf{salt}, \mathbf{i}, (K_{j,e}^{i_{j,e}}, y_{j,e})_{j\leq N, e\leq\tau})$

The motivation for adding the parameter τ in Definition 8 stems from our use of PPRFs in signatures: our signature construction uses τ parallel instances of the PPRF using the same salt, while distinct salts are used across distinct signature queries.

Furthermore, we observe our actual construction satisfies a stronger property, in which indistinguishability is preserved even the ideal world experiment does not only sample $(y_1, \cdots, y_N)$ uniformly at random, but also samples "fake" punctured keys $K_j^{x_k}$ uniformly at random over an appropriate domain. This stronger notion is not strictly necessary in our signature construction, but its use simplifies the analysis. Below, we state the definition explicitly for the punctured key domain that corresponds to our (GGM-based) construction, but the notion extends naturally to arbitrary domains.

Definition 9 ((N, τ)-instance strongly (t, ϵ)-secure PPRF). *A function family $F = \{F_K\}$ with input domain $[2^D]$, salt domain $\{0,1\}^s$, output domain $\{0,1\}^\lambda$, and punctured key domain $(\{0,1\}^\lambda)^D$ is an (N, τ)-instance (t, ϵ)-secure PPRF if it is a PPRF which additionally takes as input a salt* salt*, and for every non-uniform PPT distinguisher $\mathcal{D}$ running in time at most t, it holds that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{PPRF}}(\mathcal{D}) = |\Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}pprf}}(\lambda) = 1] - \Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}spprf}}(\lambda) = 1]| \leq \epsilon(\lambda),$$

where the experiment $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}spprf}}(\lambda)$ is defined as $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}pprf}}(\lambda)$, except that the line $\forall j \leq N, e \leq \tau :$ $K_{j,e}^{i_{j,e}} \leftarrow F.\mathsf{Punc}(K_{j,e}, i_{j,e})$ is replaced by $\forall j \leq N, e \leq \tau : K_{j,e}^{i_{j,e}} \leftarrow_r (\{0,1\}^\lambda)^D$.

5.2 Constructing Multi-Instance Puncturable PRFs

In this section, we introduce the notion of (N, τ)-instance (t, ε)-secure pseudorandom generator, which extends the notion of pseudorandom generators to the multi-instance setting (with salt) analogously to our definition of multi-instance PPRFs. Then, we show that the standard GGM construction extends immediately to the multi-instance setting: (length-doubling) (N, τ)-instance (t, ε)-secure PRGs imply (N, τ)-instance strongly $(t, D \cdot \varepsilon)$-secure PPRFs with input domain $[2^D]$ and punctured key domain $(\{0,1\}^\lambda)^D$. We start by defining (N, τ)-instance (t, ϵ)-secure length-doubling PRGs. Below, to interface more easily with the tree-based GGM construction of PPRFs, we use $(\mathsf{F}_0, \mathsf{F}_1)$ to denote functions that compute the left half and right half of the length-doubling PRG output.

Definition 10 ((N, τ)-instance (t, ϵ)-secure PRG). *A PRG $\mathsf{PRG} = (\mathsf{F}_0, \mathsf{F}_1)$ with $\mathsf{F}_b : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ is an (N, τ)-instance (t, ϵ)-secure length-doubling* PRG *if for every non-uniform PPT distinguisher $\mathcal{D}$ running in time at most t, it holds that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{PRG}}(\mathcal{D}) = |\Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}prg}}(\lambda) = 1] - \Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}prg}}(\lambda) = 1]| \leq \epsilon(\lambda),$$

where $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}prg}}(\lambda)$ and $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}prg}}(\lambda)$ are defined below.

$\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}prg}}(\lambda)$:	$\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}prg}}(\lambda)$:
– $(\mathsf{salt}_1, \mathsf{salt}_2, \dots, \mathsf{salt}_{2N}) \leftarrow_r \{0,1\}^\lambda$ – $(\mathsf{seed}_{i,e})_{i\leq N, e\leq\tau} \leftarrow_r (\{0,1\}^\lambda)^{N\cdot\tau}$ – $\forall i \leq N, e \leq \tau$: • $y_{2i-1,e} \leftarrow \mathsf{F}_0(\mathsf{seed}_{i,e}, \mathsf{salt}_{2i-1})$ • $y_{2i,e} \leftarrow \mathsf{F}_1(\mathsf{seed}_{i,e}, \mathsf{salt}_{2i})$ **Output** $b \leftarrow \mathcal{D}((\mathsf{salt}_i, (y_{i,e})_{e\leq\tau})_{i\leq 2N})$	– $(\mathsf{salt}_1, \mathsf{salt}_2, \dots, \mathsf{salt}_{2N}) \leftarrow_r \{0,1\}^\lambda$ – $(y_{i,e})_{i\leq 2N, e\leq\tau} \leftarrow_r (\{0,1\}^\lambda)^{2N\cdot\tau}$ **Output** $b \leftarrow \mathcal{D}((\mathsf{salt}_i, (y_{i,e})_{e\leq\tau})_{i\leq 2N})$

We note that the definition extends immediately to PRGs that stretch their seeds by a larger factor. We also remark that in the definition above, we assumed that each of F_0 and F_1 takes a distinct λ-bit salt. The definition can be extended to more general salting procedures, but we defined multi-instance PRG with respect to the way we use salt in our actual construction for notational convenience. Looking ahead, the fact that each F_b takes only λ bits of salt is actually a crucial byproduct of our use of block ciphers, and the main reason why the security analysis becomes highly non-trivial.

Now, given a seed $\mathsf{seed} \leftarrow_r \{0,1\}^\lambda$, salt $\mathsf{salt} := (\mathsf{salt}_0, \mathsf{salt}_1) \leftarrow_r \{0,1\}^{2\lambda}$, and a multi-instance secure PRG $\mathsf{F}_0, \mathsf{F}_1 : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$, we recursively define a PPRF $\mathsf{PPRF}(\mathsf{seed}, \mathsf{salt}) = \mathsf{PPRF}(\mathsf{seed}, \mathsf{salt}, 2^D)$ over input domain $\{0,1\}^D$ (which we later identify with $[2^D]$) in a tree-based fashion as follows:

- The first layer includes two nodes $X_0 := \mathsf{F}_0(\mathsf{seed}, \mathsf{salt}_0),\ X_1 := \mathsf{F}_1(\mathsf{seed}, \mathsf{salt}_1)$.
- Each layer of the tree is constructed from the nodes of the previous layer similarly, as follows:

$$\begin{aligned}\mathsf{PPRF}_{\mathsf{seed}}(\mathsf{salt}, i) &= \mathsf{F}_{i_D}\left(\mathsf{PPRF}_{\mathsf{seed}}\left(\mathsf{salt}, i_1, \ldots, i_{D-1}\right), \mathsf{salt}\right) \\ &= \mathsf{F}_{i_D}\left(\mathsf{F}_{i_{D-1}}\left(\ldots\left(\mathsf{F}_{i_1}(\mathsf{seed}, \mathsf{salt}), \mathsf{salt}\right), \mathsf{salt}\right)\right),\end{aligned}$$

  where $i_1, \cdots, i_D$ denote the bits of i.

As with the standard GGM construction, a punctured key at i is just the co-path to i in the tree, *i.e.*, the set of intermediate nodes that can be used to recover all leaves except the $i-$th one: $\mathsf{CoPath}_{\mathsf{seed}}(\mathsf{salt}, i) = \mathsf{PPRF}_{\mathsf{seed}}\left(\mathsf{salt}, i_{1,\ldots,\bar{j}}\right)_{j=1,\ldots,D}$. The formal construction is presented in Figure 1 and the proof of security is shown in Theorem 11. We note that the proof is a natural extension of the security analysis of the GGM construction [GGM86].

Theorem 11 (PPRF security). *Assume that* $\mathsf{PRG} = (\mathsf{F}_0, \mathsf{F}_1)$ *with* $\mathsf{F}_b : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ *is an* (N,τ)*-instance* (t,ϵ)*-secure length-doubling PRG. Then the construction* $\mathsf{PPRF}(\mathsf{seed}, \mathsf{salt}, 2^D)$ *described in Figure 1 is an* (N,τ)*-instance strongly* $(t, D\cdot\epsilon)$*-secure PPRF with input domain* $[2^D]$ *and punctured key domain* $(\{0,1\}^\lambda)^D$*.*

Proof. We proceed in a sequence of hybrids where each hybrid relies on the (N,τ)-instance security of $\mathsf{F}_0, \mathsf{F}_1$.

First, recall that for each leaf $i^{(j,e)} \in \{0,1\}^D$ in each tree $\mathsf{PPRF}(\mathsf{seed}_{j,e}, \mathsf{salt}_j, 2^D)$, the value assigned to this leaf $i^{(j,e)}$ is denoted $X_{i_1^{(j,e)},\ldots,i_D^{(j,e)}}$. The secret path from the root $(\mathsf{seed}_{j,e}, \mathsf{salt}_j)$ to the leave $i^{(j,e)}$ is the tuple of intermediate nodes $\{X_{i_1^{(j,e)}}, X_{i_1^{(j,e)},i_2^{(j,e)}}, \ldots, X_{i_1^{(j,e)},\ldots,i_D^{(j,e)}}\}$.

- **Experiment 0** ($\mathsf{Exp}^{\mathbf{0}}$). All trees of the N instances are obtained through the actual scheme described in Figure 1, which is run at each level to generate the leaves of the next level. More in detail: for each $j \le N, e \le \tau$ the construction of the (j,e)-th tree is carried out starting from a random master $(\mathsf{seed}_{j,e}, \mathsf{salt}_j)$ and using, for all 2^D levels, F_0 and F_1 to generate the right child and the left child.

  Experiment 1 ($\mathsf{Exp}^{\mathbf{1}}$). Same as the previous experiment, the only difference is at the first level of each tree. For all $j = 1, \ldots, N, e \le \tau$, the leaves at the first level $(X_{1^{(j,e)}}, X_{0^{(j,e)}})$ are not generated with $\mathsf{F}_0, \mathsf{F}_1$, but are instead randomly sampled. Since $\mathsf{F}_0, \mathsf{F}_1$ is an (N,τ)-instance (t,ϵ)-secure PRG, then
$$|\Pr[\mathsf{Exp}^0(\lambda) = 1] - \Pr[\mathsf{Exp}^1(\lambda) = 1]| \le \epsilon(\lambda)$$

  Experiment 2 ($\mathsf{Exp}^{\mathbf{2}}$). The difference with the previous experiment is in the second level of each tree: all the leaves $(X_{i_1^{(j,e)},0}, X_{i_1^{(j,e)},1})$ previously computed by using F_0 and F_1 are now randomly chosen for each $j = 1, \ldots, N, e \le \tau$. As before, using the secure property of F_0 and F_1, we obtain

$$|\Pr[\mathsf{Exp}^1(\lambda) = 1] - \Pr[\mathsf{Exp}^2(\lambda) = 1]| \le \epsilon(\lambda)$$

  As it is easy to guess, traversing along the secret path of each tree, this mechanism of replacing the two leaves $(X_{i_1^{(j,e)},\ldots,i_{k-1}^{(j,e)},0}, X_{i_1^{(j,e)},\ldots,i_{k-1}^{(j,e)},1})$ at each level $k \in [1,D]$ by uniformly random values can continue for the whole depth D of the tree. This way, we have D experiments and applying the same hypothesis about the security of the F_0 and F_1 used, we will get:

$$|\Pr[\mathsf{Exp}^{i-1}(\lambda) = 1] - \Pr[\mathsf{Exp}^i(\lambda) = 1]| \le \epsilon(\lambda)$$

PARAMETERS:

- Two functions $\mathsf{F}_0, \mathsf{F}_1 : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Number of leaves $n = 2^D \in \mathbb{N}$, computational security parameter λ.

CONSTRUCTION:

- Sample $(\mathsf{seed}, \mathsf{salt}) \leftarrow_r \{0,1\}^{3\lambda}$ where $\mathsf{salt} := (\mathsf{salt}_0, \mathsf{salt}_1)$. We use $\mathsf{salt}_0, \mathsf{salt}_1$ for $\mathsf{F}_0, \mathsf{F}_1$ respectively. For simplicity, we sometimes write $\mathsf{F}_i(\mathsf{seed}, \mathsf{salt}_i)$ as $\mathsf{F}_i(\mathsf{seed}, \mathsf{salt})$ for $i \in \{0,1\}$.
- Let $X_0 := \mathsf{F}_0(\mathsf{seed}, \mathsf{salt}_0)$, $X_1 := \mathsf{F}_1(\mathsf{seed}, \mathsf{salt}_1)$.
- For $i \in [2, D]$, define $X_{b_1,\ldots,b_{i-1},0} = \mathsf{F}_0(\mathsf{F}_{b_{i-1}}(X_{b_1,\ldots,b_{i-1}}), \mathsf{salt}_0)$, $X_{b_1,\ldots,b_{i-1},1} = \mathsf{F}_1(\mathsf{F}_{b_{i-1}}(X_{b_1,\ldots,b_{i-1}}), \mathsf{salt}_1)$ where $b_j \in \{0,1\}$ for all $j \in [1, i-1]$.
- We generalize the formula to compute the leaf of the tree as follows:
  For each $i \in [0, n-1]$, bit-decompose i as $\sum_{j=1}^{D} 2^{j-1} \cdot i_j$ for $i_j \in \{0,1\}$ then:

$$\begin{aligned} X_i = X_{i_1,\ldots,i_D} &= \mathsf{F}_{i_D}(\mathsf{F}_{i_{D-1}}(X_{i_1,\ldots,i_{D-1}}), \mathsf{salt}_{i_D}) \\ &= \mathsf{F}_{i_D}(\mathsf{F}_{i_{D-1}}(\ldots(\mathsf{F}_{i_1}(\mathsf{seed}_{i_1}, \mathsf{salt}_{i_1}), \mathsf{salt}_{i_{D-1}}), \mathsf{salt}_{i_D}) \end{aligned}$$

  To formalize, the value for each leaf $i \in [0, n-1]$ is denoted as:

$$\begin{aligned} \mathsf{PPRF}_{\mathsf{seed}}(\mathsf{salt}, i) &= \mathsf{F}_{i_D}\left(\mathsf{PPRF}_{\mathsf{seed}}\left(\mathsf{salt}, i_{1,\ldots,D-1}\right), \mathsf{salt}\right) \\ &= \mathsf{F}_{i_D}\left(\mathsf{F}_{i_{D-1}}\left(\ldots\left(\mathsf{F}_{i_1}(\mathsf{seed}, \mathsf{salt}\right), \mathsf{salt}\right), \mathsf{salt}\right) \end{aligned}$$

  where $i_{1,\ldots,k} = \sum_{j=1}^{k} 2^{k-j} i_j$ for any $k \in [1, D]$.
- We define the co-path $\mathsf{CoPath}(i)$ for each $i = \sum_{j=1}^{D} 2^{j-1} \cdot i_j \in [0, n-1]$ as follows:

$$\mathsf{CoPath}(i) = \mathsf{CoPath}(X_{i_1,\ldots,i_D}) = \{X_{\bar{i}_1}, X_{i_1,\bar{i}_2}, \ldots, X_{i_1,\ldots,\bar{i}_D}\}$$

  Formalizing, we have:

$$\mathsf{CoPath}_{\mathsf{seed}}(\mathsf{salt}, i) = \mathsf{PPRF}_{\mathsf{seed}}\left(\mathsf{salt}, i_{1,\ldots,\bar{j}}\right)_{j=1,\ldots,D}$$

  where $i_{1,\ldots,\bar{k}} = \sum_{j=1}^{k-1} 2^{k-j}.i_j + \bar{i}_k$ for any $k \in [1, D]$.

Fig. 1. New construction $\mathsf{PPRF}(\mathsf{seed}, \mathsf{salt}, 2^D)$ of Puncturable PRF

for all $i = 1, \ldots, D$. Furthermore, when we traverse the path this way, we simultaneously replace all values on the *co-path* to the leaves $i^{(j,e)}$ by uniformly random values.

Experiment D ($\mathsf{Exp}^{\mathbf{D}}$). In the last experiment, all nodes on the co-path to $i^{(j,e)}$ as well as the leaf $i^{(j,e)}$ are picked uniformly at random, for $j = 1$ to N and $e = 1$ to τ. We obtain the final bound

$$|\Pr[\mathsf{Exp}^0(\lambda) = 1] - \Pr[\mathsf{Exp}^D(\lambda) = 1]| \le D \cdot \epsilon(\lambda),$$

which concludes the proof.

5.3 A Multi-Instance PRG in the Ideal Cipher Model

In this section, we describe the construction of multi-instance PRG in the ideal cipher model. Our construction itself is not really new, but is a tweak on a construction of [GKWY20]. The work of [GKWY20] gives a construction of PPRF in the random permutation model, which is obtained by applying the GGM reduction to the following "Davies-Meyer" construction of a length-doubling PRG:

$$G : x \to (\pi_0(x) \oplus x, \pi_1(x) \oplus x),$$

where (π_0, π_1) are pseudorandom permutations. The PRG is proven secure in the random permutation model (in the analysis, all parties are given oracle access to π_0, π_1, and their inverses). Our core observation, which is quite simple in hindsight, is that the most efficient instantiation of this construction implements the permutations π_0, π_1 by fixing two keys (K_0, K_1) and defining $\pi_b := E_{K_B}$, where E_{K_B} is a *block cipher* (such as AES). This suggests the following idea: instead of fixing the keys (K_0, K_1), sample them randomly and use them as a salt for the PRG in the multi-instance setting.

The candidate multi-instance PRG becomes $G = (\mathsf{F}_0, \mathsf{F}_1) : (x, \mathsf{salt}) \to (E_{\mathsf{salt}_0}(x) \oplus x, E_{\mathsf{salt}_1}(x) \oplus x)$. The formal construction is given in Figure 2. While the high-level intuition is straightforward, the formal analysis turns out to be considerably more involved. The remainder of this section is devoted to a formal proof that the above construction is an (N, τ)-instance (t, ε)-secure PRG, for parameters $(N, \tau, t, \varepsilon)$ which will be specified later. The proof is in the *ideal cipher model*: in this model, each key $K \in \{0,1\}^\lambda$ defines an independent uniformly random permutation π_K. All parties are given access to an oracle which, on input $(0, K, x)$, outputs $\pi_K(x)$, and on input $(1, K, y)$, outputs $\pi_K^{-1}(y)$. And the proof security is shown in Theorem 13, it relies on a careful analysis using Patarin's H-coefficient technique [Pat09, CS14] and forms one of the core technical contributions of this work.

Definition 12 (Ideal Cipher Oracle). *For every $K \in \{0,1\}^\lambda$, let $\pi_K : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a uniformly random permutation over $\{0,1\}^\lambda$. The ideal cipher oracle $\mathcal{O}_\pi$ is defined as follows:*

- *On input $(x, K) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$, outputs $\pi_K(x)$.*
- *On input (inv, x, K), outputs $\pi_K^{-1}(x)$.*

PARAMETERS:

– For each $K \in \{0,1\}^\lambda$, $\pi_K : \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a uniformly random permutation.

CONSTRUCTION:

– Sample $\mathsf{salt} \leftarrow_r \{0,1\}^{2\lambda}$. parse $\mathsf{salt} := (K_0, K_1)$.
– $\mathsf{F}_b : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ is defined as $\mathsf{F}_b(\mathsf{seed}, \mathsf{salt}_b) = \pi_{K_b}(\mathsf{seed}) \oplus \mathsf{seed}$ for $b \in \{0,1\}$ and $\mathsf{seed} \in \{0,1\}^\lambda$.

Fig. 2. Multi-instance PRG $\mathsf{F}_0, \mathsf{F}_1$ in the ideal cipher model

Theorem 13. *Let $\mathsf{F}_0, \mathsf{F}_1$ be the functions defined in Figure 2. Let q be the number of queries to the oracle $\mathcal{O}_\pi$. Then $(\mathsf{F}_0, \mathsf{F}_1)$ is an (N, τ)-instance (q, ϵ)-secure* PRG *in the ideal cipher model (where the parties are given oracle access to $\mathcal{O}_\pi$ from Definition 12), where*

$$\varepsilon \leq f_N(\lambda) \cdot q \cdot \left(\frac{1}{2^{\lambda-1}} + \frac{1}{2^\lambda - q} \right) + \frac{4\tau N}{2^{2\lambda}},$$

for some function f_N such that if $N \leq 2^{\lambda-1}$, $f_N(\lambda) \leq \frac{3\tau\lambda \cdot \ln 2}{\ln \lambda + \ln \ln 2}$, and if $N \leq 2^{\lambda/2}$, $f_N(\lambda) \leq 4\tau$.

Proof. Fix a number of instances N and a number of repetitions τ. We consider a distinguisher $\mathcal{D}$ that receives $(\mathsf{salt}_i, (y_{i,e})_{e \leq \tau})_{i \leq 2N}$ according to either the real world experiment $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}prg}}$ or the ideal world experiment $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}prg}}$ of Definition 10, interacts with the ideal cipher oracle $\mathcal{O}_\pi$, and outputs a guess b. Let q be a bound on the number of queries of $\mathcal{D}$ to $\mathcal{O}_\pi$. To simplify the discussion, we assume that the $N \cdot \tau$ seeds $(\mathsf{seed}^{(1,e)}, \cdots, \mathsf{seed}^{(N,e)})_{e \leq \tau}$ are also sampled (but not used) in the experiment $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw\text{-}prg}}$. We also write salt_i as (K_0^i, K_1^i).

Reformulating the experiment. Now, sample $(\mathsf{seed}^{(1,e)}, \cdots, \mathsf{seed}^{(N,e)})_{e \leq \tau}$ and pairs of keys $(K_0^i, K_1^i)_{i \leq N}$. If $N \cdot \tau$ is large, with high probability there will be some collisions among the seeds. Let $M \leq N \cdot \tau$ denote the number of distinct seeds. To simplify the analysis, we reorder and rename the seeds and the keys as follows:

– $\mathsf{seed}_1, \ldots, \mathsf{seed}_M$ are the M distinct seeds from the set of $N \cdot \tau$ sampled seeds $\mathsf{seed}^{(j,e)}$. For each seed seed_i, define $S_i \subseteq \{0,1\}^\lambda$ to be the set of indices such that $K \in S_i$ if there is an index (j, e) such that $\mathsf{seed}^{(j,e)} = \mathsf{seed}_i$ and either $K = K_0^j$ or $K = K_1^j$ (that is, seed_i was sampled at least once together with a salt that contains K). Note that S_i corresponds to all keys K such that π_K is queries on seed_i in $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{rw\text{-}prg}}$.
– For each π_K, define $S'_K := \{i : K \in S_i\} \subseteq [M]$ to be the set of indices of seeds that will be queried to π_K.

With the above notations, the distinguisher $\mathcal{D}$ receives the sets $S_1, \cdots, S_M$, and for each $i \leq M$, it gets either $\pi_K(\mathsf{seed}_i) \oplus \mathsf{seed}_i$ for all $K \in S_i$ (experiment $\mathsf{Exp}_{\mathcal{D}}^{\text{rw-prg}}$), or a set of random values $(y_{K,i})_{K \in S_i}$ (experiment $\mathsf{Exp}_{\mathcal{D}}^{\text{iw}}$). These alternative experiments only differ from the original experiments if it happens that two seeds $\mathsf{seed}^{(i,e)}, \mathsf{seed}^{(j,f)}$ collide, and two of their keys (K_0^i, K_1^i) and (K_0^j, K_1^j) also collide: in this case, the original experiments would return distinct values y in the ideal world, but identical values in the real world, making them trivially distinguishable. However, the probability of this even happening is very small:

$$\Pr[\exists (i,e) \neq (j,f), \mathsf{seed}^{(i,e)} = \mathsf{seed}^{(j,f)} \wedge \exists (b_i, b_j) \in \{0,1\}^2, K_{b_i}^i = K_{b_j}^j] \leq \frac{4N \cdot \tau}{2^{2\lambda}}.$$

Condition on this even not happening, the new experiments become perfectly equivalent to the original experiments. We therefore raise a flag if the above condition occurs, abort if a flag is raised, and focus on bounding the distinguishing advantage in these new experiments.

Bounding the size of S'_K. We start by bounding the maximum size of S'_K for any K. We will need a standard lemma on the maximum load of a bin when tossing m balls into n bins:

Lemma 14 (balls-and-bins). *Consider tossing m balls into n bins. For $m \leq n$, denoting* $\mathsf{max\_load}$ *as the maximum number of balls that end up in any single bin, we have*

$$\Pr\left[\mathsf{max\_load} \geq \frac{3 \ln n}{\ln \ln n}\right] \leq \frac{1}{n}.$$

By definition, the maximum size of S'_K is reached for the permutation π_K that is invoked on the largest number of distinct seeds. A tight upper bound on this number follows from a simple balls-and-bins analysis: each time τ new seeds $(\mathsf{seed}^{(i,e)})_{e \leq \tau}$ are sampled, two keys (K_0^i, K_1^i) are sampled, which we view as throwing two balls to two random bins, sampled randomly from 2^λ possible bins. After N steps of this experiment (hence after throwing $2N$ balls at random), denoting $\mathsf{max\_load}$ the maximum load of any bin, $\tau \cdot \mathsf{max\_load}$ is an upper bound on $\max_K |S'_K|$.[10] We get:

Claim. Whenever $2N \leq 2^\lambda$, the maximum load $\max_K |S'_K|$ is bounded by $\frac{3\tau \cdot \ln 2^\lambda}{\ln \ln 2^\lambda}$ with probability $1 - 2^{-\lambda}$. Furthermore, if $2N \leq 2^{\lambda/2}$, $\max_K |S'_K|$ is bounded by 4τ with probability $1 - 2^{-\lambda}$.

The first part of the claim follows directly from the balls-and-bins lemma 14. The last part of the claim follows from the fact that when $2N \leq 2^{\lambda/2}$, the probability of having 4 balls in any given bin is at most $1/2^{2\lambda}$, and the claim follows by a union bound over the 2^λ bins.

Bounding the advantage of $\mathcal{D}$. We now move to the crux of the analysis, where we bound the advantage of $\mathcal{D}$ in distinguishing the real world and the ideal world experiments. We formally define below the *transcript* of the interaction of $\mathcal{D}$ in the experiments:

Definition 15 (Transcript). *We define a transcript of $\mathcal{D}$'s interaction by*

$$Q = ((y_{i,j})_{i \leq M, j \in S_i}, Q_\pi, (\mathsf{seed}_i)_{i \leq M})$$

where $Q_\pi = (z, j, \pi_j(z))$ records all $\mathcal{D}$'s queries/answers to/from the permutation oracle $\mathcal{O}_\pi$ (the queries for the inverse of permutation can be considered as $(\pi_b^{-1}(z), b, z)$. Note that here, $(\mathsf{seed}_i)_{i \leq M}$ is included to facilitate the analysis but the distinguisher $\mathcal{D}$ does not get them: in the real-world, $(\mathsf{seed}_i)_{i \leq M}$ are used to compute $(y_{i,j})_{i \leq M, j \in S_i}$ otherwise in the ideal-world, $(y_{i,j})_{i \leq M, j \in S_i}$ are sampled uniformly random from $\{0,1\}^\lambda$.

We say that a transcript Q is ***attainable*** *for some fixed $\mathcal{D}$ if there exist some oracles $\mathcal{O}_\pi$ such that the interaction of $\mathcal{D}$ with those oracles would lead to transcript Q.*

[10] It is a very tight upper bound: because $|S'_K|$ counts only *distinct* seeds, we are overcounting whenever it happens that a new seed $\mathsf{seed}^{(j,e)}$ is sampled that collides with one of the previous seeds ($\mathsf{seed}^{(jne)} = \mathsf{seed}^{(i,f)}$ for some $i < j$) and K_0^j or K_1^j also collides with one of the two keys (K_0^i, K_1^i). But the chance that this happens is at most $\tau \cdot N / 2^{2\lambda}$.

In the game of distinguishing between the ideal world and the real world, we have

$$\mathsf{Adv}(\mathcal{D}_\pi^{\mathcal{O}}) = |\mathsf{Pr}_{\mathsf{rw}}[\mathcal{D}^{\mathcal{O}_\pi} = 1] - \mathsf{Pr}_{\mathsf{iw}}[\mathcal{D}^{\mathcal{O}_\pi} = 1]|$$

Our proof will crucially rely on Patarin's H-coefficient technique [Pat09, CS14], which we recall below. The H-coefficient theorem allows to bound the advantage of distinguisher by classifying the set of attainable transcripts into "good" and "bad" transcripts:

Theorem 16 (H-coefficient). *Fix some distinguisher $\mathcal{D}$. Let $\mathcal{T}$ denote the set of attainable transcripts Q and $\mathsf{Pr}_{\mathsf{rw}}$ and $\mathsf{Pr}_{\mathsf{iw}}$ denote the probabilities of events in the real and ideal world, respectively. Let $\mathcal{T}_{\mathsf{bad}}$ denote a set of "bad" transcripts, and $\mathcal{T}_{\mathsf{good}} = \mathcal{T} \setminus \mathcal{T}_{\mathsf{bad}}$ be the set of "good" transcripts, suppose that:*

- $\mathsf{Pr}_{\mathsf{iw}}[Q \in \mathcal{T}_{\mathsf{bad}}] \leq \nu$.
- $\left|\frac{\mathsf{Pr}_{\mathsf{rw}}[Q]}{\mathsf{Pr}_{\mathsf{iw}}[Q]} - 1\right| \leq \mu$ *for all* $Q \in \mathcal{T}_{\mathsf{good}}$.

Then $\mathsf{Adv}(\mathcal{D}_\pi^{\mathcal{O}}) \leq \nu + \mu$.

One key insight of the H-coefficient technique is that the ratio $\frac{\mathsf{Pr}_{\mathsf{rw}}[Q]}{\mathsf{Pr}_{\mathsf{iw}}[Q]}$ is equal to the ratio between the probability that the real-world oracles are consistent with Q and the probability that the ideal-world oracles are consistent with Q. We denote Pr[RW is consistent with Q] and Pr[IW is consistent with Q] as $\mathsf{Pr}_{\mathsf{rw}}(Q)$ and $\mathsf{Pr}_{\mathsf{iw}}(Q)$ respectively. Then

$$\forall Q \in \mathcal{T}_{\mathsf{good}}, \ \frac{\mathsf{Pr}_{\mathsf{rw}}[Q]}{\mathsf{Pr}_{\mathsf{iw}}[Q]} = \frac{\mathsf{Pr}_{\mathsf{rw}}(Q)}{\mathsf{Pr}_{\mathsf{iw}}(Q)}$$

Our goal now is to use the H-coefficient theorem to prove Theorem 13. Define $\mathcal{T}_{\mathsf{bad}}$ and $\mathcal{T}_{\mathsf{good}}$ from the sets of distinct seeds and permutations.

- $\mathcal{T}_{\mathsf{bad}}$ contains transcripts $Q = ((y_{i,K})_{i \leq M, K \in S_i}, Q_\pi, (\mathsf{seed}_i)_{i \leq M}) \in \mathcal{T}$ s.t.
  - $\exists(\mathsf{seed}_i, K, *) \in Q_\pi$ with $K \in S_i$.
  - $\exists(*, K, \mathsf{seed}_i \oplus y_{i,K})$ with $K \in S_i$.
- $\mathcal{T}_{\mathsf{good}} = \mathcal{T} \setminus \mathcal{T}_{\mathsf{bad}}$.

Bounding $\mathsf{Pr}_{\mathsf{iw}}[Q \in \mathcal{T}_{\mathsf{bad}}]$. Let denote $|Q_\pi| = q = \sum_{K \in \{0,1\}^\lambda}^{L} q_K$ where $q_K := |Q_{\pi_K}| := |\{(*, K, *) \in Q_\pi\}|$ for $K \in \{0,1\}$. In the ideal-world, $(\mathsf{seed}_i)_{i \leq M}$ are independent of $((y_{i,K})_{i \leq M, K \in S_i}$, and we have:

$$\begin{aligned}
\mathsf{Pr}_{\mathsf{iw}}[Q \in \mathcal{T}_{\mathsf{bad}}] \leq & \sum_{K \in \{0,1\}^\lambda} (\mathsf{Pr}_{\mathsf{iw}}[\exists(\mathsf{seed}_i, K, *) \in Q_\pi | i \in S'_K] \\
& \quad + \mathsf{Pr}_{\mathsf{iw}}[\exists(*, K, y_{i,K} \oplus \mathsf{seed}_i) \in Q_\pi | i \in S'_K]) \\
= & \sum_{K \in \{0,1\}^\lambda} \frac{2q_K \cdot |S'_K|}{|2^\lambda|} = \frac{1}{2^{\lambda-1}} \cdot \sum_{K \in \{0,1\}^\lambda} q_K \cdot |S'_K| \\
\leq & \frac{1}{2^{\lambda-1}} \cdot q \cdot \max_K |S'_K|.
\end{aligned}$$

Bounding $\mathsf{Pr}_{\mathsf{rw}}[Q]/\mathsf{Pr}_{\mathsf{iw}}[Q]$ *for* $Q \in \mathcal{T}_{\mathsf{good}}$. First, we compute the probability $\mathsf{Pr}_{\mathsf{iw}}(Q)$ that the ideal-world oracle is consistent with Q. Denote $((\mathsf{seed}'_i)_{i \leq M}, (y'_{i,K})_{i \leq M, K \in S_i}, (\pi_K)_{K \in \{0,1\}^\lambda})$ some arbitrary setting of the ideal world experiment, where $(\mathsf{seed}'_i)_{i \leq M}, (y_{i,K})_{i \leq M, K \in S_i}$ are sampled as in $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{iw}}(\lambda)$, and $\pi_K : \{0,1\}^\lambda \to \{0,1\}^\lambda$ are fixed random permutations. Let $\pi \vdash Q_\pi$ denote the event that permutation π is consistent with the queries/answers in Q_π. Let us write $(\pi_K)_K \vdash Q_\pi$ to indicate that random permutations π_k are consistent with all queries in the transcript Q_π. Since in the ideal world all these values are sampled independently, denoting $p_\pi = \Pr_{\pi_K}[(\pi_K)_{K \in \{0,1\}^\lambda} \vdash Q_\pi)]$, we have:

$$\begin{aligned}
&\mathsf{Pr}_{\mathsf{iw}}(Q) = \\
&\Pr_{\mathsf{seed}'_i, y'_{i,j}, \pi_j}[(\mathsf{seed}'_i = \mathsf{seed}_i) \ \wedge \ (\forall K \in S_i,\ y_{i,K} = y'_{i,K}) \wedge \ ((\pi_K)_{K\in\{0,1\}^\lambda} \vdash Q_\pi)] \\
&= \Pr_{\mathsf{seed}'_i}[\mathsf{seed}'_i = \mathsf{seed}_i] \ \cdot \ \Pr_{y'_{i,K}}[\forall K \in S_i,\ y_{i,K} = y'_{i,K}] \ \cdot p_\pi \\
&= \left(\frac{1}{2^\lambda}\right)^M \cdot \left(\frac{1}{2^\lambda}\right)^{\sum_{i=1}^M |S_i|} \cdot \prod_{K\in\{0,1\}^\lambda} \frac{1}{(2^\lambda)_{q_K}} \\
&= 2^{-\lambda\cdot\sum_{K\in\{0,1\}^\lambda}|S'_K|} \cdot \prod_{K\in\{0,1\}^\lambda} \frac{1}{(2^\lambda)_{q_K}} \cdot 2^{-\lambda\cdot M},
\end{aligned}$$

where for $1 \le b \le a$, $(a)_b := a \cdot (a-1)\cdot(a-2)\cdots(a-b+1)$. Note that the last equality comes from the fact that $\sum_{i=1}^M |S_i| = \sum_{K\in\{0,1\}^\lambda} |S'_K|$.

We next compute the probability $\mathsf{Pr}_{\mathsf{rw}}(Q)$ that the real-world oracle is consistent with Q. We also denote by $((\mathsf{seed}'_i)_{i\le M}, (y'_{i,K})_{i\le M, K\in S_i}, (\pi_K)_{K\in\{0,1\}^\lambda})$ a setting of the real world. The main difference is that $(y'_{i,K})_{i\le M,K\in S_i}$ are now dependent on $(\mathsf{seed}'_i)_{i\le M}$. Denoting $p_\pi = \Pr_{\pi_K}[(\pi_K)_{K\in\{0,1\}^\lambda} \vdash Q_\pi)]$, we have

$$\begin{aligned}
&\mathsf{Pr}_{\mathsf{rw}}(Q) = \\
&\Pr[(\mathsf{seed}'_i = \mathsf{seed}_i) \ \wedge \ (\forall K \in S_i,\ y_{i,K} = y'_{i,K}) \ \wedge \ ((\pi_K)_{K\in\{0,1\}^\lambda} \vdash Q_\pi)] \\
&= \Pr_{\mathsf{seed}'_i}[(\mathsf{seed}'_i = \mathsf{seed}_i) \ \wedge \ (\forall K \in S_i,\ y_{i,K} = \pi_K(\mathsf{seed}_i) \oplus \mathsf{seed}_i)] \ \cdot p_\pi \\
&= \Pr_{\mathsf{seed}'_i}[(\mathsf{seed}'_i = \mathsf{seed}_i) \ \wedge \ (\forall i \in S'_K,\ y_{i,K} = \pi_K(\mathsf{seed}_i) \oplus \mathsf{seed}_i)] \ \cdot p_\pi \\
&= \Pr_{\pi_K}\left[y_{i,K} = \pi_K(\mathsf{seed}_i) \oplus \mathsf{seed}_i) \middle| \ (\pi_K)_{K\in\{0,1\}^\lambda} \vdash Q_\pi) \right] \cdot p_\pi \cdot \Pr_{\mathsf{seed}'_i}[\mathsf{seed}'_i = \mathsf{seed}_i] \\
&= \frac{1}{2^{\lambda\cdot M}} \ \cdot \prod_{K\in\{0,1\}^\lambda} \frac{1}{(2^\lambda)_{q_K}} \cdot \Pr_{\pi_K}\left[y_{i,K} = \pi_K(\mathsf{seed}_i) \oplus \mathsf{seed}_i) \middle| \ (\pi_K)_{K\in\{0,1\}^\lambda} \vdash Q_\pi) \right]
\end{aligned}$$

Since $Q \in \mathcal{T}_{\mathsf{good}}$ then $\nexists(\mathsf{seed}_i, K, *) \in Q_\pi$ with $K \in S_i \wedge \nexists(*, K, \mathsf{seed}_i \oplus y_{i,K})$ with $K \in S_i$. This leads to

$$\begin{aligned}
&\Pr_{\pi_K}\left[y_{i,K} = \pi_K(\mathsf{seed}_i) \oplus \mathsf{seed}_i) \middle| \ (\pi_K)_{K\in\{0,1\}^\lambda} \vdash Q_\pi) \right] \\
&= \prod_{K\in\{0,1\}^\lambda} \Pr_{\pi_K}[\pi_K(\mathsf{seed}_i) = y_{i,K} \oplus \mathsf{seed}_i] = \prod_{K\in\{0,1\}^\lambda} \frac{1}{(2^\lambda - q_K)_{|S'_K|}}.
\end{aligned}$$

Putting equations together, we obtain

$$\mathsf{Pr}_{\mathsf{rw}}(Q) = \frac{1}{2^{\lambda\cdot M}} \ \cdot \prod_{K\in\{0,1\}^\lambda} \frac{1}{(2^\lambda)_{q_K}} \cdot \prod_{K\in\{0,1\}^\lambda} \frac{1}{(2^\lambda - q_K)_{|S'_K|}}$$

and eventually

$$\begin{aligned}
\forall Q \in \mathcal{T}_{\mathsf{good}},\ \frac{\mathsf{Pr}_{\mathsf{rw}}[Q]}{\mathsf{Pr}_{\mathsf{iw}}[Q]} = \frac{\mathsf{Pr}_{\mathsf{rw}}(Q)}{\mathsf{Pr}_{\mathsf{iw}}(Q)} &= \prod_{K\in\{0,1\}^\lambda} \frac{2^{\lambda\cdot\sum_{K\in\{0,1\}^\lambda}|S'_K|}}{(2^\lambda - q_K)_{|S'_K|}} \\
&= \prod_{K\in\{0,1\}^\lambda} \frac{2^{2N\cdot\lambda}}{(2^\lambda - q_K)_{|S'_K|}}.
\end{aligned}$$

Distinguishing advantage. Equipped with the above calculations, we can finally bound the distinguishing advantage of $\mathcal{D}^{\mathcal{O}^\pi}$. To upper bound $\mathsf{Adv}(\mathcal{D}^{\mathcal{O}^\pi})$, we upper bound the ratio $\frac{\mathsf{Pr}_{\mathsf{rw}}[Q]}{\mathsf{Pr}_{\mathsf{iw}}[Q]}$, which translates to computing a lower bound on $\prod_{K\in\{0,1\}^\lambda}(2^\lambda - q_K)_{|S'_K|}$. Denote $K_{\mathsf{max}} \in K \in \{0,1\}^\lambda$ the index of the set among all $\{S'_K\}_{K\in\{0,1\}^\lambda}$ that has $\max_K |S'_K|$ elements. Then we have

$$\begin{aligned}\prod_{K\in\{0,1\}^\lambda}(2^\lambda - q_K)_{|S'_K|} &\geq \prod_{K\neq K_{\mathsf{max}}}(2^\lambda)^{|S'_K|}\cdot(2^\lambda - q)_{\max_K |S'_K|}\\ &= (2^\lambda)^{\sum_{K\in\{0,1\}^\lambda}|S'_K|}\cdot\frac{(2^\lambda - q)_{\max_K |S'_K|}}{(2^\lambda)^{\max_K |S'_K|}}\\ &= 2^{2N\cdot\lambda}\cdot\frac{(2^\lambda - q)_{\max_K |S'_K|}}{(2^\lambda)^{\max_K |S'_K|}} \geq 2^{2N\cdot\lambda}\cdot\left(\frac{2^\lambda - q}{2^\lambda}\right)^{\max_K |S'_K|}.\end{aligned}$$

$$\implies \frac{\mathsf{Pr}_{\mathsf{rw}}[Q]}{\mathsf{Pr}_{\mathsf{iw}}[Q]} \leq \left(\frac{2^\lambda}{2^\lambda - q}\right)^{\max_K |S'_K|} = \left(1 + \frac{q}{2^\lambda - q}\right)^{\max_K |S'_K|}.$$

The above yields

$$\frac{\mathsf{Pr}_{\mathsf{rw}}[Q]}{\mathsf{Pr}_{\mathsf{iw}}[Q]} \leq 1 + \frac{q\cdot\max_K |S'_K|}{2^\lambda - q}.$$

Then, using the H-coefficient theorem (Theorem 5.3), we get:

$$\mathsf{Adv}(\mathcal{D}^{\mathcal{O}^\pi}) = \frac{1}{2^{\lambda-1}}\cdot q\cdot\max_K |S'_K| + \frac{q\cdot\max_j |S'_K|}{2^\lambda - q}.$$

Plugging the bound on $|S'_K|$ from the claim finishes the proof. □

6 A Signature scheme from Regular Syndrome Decoding

In this section, we introduce a new signature scheme from the regular syndrome decoding assumption. A signature scheme is given by three algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. KeyGen returns a key pair $(\mathsf{pk}, \mathsf{sk})$ where pk and sk are the public and private key. Sign on an input a message m and the secret key sk, produces a signature σ. Verify, on input a message m, a public key pk and a signature σ, returns 0 or 1. Standard security notions for signature schemes are existential unforgeability against key-only attacks ($\mathsf{EUF\text{-}KO}$, Definition 18) and against chosen-message attacks ($\mathsf{EUF\text{-}CMA}$, Definition 17).

Definition 17 (EUF-CMA security). *Given a signature scheme* $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *and security parameter* λ*, we say that* Sig *is* $\mathsf{EUF\text{-}CMA}$*-secure if any PPT algorithm* $\mathcal{A}$ *has negligible advantage in the* $\mathsf{EUF\text{-}CMA}$ *game, defined as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} = \Pr\left[\begin{array}{c|c}\mathsf{Verify}(\mathsf{pk}, \mu^*, \sigma^*) = 1 & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(\{0,1\}^\lambda)\\ \wedge \mu^* \notin Q & (\mu^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})\end{array}\right],$$

where $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}$ *denotes* $\mathcal{A}$*'s access to a signing oracle with private key* sk *and* Q *denotes the set of messages* μ *that were queried to* $\mathsf{Sign}(\mathsf{sk},\cdot)$ *by* $\mathcal{A}$*.*

Definition 18 (EUF-KO security). *Given a signature scheme* $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *and security parameter* λ*, we say that* Sig *is* $\mathsf{EUF\text{-}KO}$*-secure if any PPT algorithm* $\mathcal{A}$ *has negligible advantage in the EUF-KO game, defined as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}KO}} = \Pr\left[\mathsf{Verify}(\mathsf{pk}, \mu^*, \sigma^*) = 1 \;\middle|\; \begin{array}{c}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(\{0,1\}^\lambda)\\ (\mu^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{pk})\end{array}\right].$$

6.1 Description of the Signature Scheme

The key generation algorithm randomly samples a syndrome decoding instance $(H, \mathbf{y})$ with solution $\mathbf{x}$. We describe it on Figure 3. The signing algorithm with secret key $\mathsf{sk} = (H, \mathbf{y}, \mathbf{x})$ and message $m \in \{0,1\}^*$ is described on Figure 4. The verification algorithm with public key $\mathsf{pk} = (H, \mathbf{y})$, message $m \in \{0,1\}^*$, and signature σ, is described in Figure 6.

An optimization. For readability, the description of the signing and verification algorithms ignores an optimization that slightly reduces the signature size, but significantly complexifies the description. Concretely, because we know that the vectors u^e should be regular vectors, it suffices to share the $\mathsf{bs}-1$ first entries $(u_1, \cdots, u_{\mathsf{bs}-1})$ of each block of $\mathbf{u}^e$, since the last one can be reconstructed as $\bigoplus u_i \oplus 1$. This reduces the size of $\mathbf{u}$ from $K = w \cdot \mathsf{bs}$ to $w \cdot (\mathsf{bs}-1) = K - w$ bits. Consequently, the share u_n^e of u^e need also only be shared over $\mathbb{F}_2^{K-w}$. This reduces by w the size of aux_n^e for each $e \leq \tau$, hence overall by $\tau \cdot w$ the size of the signature. An additional byproduct of this optimization is that it reduces the number of possible "cheating" vectors u^e that a malicious prover could choose, which has some positive repercussions on the size of the RSD parameters (K, k, w) which we can choose (we elaborate in Section 7).

Inputs: A security parameter λ.

1. Sample $\mathsf{seed} \leftarrow \{0,1\}^\lambda$;
2. Set $H \leftarrow \mathsf{PRG}(\mathsf{seed})$ with $H \in \mathbb{F}_2^{k \times K}$;
3. Sample $\mathbf{x} \leftarrow_r [\mathsf{bs}]^w$ and set $\mathbf{y} \leftarrow H \cdot \mathsf{Expand}(\mathbf{x})$ and $\mathsf{sk} \leftarrow (\mathsf{seed}, \mathbf{x})$.

Fig. 3. Key generation algorithm of the signature scheme

Inputs: A secret key sk and a message $m \in \{0,1\}^*$.

Initialization.

- Parse sk as $(\mathsf{seed}, \mathbf{x})$;
- Let $H \leftarrow \mathsf{PRG}(\mathsf{seed})$ and $\mathbf{y} \leftarrow H \cdot \mathsf{Expand}(\mathbf{x})$; // $H \in \mathbb{F}_2^{k \times K}$ is a (pseudo)random matrix in systematic form.
- Sample $(K_0, K_1) \leftarrow_r \{0,1\}^\lambda \times \{0,1\}^\lambda$. Set $\mathsf{salt} \leftarrow (K_0, K_1)$.

Phase 1.
For each iteration $e \in [\tau]$:

- Sample $\mathsf{seed}^e \leftarrow_r \{0,1\}^\lambda$;
- For $d = 1$ to D, set $(X_{d,0}^e, R_{d,0}^e, U_{d,0}^e) \leftarrow (0,0,0) \in [\mathsf{bs}]^w \times [\mathsf{bs}]^w \times \{0,1\}^K$;
- Set $x_n^e \leftarrow \mathbf{x}$, $u_n^e \leftarrow 0$, and $r^e \leftarrow 0$;
- **For** $i = 1$ **to** $n-1$:
  1. Compute $\mathsf{seed}_i^e \leftarrow \mathsf{PPRF}_{\mathsf{salt}}(\mathsf{seed}^e, i)$; // Can be computed efficiently by always storing the path to the current node: to move from i to $i+1$, start from the closest ancestor of $i+1$ in the path to leave i.
  2. Set $\mathsf{state}_i^e \leftarrow \mathsf{seed}_i^e$;
  3. $(x_i^e, r_i^e, u_i^e, \mathsf{com}_i^e) \leftarrow \mathsf{PRG}(\mathsf{seed}_i^e)$; // $(x_i^e, r_i^e, u_i^e, \mathsf{com}_i^e) \in [\mathsf{bs}]^w \times [\mathsf{bs}]^w \times \{0,1\}^K \times \{0,1\}^\lambda$.
  4. $x_n^e \leftarrow x_n^e - x_i^e \bmod \mathsf{bs}$, $u_n^e \leftarrow u_n^e \oplus u_i^e$, and $r^e \leftarrow r^e + r_i^e \bmod \mathsf{bs}$;
  5. For all $d \leq D$ such that $i[d] = 0$, set: // $i[d]$ is the d-th bit of the integer i.
     - $X_{d,0}^e \leftarrow X_{d,0}^e + x_i^e \bmod \mathsf{bs}$;
     - $R_{d,0}^e \leftarrow R_{d,0}^e + r_i^e \bmod \mathsf{bs}$;
     - $U_{d,0}^e \leftarrow U_{d,0}^e \oplus u_i^e$;
- **On node** n**:**
  1. Compute $\mathsf{seed}_n^e \leftarrow \mathsf{PPRF}_{\mathsf{salt}}(\mathsf{seed}^e, n)$;
  2. Compute $r_n^e \leftarrow \mathsf{PRG}(\mathsf{seed}_n^e)$;
  3. $r^e \leftarrow r^e + r_n^e \bmod \mathsf{bs}$, $u^e \leftarrow \mathsf{Expand}(r^e)$, and $u_n^e \leftarrow u_n^e \oplus u^e$; // The $(x_i^e)_i$ form n pseudorandom shares of $\mathbf{x} \in [\mathsf{bs}]^w$, the $(r_i^e)_i$ form n pseudorandom shares of $r^e \in [\mathsf{bs}]^w$, and the $(u_i^e)_i$ form n pseudorandom shares of $u^e = \mathsf{Expand}(r^e) \in \{0,1\}^K$.
  4. Define $\mathsf{aux}_n^e \leftarrow (x_n^e, u_n^e)$;
  5. Set $\mathsf{state}_n^e \leftarrow \mathsf{aux}_n^e || \mathsf{seed}_n^e$ and $\mathsf{com}_n^e \leftarrow \mathsf{H}(\mathsf{state}_n^e)$.

Fig. 4. Signing algorithm of the signature scheme, initialization, and phase 1

Phase 2.

1. $h_1 \leftarrow \mathsf{H}_1(m, \mathsf{salt}, \mathsf{com}_1^1, \cdots, \mathsf{com}_n^1, \cdots, \mathsf{com}_1^\tau, \cdots, \mathsf{com}_n^\tau)$; // Accumulate the commitments inside the hash rather than storing and hashing all at once.
2. $\pi^e_{\{e\in\tau\}} \leftarrow \mathsf{PRG}_1(h_1)$. // $\pi^e \in \mathsf{Perm}([w])$.

Phase 3.
For each iteration $e \in [\tau]$:

1. $z^e \leftarrow x - \pi^e(r^e) \bmod \mathsf{bs}$;
2. For $d = 1$ to D, set:
   - $y^e_{d,0} \leftarrow H \cdot \mathsf{Shift}(\pi^e(U^e_{d,0}), z^e)$;
   - $y^e_{d,1} \leftarrow y^e_{d,0} \oplus \mathbf{y}$;
   - $z^e_{d,0} \leftarrow X^e_d - \pi^e(R^e_{d,0}) \mod \mathsf{bs}$;
   - $z^e_{d,1} \leftarrow z^e - z^e_{d,0} \bmod \mathsf{bs}$.

Phase 4.

1. $h_2 \leftarrow \mathsf{H}_2(m, \mathsf{salt}, h_1, (y^e_{d,b}, z^e_{d,b})_{d\le D, b\in\{0,1\}, e\le\tau})$;
2. Set $(b^e_1, \cdots b^e_D)_{e\le\tau} \leftarrow \mathsf{PRG}_2(h_2)$;
3. Let $i^e \leftarrow \sum_{d=1}^D b^e_d \cdot 2^{d-1}$.

Phase 5.

1. Output $\sigma = \Big(\mathsf{salt}, h_1, h_2, (\mathsf{CoPath}_{\mathsf{salt}}(i^e, \mathsf{seed}^e), z^e, \mathsf{com}^e_{i^e}, \mathsf{aux}^e_n)_{e\le\tau}\Big)$. // aux^e_n is not included if $i^e = n$.

Fig. 5. Signing algorithm of the signature scheme, phase 2 to 5

Inputs: A public key $\mathsf{pk} = (H, \mathbf{y})$, a message $m \in \{0,1\}^*$ and a signature σ.

1. Split the signature as follows:
$$\sigma = \Big(\mathsf{salt}, h_1, h_2, (\mathsf{CoPath}_{\mathsf{salt}}(i^e, \mathsf{seed}^e), z^e, \mathsf{com}^e_{i^e}, \mathsf{aux}^e_n)_{e\le\tau}\Big);$$
2. Recompute $\pi^e_{\{e\in\tau\}}$ where $\pi^e \in \mathsf{Perm}([w])$ via a pseudorandom generator using h_1;
3. Recompute $(b^e_1, \cdots b^e_D)_{e\le\tau}$ via a pseudorandom generator using h_2 and define $i^e \leftarrow \sum_{d=1}^D b^e_d \cdot 2^{d-1}$;
4. For each iteration $e \in [\tau]$,
   - For $d = 1$ to D:
     - Denote $b = 1 - b^e_d$;
     - Set $(X^e_{d,b}, R^e_{d,b}, U^e_{d,b}) \leftarrow (0,0,0) \in [\mathsf{bs}]^w \times [\mathsf{bs}]^w \times \{0,1\}^K$;
     - For each $i \ne i^e$:
       * Recompute seed^e_i from the $\mathsf{CoPath}_{\mathsf{salt}}(i^e, \mathsf{seed}^e)$;
       * If $i \ne n$, recompute $(x^e_i, r^e_i, u^e_i, \mathsf{com}^e_i) \leftarrow \mathsf{PRG}(\mathsf{seed}^e_i)$; else, parse aux^e_n as (x^e_n, u^e_n), and compute $r^e_n \leftarrow \mathsf{PRG}(\mathsf{seed}^e_n)$;
       * If $i[d] = b$, update:
         - $X^e_{d,b} \leftarrow X^e_{d,b} + x^e_i \bmod \mathsf{bs}$;
         - $R^e_{d,b} \leftarrow R^e_{d,b} + r^e_i \bmod \mathsf{bs}$;
         - $U^e_{d,b} \leftarrow U^e_{d,b} \oplus u^e_i$;
     - Recompute $(y^e_{d,b}, z^e_{d,b})$ by simulating the Phase 3 of the signing algorithm as below:
       - $y^e_{d,b} \leftarrow H \cdot \mathsf{Shift}(\pi^e(U^e_{d,b}), z^e)$;
       - $z^e_{d,b} \leftarrow X^e_{d,b} - \pi^e(R^e_{d,b}) \mod \mathsf{bs}$;
     - Recompute $(y^e_{d,1-b}, z^e_{d,1-b})$ as below:
       - $y^e_{d,1-b} \leftarrow y^e_{d,b} \oplus \mathbf{y}$;
       - $z^e_{d,1-b} \leftarrow z^e - z^e_{d,b} \bmod \mathsf{bs}$;
5. Check if $h_1 \leftarrow \mathsf{H}_1(m, \mathsf{salt}, \mathsf{com}_1^1, \cdots, \mathsf{com}_n^1, \cdots, \mathsf{com}_1^\tau, \cdots, \mathsf{com}_n^\tau)$;
6. Check if $h_2 \leftarrow \mathsf{H}_2(m, \mathsf{salt}, h_1, (y^e_{d,b}, z^e_{d,b})_{d\le D, b\in\{0,1\}, e\le\tau})$;
7. Output ACCEPT if both conditions are satisfied.

Fig. 6. Verification algorithm of the signature scheme

Theorem 19. *Assume that* PPRF *is a* (q_s, τ)*-instance* $(t, \epsilon_{\mathsf{PPRF}})$*-secure PPRF, that* PRG *is a* (q_s, τ)*-instance* $(t, \epsilon_{\mathsf{PRG}})$*-secure PRG, and that any adversary running in time* t *has at advantage at most* ϵ_{SD}

against the regular syndrome decoding problem. Model the hash functions $\mathsf{H}_1, \mathsf{H}_2$ *as random oracles with output of length* 2λ*-bit and the pseudorandom generator* PRG_2 *as a random oracle. Then chosen-message adversary against the signature scheme depicted in Figure 4 and Figure 5, running in time* t*, making* q_s *signing queries, and making* q_1, q_2, q_3 *queries, respectively, to the random oracles* $\mathsf{H}_1, \mathsf{H}_2$ *and* PRG_2*, succeeds in outputting a valid forgery with probability*

$$\Pr[\text{Forge}] \leq \frac{q_s\,(q_s + q_1 + q_2 + q_3)}{2^{2\lambda}} + \epsilon_{\mathsf{PPRF}} + \varepsilon_{\mathsf{PRG}} + \epsilon_{\mathsf{SD}} + \Pr[X + Y = \tau] + \varepsilon_{\mathsf{G}} + \frac{1}{2^{\lambda}},$$

where $\epsilon = \mathsf{p} + \frac{1}{n} - \frac{\mathsf{p}}{n}$*, with* $\mathsf{p} = 4/B$ *and* $\varepsilon_{\mathsf{G}} = \varepsilon_{\mathsf{G}}(K, k, w, B)$ *is* $\Pr[(H, \mathbf{y}) \notin \mathsf{GOOD}_B]$*, which is defined on Lemma 21,* $X = \max_{\alpha \in Q_1}\{X_\alpha\}$ *and* $Y = \max_{\beta \in Q_2}\{Y_\beta\}$ *with* $X_\alpha \sim \mathsf{Binomial}(\tau, \mathsf{p})$ *and* $Y_\beta \sim \mathsf{Binomial}\left(\tau - X, \frac{1}{n}\right)$ *where* Q_1 *and* Q_2 *are sets of all queries to oracles* H_1 *and* H_2*.*

Computing the bound p from Theorem 19 requires a dedicated and involved combinatorial analysis which forms a core technical contribution of this work. We cover it extensively in section 6.2. The proof of Theorem 19 is deferred to next section 6.3.

6.2 Combinatorial Analysis of the Construction

In this section, we provide bounds on the probability that a random regular syndrome decoding instance $(H, \mathbf{y})$ are *bad*, in a sense that we formally define below. The bounds obtained in this section form a core component of the security analysis of our scheme in Section 6.3.

Bounding the Number of Distinct $\pi(\mathbf{u})\downarrow\mathbf{x}$ Solutions Let $\mathsf{Perm}_w := \mathsf{Perm}(w)$ denote the set of all permutations $\pi : [w] \mapsto [w]$. Given a vector $\mathbf{u} = (\mathbf{u}_1, \cdots, \mathbf{u}_w) \in \mathbb{F}_2^K$, where $(\mathbf{u}_1, \cdots, \mathbf{u}_w)$ forms a decomposition of $\mathbf{u}$ into w blocks $\mathbf{u}_i \in \mathbb{F}_2^{K/w}$, we write $\pi(\mathbf{u})$ to denote the vector $(\mathbf{u}_{\pi(1)}, \cdots, \mathbf{u}_{\pi(w)})$. That is, $\pi(\mathbf{u})$ is the vector obtained by shuffling the w blocks of $\mathbf{u}$ according to the permutation π.

For every $\mathbf{u} \in \mathbb{F}_2^K$, define $\mathsf{pn}(\mathbf{u}) = |\{\pi(\mathbf{u}) \mid \pi \in \mathsf{Perm}([w])\}|$. That is, $\mathsf{pn}(\mathbf{u})$ is the number of distinct vectors in $\mathbb{F}_2^K$ which can be obtained by permuting $\mathbf{u}$ blockwise. Given a bound B, define $\mathsf{PN}_B = \{\mathbf{u} \mid \mathsf{pn}(\mathbf{u}) > B\}$.

Definition 20 (GOOD_B). *Given a bound* B*,* GOOD_B *is defined as the set of syndrome decoding instances* $(H, \mathbf{y}) \in \mathbb{F}_2^{k \times K} \times \mathbb{F}_2^k$ *such that for every* $\mathbf{u} \in \mathsf{PN}_B \setminus \mathsf{Reg}_w$ *and for all* $\mathbf{x}^* \in [\mathsf{bs}]^w$*,*

$$\Pr_{\pi \leftarrow_r \mathsf{Perm}_w}[H \cdot (\pi(\mathbf{u})\downarrow\mathbf{x}^*) = \mathbf{y}] \leq \frac{4}{B}.$$

In other words, GOOD_B *is the set of syndrome decoding instances* $(H, \mathbf{y})$ *such that for every* $\mathbf{u} \notin \mathsf{Reg}_w$ *with at least* B *distinct blockwise permutations, at most a fraction* $4/B$ *of all blockwise permutations* $\pi(\mathbf{u})$ *are* close *to being solutions to* $H \cdot \mathbf{x} = \mathbf{y}$*, where we say that* $\pi(\mathbf{u})$ *is "close" to a solution if there exists a suitable cyclic shift of its block* $\pi(\mathbf{u})\downarrow\mathbf{x}^*$ *which is a solution.*

Equipped with this definition, we have the following lemma:

Lemma 21 (Most syndrome decoding instances are good).

$$\Pr_{H, \mathbf{y}}[(H, \mathbf{y}) \in \mathsf{GOOD}_B] > 1 - \varepsilon_{\mathsf{G}},$$

where

$$\varepsilon_{\mathsf{G}} = \binom{2B}{5} \cdot \frac{2^{K+1}}{B \cdot 2^{3k}} \cdot \left(10 + \frac{K^w}{w^w \cdot 2^k}\right).$$

Proof. The proof hinges upon a small technical lemma which we state below:

Claim. For any integer $t \leq K$ and every t-tuple of linearly-independent vectors $(\mathbf{v}_1, \cdots, \mathbf{v}_t)$, it holds that

$$\Pr_{H \leftarrow_r \mathbb{F}_2^{k \times K}}[H \cdot \mathbf{v}_i = \mathbf{0} \text{ for } i = 1 \text{ to } t] = \frac{1}{2^{k \cdot t}}.$$

Proof. Let V denote the matrix $(\mathbf{v}_1||\cdots||\mathbf{v}_t)$. Write $V = V^\top /\!/ V^\perp$, where $V^\top \in \mathbb{F}_2^{t\times t}$ denotes the invertible square matrix formed by the first t rows of V, and $V^\perp$ denotes the bottom $K-t$ rows. Given a matrix H, we write $H = H_L||H_R$, where H_L denotes the t leftmost columns of H, and H_R its remaining columns. We have:

$$\begin{aligned} H\cdot V = 0 &\iff H\cdot[V^\top /\!/ V^\perp] = 0\\ &\iff H\cdot[\mathsf{Id}_t /\!/ V^\perp\cdot(V^\top)^{-1}]\cdot V^\top = 0\\ &\iff (H_L\cdot\mathsf{Id}_t + H_R\cdot V^\perp\cdot(V^\top)^{-1})\cdot V^\top = 0\\ &\iff H_R\cdot V^\perp\cdot(V^\top)^{-1} = H_L. \end{aligned}$$

Therefore, when H is sampled as a uniformly random matrix, we have $\Pr[H\cdot V = 0] = \Pr[H_R\cdot V^\perp\cdot(V^\top)^{-1} = H_L] = 1/2^{k\cdot t}$, since the right hand side is a uniformly random matrix $H_L \leftarrow_r \mathbb{F}_2^{k\times t}$, sampled independently of the left hand side. The claim follows. ◻

Now, fix $\mathbf{u}\in\mathsf{PN}_B\backslash\mathsf{Reg}_w$ and $\mathbf{x}^*\in[\mathsf{bs}]^w$. Let $N\leftarrow\mathsf{pn}(\mathbf{u})$ and $\mathbf{u}^{(1)},\cdots,\mathbf{u}^{(N)}$ be the lexical ordering of all distinct vectors of the form $\pi(\mathbf{u})$ for some $\pi\in\mathsf{Perm}_w$. Fix any subset $S=\{i_1,\cdots,i_5\}\subset[N]$ of five indices. In the following, we will bound the probability

$$p(S) = \Pr_{H,\mathbf{y}}[H\cdot(\mathbf{u}^{(i_1)}\!\downarrow\!\mathbf{x}^*) = \mathbf{y}\wedge\cdots\wedge H\cdot(\mathbf{u}^{(i_5)}\!\downarrow\!\mathbf{x}^*) = \mathbf{y}].$$

Recall that a syndrome decoding instance $(H,\mathbf{y})$ is sampled by picking a uniformly random matrix $H\leftarrow_r F_2^{k\times K}$, a uniformly random regular vector $\mathbf{x}\leftarrow_r\mathsf{Reg}_w$, and setting $\mathbf{y}\leftarrow H\cdot\mathbf{x}$. When making the sampling of $\mathbf{x}$ explicit, the probability $p(S)$ rewrites to

$$p(S) = \Pr_{H,\mathbf{x}}[H\cdot(\mathbf{u}^{(i_1)}\!\downarrow\!\mathbf{x}^*\oplus\mathbf{x}) = \mathbf{0}\wedge\cdots\wedge H\cdot(\mathbf{u}^{(i_5)}\!\downarrow\!\mathbf{x}^*\oplus\mathbf{x}) = \mathbf{0}].$$

Now, write $(\mathbf{v}_1,\cdots,\mathbf{v}_5)\leftarrow(\mathbf{u}^{(i_1)}\!\downarrow\!\mathbf{x}^*\oplus\mathbf{x},\cdots,\mathbf{u}^{(i_5)}\!\downarrow\!\mathbf{x}^*\oplus\mathbf{x})$, which are random variables defined over the sampling of $\mathbf{x}$, and let Z_S denote the event (defined over the sampling of both $\mathbf{x}$ and H) that $H\cdot\mathbf{v}_i = \mathbf{0}$ for $i=1$ to 5 (in other words, $p(S)=\Pr[Z_S]$). If the vectors $(\mathbf{v}_1,\cdots,\mathbf{v}_5)$ were guaranteed to be linearly independent, we would immediately get $p(S)=\Pr[Z_S]=1/2^{5k}$ by the previous claim; however, they are not necessarily independent, and a more fine-grained approach is required. To bound $p(S)$, we make a few simple observations:

- Since $\mathbf{u}\notin\mathsf{Reg}_w$, it also holds that for any permutation π and shifts $\mathbf{x}^*$, $\pi(\mathbf{u})\downarrow\mathbf{x}^*\notin\mathsf{Reg}_w$ (since shuffling the blocks and cyclically shifting each block yields an invertible mapping that preserves regularity). This implies that $\mathbf{v}_j\neq\mathbf{0}$ holds with probability 1 for $j=1$ to 5 (since $\mathbf{v}_j = 0 \iff \mathbf{u}^{(i_1)}\!\downarrow\!\mathbf{x}^* = \mathbf{x}$, and $\mathbf{x}\in\mathsf{Reg}_w$).
- Because the $\mathbf{u}^{(i)}$ are pairwise distinct (by definition), the $\mathbf{v}_j$ are pairwise distinct.

Equipped with these observations, let us denote E_S the event that there exist three integers $\alpha\neq\beta\neq\gamma\in[5]$ such that $\mathbf{v}_\alpha\oplus\mathbf{v}_\beta\oplus\mathbf{v}_\gamma = 0$. Observe that

$$\begin{aligned} \Pr[E_S] &= \Pr_{\mathbf{x}}[\exists\alpha\neq\beta\neq\gamma\in[5]\ :\ (\mathbf{u}^{(i_\alpha)}\!\downarrow\!\mathbf{x}^*\oplus\mathbf{x})\oplus(\mathbf{u}^{(i_\beta)}\!\downarrow\!\mathbf{x}^*\oplus\mathbf{x})\oplus(\mathbf{u}^{(i_\gamma)}\!\downarrow\!\mathbf{x}^*\oplus\mathbf{x}) = 0]\\ &= \Pr_{\mathbf{x}}[\exists\alpha\neq\beta\neq\gamma\in[5]\ :\ (\mathbf{u}^{(i_\alpha)}\!\downarrow\!\mathbf{x}^*)\oplus(\mathbf{u}^{(i_\beta)}\!\downarrow\!\mathbf{x}^*)\oplus(\mathbf{u}^{(i_\gamma)}\!\downarrow\!\mathbf{x}^*) = \mathbf{x}]\\ &\leq \sum_{\alpha\neq\beta\neq\gamma}\Pr_{\mathbf{x}}[(\mathbf{u}^{(i_\alpha)}\!\downarrow\!\mathbf{x}^*)\oplus(\mathbf{u}^{(i_\beta)}\!\downarrow\!\mathbf{x}^*)\oplus(\mathbf{u}^{(i_\gamma)}\!\downarrow\!\mathbf{x}^*) = \mathbf{x}]\\ &\leq \binom{5}{3}\cdot\left(\frac{K}{w}\right)^{-w} = 10\cdot\left(\frac{K}{w}\right)^{-w}, \end{aligned}$$

which follows from a union bound over all possible size-3 subsets of $[5]$ and because there are $(K/w)^w$ vectors in Reg_w, hence a $(K/w)^{-w}$ probability (at most) that a random vector $\mathbf{x}\leftarrow_r\mathsf{Reg}_w$ is equal to the fixed vector $(\mathbf{u}^{(i_\alpha)}\!\downarrow\!\mathbf{x}^*)\oplus(\mathbf{u}^{(i_\beta)}\!\downarrow\!\mathbf{x}^*)\oplus(\mathbf{u}^{(i_\gamma)}\!\downarrow\!\mathbf{x}^*)$. Now, we have

$$\begin{aligned} \Pr[Z_S] &= \Pr[Z_S\mid E_S]\cdot\Pr[E_S] + \Pr[Z_S\mid\neg E_S]\cdot\Pr[\neg E_S]\\ &\leq 10\cdot(K/w)^{-w}\cdot\Pr[Z\mid E_S] + \Pr[Z\mid\neg E_S]. \end{aligned}$$

We now bound $\Pr[Z_S \mid E_S]$. For simplicity and without loss of generality, assume that after sampling $\mathbf{x}$, we have $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3 = 0$ (this is without loss of generality because we can always reorder the $\mathbf{v}_i$'s after sampling $\mathbf{x}$; note that the event E_S is defined only over the sampling of $\mathbf{x}$). Then, because $\mathbf{v}_4 \neq \mathbf{v}_3$, it necessarily holds that $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_4 \neq 0$. Furthermore, since the $\mathbf{v}_i$ are all pairwise distinct, and all nonzero, this implies that $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_4)$ are linearly independent. Then, using the claim:

$$\Pr[Z_S \mid E_S] \leq \Pr[H \cdot \mathbf{v}_1 = 0 \wedge H \cdot \mathbf{v}_2 = 0 \wedge H \cdot \mathbf{v}_4 = 0 \mid E_S] = \frac{1}{2^{3k}}.$$

We now bound $\Pr[Z_S \mid \neg E_S]$. By a similar reasoning, after sampling $\mathbf{x}$, it necessarily holds that there is a 4-tuple of the $\mathbf{v}_i$'s that does not XOR to 0 (since if all 4-tuples of the $\mathbf{v}_i$'s XOR to 0, we have $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3 \oplus \mathbf{v}_4 = \mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3 \oplus \mathbf{v}_5 = 0$, which implies $\mathbf{v}_4 = \mathbf{v}_5$, contradicting the fact that the $\mathbf{v}_i$'s are pairwise distinct). Without loss of generality, assume that $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4)$ do not XOR to 0. Because we condition on $\neg E_S$, it also holds that no 3-tuple of vectors from $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4)$ XOR to 0, and because the $\mathbf{v}_i$'s are pairwise distinct (*i.e.* no two-tuple XOR to 0) and nonzero, it follows that $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4)$ are linearly independent. By the previous claim:

$$\Pr[Z_S \mid \neg E_S] \leq \Pr[H \cdot \mathbf{v}_1 = 0 \wedge H \cdot \mathbf{v}_2 = 0 \wedge H \cdot \mathbf{v}_3 = 0 \wedge H \cdot \mathbf{v}_4 = 0 \mid E] = \frac{1}{2^{4k}}.$$

Eventually, we get

$$\begin{aligned} p(S) = \Pr[Z_S] &\leq 10 \cdot (K/w)^{-w} \cdot \Pr[Z_S \mid E_S] + \Pr[Z_S \mid \neg E_S] \\ &\leq \frac{1}{2^{3k}} \cdot \left(10 \cdot \left(\frac{K}{w}\right)^{-w} + \frac{1}{2^k}\right). \end{aligned}$$

We now finish the proof of Lemma 21 by a careful union bound. Given $\mathbf{u} \in \mathsf{PN}_B \backslash \mathsf{Reg}_w$ and $\mathbf{x}^* \in [\mathsf{bs}]^w$, let us partition the $N = \mathsf{pn}(\mathbf{u})$ vectors $\mathbf{u}^{(i)} {\downarrow} \mathbf{x}^*$ into $m \leq N/B$ blocks of at most $2B$ vectors each. Let $N_1, \cdots, N_m$ denote the m disjoint subsets $N_i \subset [N]$ of size $|N_i| \leq B$ corresponding to this partition. We first use a union bound over all possible blocks N_i, and all possible size-5 subsets of N_i:

$$\begin{aligned} &\Pr_{H,\mathbf{x}}[\exists i \leq m, \exists S_i \subset N_i \subset [N] \text{ with } |S_i| = 5 \ : \ H \cdot (\mathbf{u}^{(j)} {\downarrow} \mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0} \text{ for all } j \in S_i] \\ &\leq m \cdot \binom{2B}{5} \cdot \frac{1}{2^{3k}} \cdot \left(10 \cdot \left(\frac{K}{w}\right)^{-w} + \frac{1}{2^k}\right). \end{aligned}$$

In particular, this implies that for any fixed $\mathbf{u} \in \mathbb{F}_2^K$ with $\mathsf{pn}(\mathbf{u}) = N$, and any fixed $\mathbf{x}^* \in [\mathsf{bs}]^w$, there are at most $4 \cdot m$ indices $j \in [N]$ such that $H \cdot (\mathbf{u}^{(j)} {\downarrow} \mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0}$ with high probability (since with high probability, in each of the m block, there are at most 4 such indices):

$$\begin{aligned} &1 - m \cdot \binom{2B}{5} \cdot \frac{1}{2^{3k}} \cdot \left(10 \cdot \left(\frac{K}{w}\right)^{-w} + \frac{1}{2^k}\right) \\ &< \Pr_{H,\mathbf{x}}[\forall i \leq m, \forall S_i \subset N_i \subset [N] \text{ with } |S_i| = 5 \ : \ \exists j \in S_i, H \cdot (\mathbf{u}^{(j)} {\downarrow} \mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0}] \\ &= \Pr_{H,\mathbf{x}}[\forall i \leq m \ : \ \text{ there are at most } 4 \ j \in N_i \text{ s.t. } H \cdot (\mathbf{u}^{(j)} {\downarrow} \mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0}] \\ &\leq \Pr_{H,\mathbf{x}}[\exists \leq 4 \cdot m \text{ indices } j \in [N] \text{ such that } H \cdot (\mathbf{u}^{(j)} {\downarrow} \mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0} \text{ for all } j \in S_i] \\ &= \Pr_{H,\mathbf{x}}\left[\Pr_{\pi \in \mathsf{Perm}_w}[H \cdot (\pi(\mathbf{u}) {\downarrow} \mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0}] \leq \frac{4 \cdot N/B}{N} = \frac{4}{B}\right]. \end{aligned}$$

Next, we compute a union bound over all possible vectors $\mathbf{u}$ with permutation number $\mathsf{pn}(\mathbf{u}) = N$ (where $N \leq w!$, with equality when all blocks of $\mathbf{u}$ are distinct) and all shifts $\mathbf{x}^* \in [\mathsf{bs}]^w$. For any $N \in [w!]$, let $n(N)$ denote the total number of vectors $\mathbf{u} \in \mathbb{F}_2^K$ with $\mathsf{pn}(\mathbf{u}) = N$ (note that $\sum_{i \in [w!]} n(N) = 2^K$). We group all vectors $\mathbf{u}$ with $\mathsf{pn}(\mathbf{u}) = N$ into $n(N)/N$ equivalence classes $U_1, \cdots, U_{n(N)/N}$, where two vectors $\mathbf{u}_1, \mathbf{u}_2$ belong to the same equivalence class U_i if and only if there exists $\pi \in \mathsf{Perm}_w$ such that $\mathbf{u}_1 = \pi(\mathbf{u}_2)$ (note that each equivalence class is of size exactly N by definition of pn, and the U_i form a partition of the set $\{\mathbf{u} \in \mathbb{F}_2^K \ : \ \mathsf{pn}(\mathbf{u}) = N\}$). An important observation is that, because any two vectors $\mathbf{u}_1, \mathbf{u}_2$ that belong to the same equivalence class U_i generate the exact same N-tuple of distinct permuted vectors $(\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(N)})$ (ordered lexically), it suffices to do the union bound over all possible *equivalence classes* $(U_1, \cdots, U_{n(N)/N})$, and over all shifts $\mathbf{x}^*$:

$$\Pr_{H,\mathbf{x}}\left[\exists i \leq n(N)/N, \exists \mathbf{x}^* \in [\mathsf{bs}]^w, \Pr_{\pi \in \mathsf{Perm}_w}[H \cdot (\pi(\mathbf{u}){\downarrow}\mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0}] > \frac{4}{B}\right]$$
$$\leq \frac{n(N)}{N} \cdot \frac{N}{B} \cdot \binom{2B}{5} \cdot \left(\frac{K}{w}\right)^w \cdot \frac{1}{2^{3k}} \cdot \left(10 \cdot \left(\frac{K}{w}\right)^{-w} + \frac{1}{2^k}\right),$$

where the vector $\mathbf{u}$ in the probability denote any representent of the class U_i. Eventually, we use a union bound over all possible values $N \in [w!]$ with $N \geq B$:

$$\Pr_{H,\mathbf{y}}[(H, \mathbf{y}) \notin \mathsf{GOOD}_B]$$
$$= \Pr_{H,\mathbf{x}}\left[\exists N \geq B \in [w!], \exists i \leq n(N)/N, \exists \mathbf{x}^* \in [\mathsf{bs}]^w, \Pr_{\pi \in \mathsf{Perm}_w}[H \cdot (\pi(\mathbf{u}){\downarrow}\mathbf{x}^* \oplus \mathbf{x}) = \mathbf{0}] > \frac{4}{B}\right]$$
$$\leq \binom{2B}{5} \cdot \left(\frac{K}{w}\right)^w \cdot \frac{1}{B \cdot 2^{3k}} \cdot \left(10 \cdot \left(\frac{K}{w}\right)^{-w} + \frac{1}{2^k}\right) \cdot \sum_{N=B}^{w!} n(N)$$
$$< \binom{2B}{5} \cdot \frac{2^{K+1}}{B \cdot 2^{3k}} \cdot \left(10 + \frac{K^w}{w^w \cdot 2^k}\right),$$

which concludes the proof of Lemma 21. □

6.3 Security Analysis of the Signature Scheme

In this section, we prove Theorem 19.

Reducing to EUF-KO Security We start by proving the following lemma:

Lemma 22 (EUF-KO $\implies$ EUF-CMA)**.**

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}KO}} + \frac{q_s\,(q_s + q_1 + q_2 + q_3)}{2^{2\lambda}} + \epsilon_{\mathsf{PPRF}} + \epsilon_{\mathsf{PRG}}$$

Proof. Let us consider an adversary $\mathcal{A}$ against the EUF-CMA property of the signature scheme. To prove security we will define a sequence of experiments involving $\mathcal{A}$, where the first corresponds to the experiment in which $\mathcal{A}$ interacts with the real signature scheme, and the last one is an experiment in which $\mathcal{A}$ is using only random element independent from the witness.

Game 1 (Gm^1). This corresponds to the actual interaction of $\mathcal{A}$ with the real signature scheme. We need to bound the probability of what we'll call Forge, i.e. the event that $\mathcal{A}$ can generate a valid signature for a message that was not previously queried to the signing oracle.

Game 2 (Gm^2). For this step, we abort if the sampled salt salt collides with the value sampled in any of the previous queries to hash functions H_1 or H_2 or if the input of PRG_2 collides with the value obtained in any of the previous queries. Therefore we can bound this probability by

$$|\Pr[\mathsf{Gm}^1(\mathsf{Forge})] - \Pr[\mathsf{Gm}^2(\mathsf{Forge})]| \leq \frac{q_s \cdot (q_s + q_1 + q_2 + q_3)}{2^{2\lambda}}$$

Game 3 (Gm^3). The difference with the previous game is that now before signing a message we choose uniformly random values h_1, h_2 and i^*. Since Phase1, Phase3 and Phase5 are computed as before and the only change compared to the previous game is that we set the output of H_1 as h_1, the output of H_2 as h_2 and the output of $\mathsf{PRG}_2(h_2)$ as i^* then the difference in forgery probability is due to the event that query to H_1, H_2 or PRG_2 was ever made before but in this scenario Game 2 aborts, so

$$\Pr[\mathsf{Gm}^2(\mathsf{Forge})] = \Pr[\mathsf{Gm}^3(\mathsf{Forge})]$$

Game 4 (Gm^4) In this game we sample at random the i^*−th seed seed_{i^*} and the related co-path CoPath_{i^*}. By using all the seeds $\{\mathsf{seed}_i\}_{i \neq i^*}$ in the CoPath_{i^*} we will proceed by computing all the parties' views as well as the auxiliary material. Therefore, Phase 1 and Phase 3 are executed in the actual way (i.e. by using the real witness) except for i^*, for which the values are obtained randomly instead of using the PPRF. Distinguishing between this game and the previous one is perfectly equivalent to breaking the multi-instance security of the PPRF:

$$|\Pr[\mathsf{Gm}^4(\mathsf{Forge})] - \Pr[\mathsf{Gm}^6(\mathsf{Forge})]| \leq \epsilon_{\mathsf{PPRF}}$$

Game 5 (Gm^5). Now, before signing a message, we choose a uniformly random value to be used as the i^*- th party's view, i.e. $(x_{i^*}, r_{i^*}, u_{i^*})$, and its commitment com_{i^*}. Since in the previous game, these values were computed by using a multi-instance PRG on a random seed, with salt salt, we can bound

$$|\Pr[\mathsf{Gm}^4(\mathsf{Forge})] - \Pr[\mathsf{Gm}^6(\mathsf{Forge})]| \leq \epsilon_{\mathsf{PRG}}$$

Game 6 (Gm^6) In this game, we will change Phase 1 and Phase 3 by making the signer use the internal HVZK simulator described in 7. Looking in detail, the only change between the previous game and this one is that the auxiliary material aux is now selected as random. Anyway, since in the previous game aux was computed by using all real values but one (randomly chosen and never made public), there is essentially no difference between this game and the previous one. Therefore,

$$\Pr[\mathsf{Gm}^5(\mathsf{Forge})] = \Pr[\mathsf{Gm}^6(\mathsf{Forge})]$$

Game 7 (Gm^7). We say that an execution e^* of a query

$$h_2 = \mathsf{H}_2(m, \mathsf{salt}, h_1, (y^e_{d,b}, z^e_{d,b})_{d \leq D, b \in \{0,1\}, e \leq \tau})$$

defines a correct witness if the following criteria are satisfied:

- h_1 was output by a previous query

$$h_1 \leftarrow \mathsf{H}_1(m, \mathsf{salt}, \mathsf{com}^1_1, \cdots, \mathsf{com}^1_n, \cdots, \mathsf{com}^\tau_1, \cdots, \mathsf{com}^\tau_n);$$

- each $\mathsf{com}^{e^*}_i$ in this query was output by a previous query

$$\mathsf{com}^{e^*}_i = \mathsf{PRG}(\mathsf{seed}^{e^*}, i)$$

  for each $i \in [N]$;
- The vector $\mathbf{x}$ defined by the leaf party states $\{\mathsf{state}_i\}_{i \in N^D}$ satisfies $\mathsf{HW}(\mathbf{x}) = w$ and $H\mathbf{x} = \mathbf{y}$.

In this game, for each query of H_2 made by the adversary, we will check if there is an execution e^* that defines a correct witness. Calling this event Solve of course, since if it occurs then the states $\{\mathsf{state}^{e^*}_i\}$ define a solution for the RSD, the probability $\Pr[\mathsf{Solve} \leq \epsilon_{\mathsf{SD}}]$.

EUF-KO Security We prove the following lemma:

Lemma 23 (EUF-KO **security).**

$$\mathsf{Adv}^{\mathsf{EUF\text{-}KO}}_{\mathcal{A}} \leq \varepsilon_{\mathsf{SD}} + \Pr[X + Y = \tau] + \varepsilon_{\mathsf{G}} + \frac{1}{2^\lambda}.$$

Together with lemma 22, this completes the proof of Theorem 19. To prove EUF-KO security of our signature scheme, we first analyze the soundness of the underlying identification scheme, and then apply the standard reduction to EUF-KO security after compiling the scheme with the Fiat-Shamir transform. Concretely, our signature scheme is obtained by applying the Fiat-Shamir transform to the τ-fold parallel repetition of the identification scheme defined in Figure 8.

Step 1: (Sample Challenge).

1. Sample $\mathsf{CH}_1 = \pi \in \mathsf{Perm}([w])$ and $\mathsf{CH}_2 = i^* \in [n]$ and $\mathsf{salt} \in \{0,1\}^{2\lambda}$.

Step 2: (Sample Leaf Party States).

1. Sample $(x_{i^*}, r_{i^*}, u_{i^*}, \mathsf{com}_{i^*}) \leftarrow_r [\mathsf{bs}]^w \times [\mathsf{bs}]^w \times \{0,1\}^K \times \{0,1\}^\lambda$;
2. Sample the CoPath_{i^*} at random;
3. Sample $\mathsf{aux} \leftarrow_r [\mathsf{bs}]^w \times \{0,1\}^K$.

Step 3: (Generate Leaf Party Commitments).

1. For $i \neq i^*$:
   - If $i \neq n$:
     - Expand the leaf party into shares (x_i, r_i, u_i) and commitment com_i by using a PRG on seed_i;
   - If $i = n$:
     - Set $\mathsf{state}_n = \mathsf{seed}_n || \mathsf{aux}$ and compute $\mathsf{com}_n = H(\mathsf{state}_n)$;
     - Recompute r_n s.t. $u = \mathsf{Expand}(r)$ where $u = \sum_{i=1}^n u_i$ and $r = \sum_{i=1}^n r_i$.
2. Compute $\mathsf{COM} = H_1(m, \mathsf{salt}, \mathsf{com}_1, \ldots, \mathsf{com}_n)$.

Step 4: (Generate party communication).

1. Sample $z \in [\mathsf{bs}]^w$ at random;
2. For $d = 1$ to D:
   - Set $(X_{d,0}, R_{d,0}, U_{d,0}) \leftarrow (0,0,0) \in [\mathsf{bs}]^w \times [\mathsf{bs}]^w \times \{0,1\}^K$;
   - Compute
     - $X_{d,0} \leftarrow X_{d,0} + x_i \bmod \mathsf{bs}$;
     - $R_{d,0} \leftarrow R_{d,0} + r_i \bmod \mathsf{bs}$;
     - $U_{d,0} \leftarrow U_{d,0} \oplus u_i$;
     - $y_{d,0} \leftarrow H \cdot \mathsf{Shift}(\pi(U_{d,0}), z)$;
     - $y_{d,1} \leftarrow y_{d,0} \oplus \mathbf{y}$;
     - $z_{d,0} \leftarrow X_d - \pi(R_{d,0}) \mod \mathsf{bs}$;
     - $z_{d,1} \leftarrow z_{d,0} - z \bmod \mathsf{bs}$.

Step 5: (Output transcript).

1. $\mathsf{RSP}_1 = \mathsf{H}_2\left(m, \mathsf{salt}, \mathsf{COM}, (y_{d,b}, z_{d,b})_{d \leq D, b \in \{0,1\}}\right)$;
2. Program PRG_2 as a ROM s.t. $\mathsf{PRG}_2(\mathsf{RSP}_1) = \mathsf{CH}_2$;
3. $\mathsf{RSP}_2 = \mathsf{com}_{i^*}, \mathsf{CoPath}_{i^*}, \mathsf{aux}_n$.

Output $(\mathsf{COM}, \mathsf{RSP}_1, \mathsf{RSP}_2)$.

Fig. 7. Internal HVZK simulator for signing algorithm

Soundness of the identification scheme. The core of our analysis will be dedicated to show that from any cheating prover, one can extract a *weakly valid witness*. Concretely, a weakly valid witness is a pair $(\mathbf{v}, \mathbf{x}^*)$ where $\mathbf{v}$ is a solution to $H \cdot \mathbf{v} = \mathbf{y}$ which might not be regular, but which satisfies $\mathsf{pn}(\mathbf{v}{\downarrow}\mathbf{x}^*) \leq B$. In other words, this means that $\mathbf{v}$ contains mostly identical blocks "up to shift". Note that a regular vector contains only copies of the unit vector $\mathbf{e}_1$ "up to shift", hence this generalizes the class of regular vectors in a specific sense. Formally:

Definition 24. *A* weakly valid witness *to a syndrome decoding instance* $(H, \mathbf{y})$ *is a pair* $(\mathbf{v}, \mathbf{x}^*)$ *such that* $H \cdot \mathbf{v} = \mathbf{y}$ *and* $\mathbf{v} \in X$*, where* X *is defined as:*

$$X = \{\mathbf{v} \in \mathbb{F}_2^K \ : \ \exists \mathbf{u} \in \mathbb{F}_2^K \setminus \mathsf{PN}_B, \exists \mathbf{x}^* \in [\mathsf{bs}]^w, \mathbf{v} = \mathbf{u}{\downarrow}\mathbf{x}^*\}.$$

Additionally, the second term $\mathbf{x}^*$ *of the pair is a shift that satisfies* $\mathbf{v}{\downarrow}\mathbf{x}^* \in \mathbb{F}_2^K \setminus \mathsf{PN}_B$*.*

Lemma 25 (Soundness of the identification scheme in Figure 8). *Assume that* $\mathsf{H}_1, \mathsf{H}_2$ *are collision-resistant hash functions, that the mapping* $\mathsf{PRG}(\mathsf{seed})_{1..\lambda}$ *(i.e. the first* λ *bits of the output of* PRG *on an input* seed*) is computationally binding, and that the PRG used during key generation in* $H \leftarrow \mathsf{PRG}(\mathsf{seed})$ *is modeled as a random oracle (hence* $\mathsf{PRG}(\mathsf{seed})$ *selects a truly random matrix* H*).*

Initialization.

1. Parse the secret key sk as $(\mathsf{seed}, \mathbf{x})$;
2. Let $H \leftarrow \mathsf{PRG}(\mathsf{seed})$ and $\mathbf{y} \leftarrow H \cdot \mathsf{Expand}(\mathbf{x})$;
3. Sample $(K_0, K_1) \leftarrow_r \{0,1\}^\lambda \times \{0,1\}^\lambda$. Set $\mathsf{salt} \leftarrow (K_0, K_1)$.

Round 1: (Prover to Verifier).

1. Sample $\mathsf{seed} \leftarrow_r \{0,1\}^\lambda$;
2. For $d = 1$ to D:
   - Set $(X_{d,0}, R_{d,0}, U_{d,0}) \leftarrow (0,0,0) \in [\mathsf{bs}]^w \times [\mathsf{bs}]^w \times \{0,1\}^K$;
   - Set $x_n \leftarrow \mathbf{x}$, $u_n \leftarrow 0$, and $r \leftarrow 0$.
3. **For** $i = 1$ **to** $n-1$:
   - Compute $\mathsf{seed}_i \leftarrow \mathsf{PPRF}_{\mathsf{salt}}(\mathsf{seed}, i)$;
     Set $\mathsf{state}_i \leftarrow \mathsf{seed}_i$ and $(x_i, r_i, u_i, \mathsf{com}_i) \leftarrow \mathsf{PRG}(\mathsf{seed}_i)$;
   - Set $x_n \leftarrow x_n - x_i \bmod \mathsf{bs}$, $u_n^e \leftarrow u_n \oplus u_i$, and $r \leftarrow r + r_i \bmod \mathsf{bs}$;
   - For all $d \leq D$ such that $i[d] = 0$:
     Set $X_{d,0} \leftarrow X_{d,0} + x_i \bmod \mathsf{bs}$, $R_{d,0} \leftarrow R_{d,0} + r_i \bmod \mathsf{bs}$,
     and $U_{d,0} \leftarrow U_{d,0} \oplus u_i$.
4. **On node** n**:**
   - Compute $\mathsf{seed}_n \leftarrow \mathsf{PPRF}_{\mathsf{salt}}(\mathsf{seed}, n)$, $r_n \leftarrow \mathsf{PRG}(\mathsf{seed}_n)$, and set $r \leftarrow r + r_n \bmod \mathsf{bs}$, $u \leftarrow \mathsf{Expand}(r)$, and $u_n \leftarrow u_n \oplus u$;
   - Define $\mathsf{aux}_n \leftarrow (x_n, u_n)$ and set $\mathsf{state}_n \leftarrow \mathsf{aux}_n || \mathsf{seed}_n$ and $\mathsf{com}_n \leftarrow \mathsf{H}(\mathsf{state}_n)$;
   - Compute and send $h_1 \leftarrow \mathsf{H}_1(\mathsf{salt}, \mathsf{com}_1, \cdots, \mathsf{com}_n)$.

Round 2: (Verifier to Prover).

1. Send $\pi \leftarrow_r \mathsf{Perm}([w])$.

Round 3: (Prover to Verifier).

1. Set $z \leftarrow x - \pi(r) \bmod \mathsf{bs}$;
2. For $d = 1$ to D:
   - Set $y_{d,0} \leftarrow H \cdot \mathsf{Shift}(\pi(U_{d,0}), z)$, $y_{d,1} \leftarrow y_{d,0} \oplus \mathbf{y}$, $z_{d,0} \leftarrow X_d - \pi(R_{d,0}) \bmod \mathsf{bs}$, and $z_{d,1} \leftarrow z_{d,0} - z \bmod \mathsf{bs}$;
   - Compute and send $h_2 \leftarrow \mathsf{H}_2(\mathsf{salt}, h_1, (y_{d,b}, z_{d,b})_{d \leq D, b \in \{0,1\}})$.

Round 4: (Verifier to prover).

1. Send $(b_1, \cdots b_D) \leftarrow_r \{0,1\}^D$. Let $i \leftarrow \sum_{d=1}^{D} b_d \cdot 2^{d-1}$.

Round 5: (Prover to Verifier).

1. Send $(\mathsf{salt}, z, (\mathsf{CoPath}_{\mathsf{salt}}(i, \mathsf{seed}), \mathsf{com}_i, \mathsf{aux}_n))$.

Fig. 8. A five-round identification scheme with secret key $\mathsf{sk} = (\mathsf{seed}, \mathbf{y})$ for the relation $\mathbf{y} = H \cdot \mathbf{x}$ with $\mathbf{y} \in \mathsf{Reg}_w$ and $H = \mathsf{PRG}(\mathsf{seed})$. The scheme has soundness $\varepsilon = \mathsf{p} + (1 - \mathsf{p})/n$.

Then with probability at least $1 - 1/2^\lambda - \varepsilon_{\mathsf{G}}$ over the random choice of the RSD instance $(H, \mathbf{y})$, there exists an expected polynomial time extractor algorithm which, given rewinding access to a prover $\tilde{\mathsf{P}}$ which generates an accepting proof with probability at least $\tilde{\varepsilon} > \mathsf{p} + 1/n - \mathsf{p}/n$, extracts a weakly valid witness $\mathbf{x}$ *for the relation $H \cdot \mathbf{x} = \mathbf{y}$.*

Looking ahead, our soundness proof does not prevent a cheating prover from coming up with a weakly valid witness which is not a true regular witness. This will be guaranteed by the fact that with probability $1 - 1/2^\lambda$ over the choice of a random instance $(H, \mathbf{y})$, the system of equations $H \cdot \mathbf{v} = \mathbf{y}$ does not have *any* solution in X beyond the regular solution. This holds for a suitable choice of the parameters (K, k, w), which we cover in detail in Section 7. The term $1/2^\lambda$ in the bound of Lemma 23 reflects the probability that $(H, \mathbf{y})$ admits weakly valid solutions which are not regular.

Proof of Lemma 25. Let $\tilde{\mathsf{P}}$ be a prover which manages to generate an accepting proof with probability $\tilde{\varepsilon} > \varepsilon$. We exhibit an *extractor* which finds a witness $\mathbf{x}$ such that $H \cdot \mathbf{x} = \mathbf{y}$, where $\mathbf{x}$ is guaranteed to be a *weakly valid* witness (see Definition 24). Let R denote the randomness used by $\tilde{\mathsf{P}}$ to generate the commitment h of the first round, and by R^* a possible realization of R. Let $\mathsf{Succ}_{\tilde{\mathsf{P}}}$ denote the event that $\tilde{\mathsf{P}}$ succeeds in convincing V. By hypothesis

$$\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}}] = \tilde{\varepsilon} > \varepsilon = \mathsf{p} + \frac{1}{n} - \frac{\mathsf{p}}{n}.$$

Let us fix an arbitrary value $\alpha \in \{0,1\}$ such that $(1-\alpha)\tilde{\varepsilon} > \varepsilon$, which exists since $\tilde{\varepsilon} > \varepsilon$. We say that a realization R^* of the prover randomness for the first flow is *good* if it holds that

$$\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^*] \geq (1-\alpha)\tilde{\varepsilon}.$$

Furthermore, by the Splitting Lemma (see e.g. [FJR22]), we have $\Pr[R \text{ good} | \mathsf{Succ}_{\tilde{\mathsf{P}}}] \geq \alpha$. Assume now that T_0 is the transcript of a successful execution of the zero-knowledge proof with $\tilde{\mathsf{P}}$. Let R^* denote the random coin used by $\tilde{\mathsf{P}}$ in the first round, and let i_0 denote the Round 4 message of the verifier. If R^* is *good*, then

$$\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^*] \geq (1-\alpha)\tilde{\varepsilon} > \varepsilon > \frac{1}{n},$$

which implies that there necessarily exists a second successful transcript T_1 with a different Round 4 message $i_1 \neq i_0$.

Consistency of (T_0, T_1). Let (π_0, i_0) and (π_1, i_1) be the verifier challenges in the successful transcripts T_0 and T_1 respectively, with $i_0 \neq i_1$. Let us denote $(\mathsf{state}^0_{i \neq i_0}, \mathsf{com}^0_{i_0})$ and $\left(\mathsf{state}^1_{i \neq i_1}, \mathsf{com}^1_{i_1}\right)$ the states (recomputed from the co-path included in the transcript) and the commitment in the transcripts T_0 and T_1 respectively. Suppose that $\exists i \in [n] \setminus \{i_0, i_1\}$ such that $\mathsf{state}^0_i \neq \mathsf{state}^1_i$. Then there are two possibilities:

– The commitments are different:

$$\mathsf{com}_i = \mathsf{PRG}(\mathsf{state}_i)_{1..\lambda} \neq \mathsf{PRG}(\mathsf{state}'_i)_{1..\lambda} = \mathsf{com}'_i.$$

But since T_0 and T_1 are accepting transcripts, this implies in particular that $h = \mathsf{H}_1(\mathsf{com}_1, \cdots, \mathsf{com}_n)$ and $h = \mathsf{H}_1(\mathsf{com}'_1, \cdots, \mathsf{com}'_n)$ which contradicts the collision resistance of H_1.

– The commitments are equal:

$$\mathsf{com}_i = \mathsf{PRG}(\mathsf{state}_i)_{1..\lambda} = \mathsf{PRG}(\mathsf{state}'_i)_{1..\lambda} = \mathsf{com}'_i.$$

This directly contradicts the binding property of PRG.

Therefore, it necessarily holds that the states are mutually consistent (that is $\mathsf{state}^0_{i \neq i_0, i_1} = \mathsf{state}^1_{i \neq i_0, i_1}$. Since $i_0 \neq i_1$, they jointly define a unique tuple $(\mathsf{state}_i)_{i \in [n]}$, from which we can recompute $\mathbf{x} = \sum_i x_i \bmod \mathsf{bs}$, $u = \bigoplus_i u_i$, and $r = \sum_i r_i \bmod \mathsf{bs}$. Let us denote $v \leftarrow u \uparrow r$.

Claim. The vector v belongs to $\mathbb{F}_2^K \setminus \mathsf{PN}_B$.

To prove the claim, we show that if $v \in \mathsf{PN}_B$, then $\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^*] \leq \varepsilon$, contradicting our assumption that R^* is good. Let us denote $\mathsf{BadPerm} = \mathsf{BadPerm}_{v,\mathbf{x}}$ the event (defined over the random choice of a permutation π, and for the fixed value of $(v, \mathbf{x}, H, \mathbf{y})$) that $\mathbf{y} = H \cdot (\pi(v) \downarrow \mathbf{x})$. Let ε_{G} denote the bound of Lemma 21.

By Lemma 21, it holds with probability $1 - \varepsilon_{\mathsf{G}}$ over the random choice of H that $\Pr[\mathsf{BadPerm}] \leq \mathsf{p}$ with $\mathsf{p} = 4/B$ (here, we use the fact that in the random oracle model, $H = \mathsf{PRG}(\mathsf{seed})$ is uniformly random). Now,

$$\begin{aligned}\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^*] &= \Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} \wedge \mathsf{BadPerm} | R = R^*] + \Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} \wedge \neg\mathsf{BadPerm} | R = R^*] \\ &\leq \mathsf{p} + (1-\mathsf{p}) \cdot \Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^* \wedge \neg\mathsf{BadPerm}].\end{aligned}$$

We now bound $\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^* \wedge \neg\mathsf{BadPerm}]$. Assume for the sake of contradiction that $\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^* \wedge \neg\mathsf{BadPerm}] > 1/n$. This implies that given any successful transcript T'_0 with fourth-round i'_0, there necessarily exists a second successful transcript T'_1 *with the same first three rounds* and a different fourth-round $i'_1 \neq i'_0$. Let us fix two such transcripts (T'_0, T'_1), and let π' be the (common) permutation sent in Round 2 of these transcripts.

By the same argument as before, T'_0 and T'_1 are necessarily consistent, and uniquely define a tuple $(\mathsf{state}'_i)_{i\in[n]}$. Furthermore, since we condition on $R = R^*$, meaning that the first flow h'_1 is the same as the first flow h_1 in T_0, T_1, it must holds that $(\mathsf{state}'_i)_{i\in[n]} = (\mathsf{state}_i)_{i\in[n]}$, the states uniquely defined by (T_0, T_1) (else, we contradict either the collision-resistance of H or the binding of PRG, as already shown).

Let $d \leq D$ be a position such that $i'_0[d] \neq i'_1[d]$. Without loss of generality (since we can always swap the roles of T'_0 and T'_1), let us assume that $i'_0[d] = 0$ and $i'_1[d] = 1$. Reconstruct the values $(y^{(0)}_{b,d}, z^{(0)}_{b,d})_{b\in\{0,1\}}$ using the seeds $(\mathsf{seed}_i)_{i\neq i'_0}$ and the permutation π' from transcript T'_0, using the same procedure as the verification procedure. Similarly, reconstruct the values $(y^{(1)}_{b,d}, z^{(1)}_{b,d})_{d\leq D, b\in\{0,1\}}$ using the seeds $(\mathsf{seed}_i)_{i\neq i'_0}$ and the permutation π' from the transcript T'_1 (which are the same as in T'_0). This yields

$$y^{(0)}_{d,0} = H \cdot (\pi'(U^{(0)}_{d,0}) \downarrow z^{(0)})$$
$$y^{(1)}_{d,1} = H \cdot (\pi'(U^{(1)}_{d,1}) \downarrow z^{(1)}),$$

where $z^{(0)}$ and $z^{(1)}$ are the Round 5 vectors included in the transcripts T'_0 and T'_1, and

$$z^{(0)}_{d,0} = X^{(0)}_{d,0} - \pi'(R^{(0)}_{d,0})$$
$$z^{(1)}_{d,1} = X^{(1)}_{d,1} - \pi'(R^{(1)}_{d,1}).$$

Now, because T'_0 and T'_1 share the same states $(\mathsf{state}'_i)_{i\leq n}$, it holds by construction that $U^{(0)}_{d,0} + U^{(1)}_{d,1} = u$, $X^{(0)}_{d,0} + X^{(1)}_{d,1} = \mathbf{x}$, $R^{(0)}_{d,0} + R^{(1)}_{d,1} = r$, and $\mathbf{y} = y^{(0)}_{d,0} + y^{(1)}_{d,1}$. Furthermore, by the collision-resistance of H_2, it must hold that $y^{(0)}_{d,b} = y^{(1)}_{d,b}$ and $z^{(0)}_{d,b} = z^{(1)}_{d,b}$ for every $b \in \{0,1\}$. The latter equality implies that $z^{(0)} = z^{(0)}_{d,0} + z^{(0)}_{d,1} = z^{(1)}$ (we denote z this value from now on). This gives

$$z = z^{(0)}_{d,0} + z^{(1)}_{d,1} = X^{(0)}_{d,0} - \pi'(R^{(0)}_{d,0}) + X^{(1)}_{d,1} - \pi'(R^{(1)}_{d,1}) = \mathbf{x} - \pi(r).$$

Furthermore,

$$\mathbf{y} = y^{(0)}_{d,0} + y^{(1)}_{d,1} = H \cdot (\pi'(U^{(0)}_{d,0} + \pi'(U^{(1)}_{d,1}) \downarrow z) = H \cdot (\pi'(u) \downarrow z).$$

We conclude by observing that $\pi(u) \downarrow z = \pi'(u) \downarrow (\mathbf{x} - \pi'(r)) = \pi'(u \uparrow v) \downarrow \mathbf{x} = \pi'(v) \downarrow \mathbf{x}$, hence we have $H \cdot (\pi'(v) \downarrow \mathbf{x}) = \mathbf{y}$, which is a contradiction since the sampling on π' is conditioned on $\neg\mathsf{BadPerm}$. Hence, assuming the collision-resistance of H_2, it necessarily holds that $\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^* \wedge \neg\mathsf{BadPerm}] \leq 1/n$. Finishing the proof:

$$\begin{aligned}\Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^*] &\leq \mathsf{p} + (1-\mathsf{p}) \cdot \Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}} | R = R^* \wedge \neg\mathsf{BadPerm}] \\ &\leq \mathsf{p} + (1-\mathsf{p}) \cdot \frac{1}{n} = \varepsilon,\end{aligned}$$

contradicting our assumption that R^* is good. Therefore, we have extracted a vector v, a tuple $\mathbf{x}$, and a permutation π' such that $H \cdot (\pi'(v) \downarrow \mathbf{x}) = \mathbf{y}$, yet $v \in \mathbb{F}_2^K \setminus \mathsf{PN}_B$.

The extractor. Equipped with the above analysis, we describe an extractor $\mathcal{E}$ which is given rewindable black-box access to a prover $\tilde{\mathsf{P}}$. Define $N \leftarrow \ln(2)/((1-\alpha)\tilde{\varepsilon} - \varepsilon)$. $\mathcal{E}$ works as follows:

- Run $\tilde{\mathsf{P}}$ and simulate a honest verifier V to get a transcript T_0. Restart until T_0 is a successful transcript.
- Do N times:
  - Run $\tilde{\mathsf{P}}$ with a honest V and the same randomness as in T_0 to get a transcript T_1.
  - If T_1 is a successful transcript with $i_0 \neq i_1$, extract the tuple $(\mathbf{x}, u, r)$ and the permutation π. Output $\pi(v) \downarrow \mathbf{x}$.

The end of the proof is perfectly identical to the analysis in [FJR22, Appendix F]: given that $\mathcal{E}$ found a first successful transcript T_0, we have

$$\begin{aligned}\Pr[\mathsf{Succ}^{T_1}_{\tilde{\mathsf{P}}} \wedge i_1 \neq i_0 | R \text{ good}] &= \Pr[\mathsf{Succ}^{T_1}_{\tilde{\mathsf{P}}} | R \text{ good}] - \Pr[\mathsf{Succ}^{T_1}_{\tilde{\mathsf{P}}} \wedge i_1 = i_0 | R \text{ good}] \\ &\geq (1-\alpha)\tilde{\varepsilon} - 1/n \geq (1-\alpha)\tilde{\varepsilon} - \varepsilon,\end{aligned}$$

hence by definition of N, $\mathcal{E}$ gets a second successful transcript with probability at least 1/2. From there, the analysis of the expected number of calls $\mathbb{E}[\mathsf{call}]$ of $\mathcal{E}$ to $\tilde{\mathsf{P}}$ is identical to [FJR22, Appendix F]:

$$\begin{aligned}&\mathbb{E}[\mathsf{call}] \leq 1 + (1 - \Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}}]) \cdot \mathbb{E}[\mathsf{call}] + \Pr[\mathsf{Succ}_{\tilde{\mathsf{P}}}] \cdot (N + (1 - \alpha/2) \cdot \mathbb{E}[\mathsf{call}]) \\ \implies\ &\mathbb{E}[\mathsf{call}] \leq \frac{2}{\alpha\tilde{\varepsilon}} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{\ln(2)}{(1-\alpha)\tilde{\varepsilon} - \varepsilon}\right),\end{aligned}$$

which gives an expected number of calls $\mathsf{poly}(\lambda, (\tilde{\varepsilon}-\varepsilon)^{-1})$ by setting $\alpha \leftarrow (1-\varepsilon/\tilde{\varepsilon})/2$ (corresponding to $(1-\alpha)\tilde{\varepsilon} = (\varepsilon + \tilde{\varepsilon})/2$). This concludes the proof.

From soundness to $\mathsf{EUF\text{-}KO}$ security. Given a five-round identification protocol where the probability of sampling a "bad" Round 3 challenge is bounded by p, and the probability of sampling a "bad" Round 5 challenge is bounded by $1/n$, it follows from a standard application of the Fiat-Shamir methodology (adapted to 5-round protocols) to the τ-fold parallel repetition of the identification scheme given in Figure 8 that, when modeling H_1 and H_2 with random oracles, there exists an extractor which extracts a weakly valid witness $\mathbf{x}^* \in X$ given any adversary that succeeds with probability at least $\Pr[X + Y = \tau]$, where with probability at least $1 - \varepsilon_{\mathsf{G}}$ over the random choice of $(H, \mathbf{y})$, it holds that $X = \max_{\alpha \in Q_1}\{X_\alpha\}$ and $Y = \max_{\beta \in Q_2}\{Y_\beta\}$ with $\mathsf{p} = 4/B$, $X_\alpha \sim \mathsf{Binomial}(\tau, \mathsf{p})$, and $Y_\beta \sim \mathsf{Binomial}\left(\tau - X, \frac{1}{n}\right)$ where Q_1 and Q_2 are sets of all queries to the oracles H_1 and H_2. With probability at least $1 - 1/2^\lambda$, this weakly valid witness is necessarily a regular witness. This concludes the proof.

7 Parameter Selection and Efficiency

7.1 Parameters Selection Process

In this section, we explain how to select parameters for our new signature scheme. The first goal is to pick parameters that minimize the number of repetitions τ of the underlying identification scheme, since this parameter has a large impact on the signature size. Concretely, as in previous works, we choose τ such that the cost of the forgery attack on the Fiat-Shamir-compiled signature is at least 2^{128}, where cost is given by the formula below (which comes from the attack of Kales and Zaverucha [KZ20]):

$$\mathsf{cost} = \min_{\tau_1, \tau_2 : \tau_1 + \tau_2 = \tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} \mathsf{p}^i (1-\mathsf{p})^{\tau - i}} + n^{\tau_2} \right\}. \tag{1}$$

We observe that setting $B = 2^{-70}$ in $\mathsf{p} = 4/B$ suffices to guarantee that τ is always the smallest possible for any given number of leaves $n = 2^D$ (*i.e.* reducing p further does not reduce τ) in the range $D \in \{8, \cdots, 17\}$. The choice of the number of leaves, 2^D, is a tradeoff parameter: larger values of D yield smaller signature size, at the expense of a larger runtime.

Finding a bound on k. A crucial aspect of our parameter selection process is that our combinatorial analysis in section 6.2 only guarantees that with very high probability, for any $(\mathbf{u}, \mathbf{x}^*) \in \mathsf{PN}_B \times [\mathsf{bs}]^w$ where $\mathbf{u}$ is not regular, $\pi(\mathbf{u}) \downarrow \mathbf{x}^*$ will not be a valid solution $\mathbf{v}$ to $H \cdot \mathbf{v} = \mathbf{y}$. However, it says nothing about vectors $\mathbf{u}$ outside PN_B, that is, vectors with low permutation number $\mathsf{pn}(\mathbf{u}) \leq B$. Therefore, we must select RSD parameters such that, with overwhelming probability, there will not be *any* solution $\mathbf{v}$ to $H \cdot \mathbf{v} = \mathbf{y}$ of the form $\mathbf{v} = \mathbf{u} \downarrow \mathbf{x}^*$ for $\mathbf{u} \in \mathbb{F}_2^K \setminus \mathsf{PN}_B$. Formally, define the set $X = \{\mathbf{v} \in \mathbb{F}_2^K \ :\ \exists \mathbf{u} \in \mathbb{F}_2^K \setminus \mathsf{PN}_B, \exists \mathbf{x}^* \in [\mathsf{bs}]^w, \mathbf{v} = \mathbf{u} \downarrow \mathbf{x}^*\}$. To guarantee that there will not be any solution $\mathbf{v} \in X$ to $H \cdot \mathbf{v} = \mathbf{y}$, it suffices to pick $k \geq |X| + \lambda$. This follows from a standard "Gilbert-Varshamov-style" analysis: when sampling a random instance $(H, \mathbf{y} = H \cdot \mathbf{x})$ of the RSD problem, the expected number of solutions in X (beyond $\mathbf{x}$) is

$$\mathop{\mathbb{E}}_{H,\mathbf{x}}[|\{\mathbf{x}' \ :\ H \cdot \mathbf{x}' = H \cdot \mathbf{x} \ \wedge\ \mathbf{x}' \in X\}|] = \sum_{\substack{\mathbf{x}' \neq \mathbf{x} \\ \mathbf{x}' \in X}} \Pr_{H,\mathbf{x}}[H \cdot \mathbf{x}' = H \cdot \mathbf{x}] = \frac{|X| - 1}{2^k},$$

and we conclude with a Markov bound. To choose k, we use a bound on $|X|$:

Lemma 26. *Let* $\mathsf{P_{i,w}}$ *denote the set of* integer partitions *of* w *in* i *parts,* i.e.*, the set of all tuples* $(k_1, \cdots, k_i)$ *with* $0 < k_1 \leq k_2 \leq \cdots \leq k_i \leq w$ *such that* $\sum_{j=1}^{i} k_j = w$. *Let* $\mathsf{T_B}$ *denote the function such that* $\mathsf{T_B}(x) = x$ *when* $x \leq B$*, and* $\mathsf{T_B}(x) = 0$ *when* $x > B$. *Then*

$$|X| \leq \mathsf{bs}^w \cdot \sum_{i=1}^{L} \left(\binom{L}{i} \cdot i! \cdot \sum_{(k_1,\cdots,k_i)\in \mathsf{P_{i,w}}} \mathsf{T_B}\left(\frac{w!}{\prod_{j=1}^{i}(k_j)!} \right) \right),$$

where L *is (using the Euler totient* ϕ *and denoting* $a|b$ *for "a divides b"):*

$$L = \frac{1}{\mathsf{bs}} \cdot \sum_{\substack{i \leq \mathsf{bs} \\ i \ odd}} \sum_{d|\gcd(\mathsf{bs}-i,i)} \phi(d) \cdot \binom{\mathsf{bs}/d}{i/d}.$$

The proof of Lemma 26 follows from a counting argument, which we detail below. We enumerate over the bs^w possible vectors $\mathbf{x}^* \in [\mathsf{bs}]^w$, and over all possible $\mathbf{u}$ with $\mathsf{pn}(\mathbf{u}) \geq B$. To count the latter, we proceed in steps:

Counting the number of distinct blocks. We compute the number L of possible distinct blocks. A loose upper bound would be $L \leq 2^{\mathsf{bs}}$ (since a block is a vector in $\mathbb{F}_2^{\mathsf{bs}}$). However, because we already enumerate over all possible shifts $\mathbf{x}^*$ of the w blocks, we must only count the number of distinct blocks *up to cyclic shift.* In combinatorics, this amounts to counting the number of length-bs necklaces with two colors. Additionally, because of the optimization given in Section 6 where the last entry of each block is fixed such that all entries of a block XOR to 1, we only need to enumerate over all necklaces with an odd number of ones. The formula for L in Lemma 26 is a direct application of Pólya's enumeration theorem [Red27], a classical theorem on the combinatorics of necklaces.

Counting the number of vectors. For $i = 1$ to L, we count the number of vectors which have exactly i distinct blocks. There are $\binom{L}{i}$ ways to select the i distinct blocks out of L possible blocks. Since each vector has w blocks in total, we enumerate over all partitions of the integer w in exactly i parts $0 < k_1 \leq k_2 \leq \cdots \leq k_i \leq w$, where k_j denotes the number of copies of the j-th block from the selection. Because we enumerate over *ordered* partitions, we are ordering the i selected blocks by number of copies; hence, we multiply by $i!$ to account for all possible configurations of number of copies (this is a slightly loose upper bound, since some partitions may have equal numbers $k_j = k_{j+1}$: the right value would be to multiply by the factorial of the number of distinct integers in $(k_1, \cdots, k_j)$, but we ignore this for simplicity). Then, having fixed a choice of i specific distinct blocks and the numbers $(k_1, \cdots, k_i)$ of copies of each block, there are $w!/\prod_{j=1}^{i}(k_j)!$ distinct blockwise permutations of (this is the standard combinatorial formula for counting multisets). Eventually, since we only want to keep vectors whose permutation number is at most B, we keep only in the count the vectors for which $w!/\prod_{j=1}^{i}(k_j)! \leq B$ (this is the purpose of the threshold function $\mathsf{T_B}$ in the formula). This yields the formula stated in Lemma 26.

Computing $|X|$. It remains to compute explicitly the formula of Lemma 26. We used a Python script to perform the calculation. A small nontriviality is that enumerating over all integer partitions of w (which is around 120) would be very slow. Fortunately, we observe that the condition $w!/\prod_{j=1}^{i}(k_j)! \leq B$, together with the bound L on i, impose a sharp bound on the value of k_i: a quick calculation shows that we need $k_i \geq w/2$ to be such that $\binom{w}{k_i} \leq B$. Given this bound on k_i, we enumerate over all remaining possible values of k_i, and compute the number of partitions of $w - k_i$ into $i - 1$ parts to obtain the rest of the partition. We provide the script used to compute this bound in Appendix B of the Supplementary Material.

Finding (K, w). To find the RSD parameters (K, k, w), we proceed iteratively: we fix a choice of K, w and compute the value of k as $|X| + 128$ (note that X depends on (K, w)), using our script to compute the bound on $|X|$ from Lemma 26. Then, we rely on the estimator implemented in state-of-the-art cryptanalysis of [ES23] (which improves over a previous cryptanalysis from [CCJ23]) to compute (an estimate of) the bit security of the instance obtained against all known attacks on RSD. If the bit security is below 128, we increase K by 1 and start over (every time, we also compute the parameters for a list of weight parameters w, since the impact of w on the proof size is slightly subtle). Eventually, after settling for a choice of (K, k, w), we check that the probability bound of Lemma 21 is overwhelming (with our choice of parameters, it is always above $1 - 2^{-200}$).

7.2 Concrete Parameters and Implementation

We outline below a few parameter sets, for different values of $D \in \{8, \cdots, 17\}$. For all values of D, the smallest signature size was achieved by setting $K = 1560$, $k = 817$, $w = 195$, $\mathsf{bs} = 8$.

Table 2. Signature size and signing time for various values of D, using the parameter set $K = 1560$, $k = 817$, $w = 195$, $\mathsf{bs} = 8$. Timings computing on one core of an Intel Core i7 processor 14700KF.

D	τ	$\lvert\sigma\rvert$	signing time	verification time
8	17	7.73 kB	1.65 ms	1.69 ms
9	15	7.07 kB	3.03 ms	3.07 ms
10	13	6.35 kB	5.35 ms	5.59 ms
11	12	6.06 kB	10.2 ms	10.9 ms
12	11	5.73 kB	19.1 ms	21.3 ms
13	10	5.38 kB	44.0 ms	49.8 ms
15	9	5.13 kB	141 ms	166 ms
17	8	4.82 kB	-	-

We implemented our signature scheme in C. Our implementation is a proof of concept implementation: we did not use any of the optimizations such as batching, vectorization, or bit slicing, and an optimized implementation can likely achieve significantly faster runtime. We used the AES-NI instruction set to implement our multi-instance PPRF and our multi-instance PRG from Section 5. All our experiments were run on one core of an Intel Core i7 processor 14700KF. The following optimization flags have been used during compilation: `-O3 -flto -mavx2 -mpclmul -msse4.2 -maes -rdrnd`. We note that our verification time is slightly slower than our signing time. This is an artifact of our proof-of-concept implementation: a more properly optimized implementation would have verification run slightly faster than signing. We plan to optimize our implementation in the future. To have a fair comparison with SDitH, we used their benchmarking framework to obtain our performance results.

Acknowledgement

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 945332.
Dung Bui, Eliana Carozza, and Geoffroy Couteau acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE). This work was also supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute. The work of Dung Bui is supported by Dim Math Innov funding from the Paris Mathematical Sciences Foundation (FSMP) funded by the Paris Ile-de-France Region.
The work of Antoine Joux has been supported by the European Union's H2020 Programme under grant agreement number ERC-669891.

References

AFS03. D. Augot, M. Finiasz, and N. Sendrier. A fast provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2003/230, 2003. `https://eprint.iacr.org/2003/230`.

AGH+23. C. Aguilar Melchor, N. Gama, J. Howe, A. Hülsing, D. Joseph, and D. Yue. The return of the SDitH. In *EUROCRYPT 2023, Part V*, *LNCS* 14008, pages 564–596. Springer, Heidelberg, April 2023.

BBdSG+23. C. Baum, L. Braun, C. D. de Saint Guilhem, M. Klooß, E. Orsini, L. Roy, and P. Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*, *Lecture Notes in Computer Science* 14085, pages 581–615. Springer, 2023.

BCdSG24. D. Bui, K. Cong, and C. D. de Saint Guilhem. Improved all-but-one vector commitment with applications to post-quantum signatures. Cryptology ePrint Archive, Paper 2024/097, 2024. `https://eprint.iacr.org/2024/097`.

BCG$^+$19a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

BCG$^+$19b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 489–518. Springer, Heidelberg, August 2019.

BCG$^+$20a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

BCG$^+$20b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *CRYPTO 2020, Part II*, *LNCS* 12171, pages 387–416. Springer, Heidelberg, August 2020.

BCG$^+$21. E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 871–900. Springer, Heidelberg, October 2021.

BCG$^+$22. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, N. Resch, and P. Scholl. Correlated pseudorandomness from expand-accumulate codes. In *CRYPTO 2022, Part II*, *LNCS* 13508, pages 603–633. Springer, Heidelberg, August 2022.

BCGI18. E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

BDK$^+$21. C. Baum, C. Delpech de Saint Guilhem, D. Kales, E. Orsini, P. Scholl, and G. Zaverucha. Banquet: Short and fast signatures from AES. In *PKC 2021, Part I*, *LNCS* 12710, pages 266–297. Springer, Heidelberg, May 2021.

BGI14. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, *LNCS* 8383, pages 501–519. Springer, Heidelberg, March 2014.

BGI15. E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT 2015, Part II*, *LNCS* 9057, pages 337–367. Springer, Heidelberg, April 2015.

BGI16. E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

BGI19. E. Boyle, N. Gilboa, and Y. Ishai. Secure computation with preprocessing via function secret sharing. In *TCC 2019, Part I*, *LNCS* 11891, pages 341–371. Springer, Heidelberg, December 2019.

BLPS11. D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. Really fast syndrome-based hashing. In *AFRICACRYPT 11*, *LNCS* 6737, pages 134–152. Springer, Heidelberg, July 2011.

BW13. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT 2013, Part II*, *LNCS* 8270, pages 280–300. Springer, Heidelberg, December 2013.

CCJ23. E. Carozza, G. Couteau, and A. Joux. Short signatures from regular syndrome decoding in the head. In *EUROCRYPT 2023, Part V*, *LNCS* 14008, pages 532–563. Springer, Heidelberg, April 2023.

CDG$^+$20. M. Chase, D. Derler, S. Goldfeder, J. Katz, V. Kolesnikov, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, X. Wang, et al. The picnic signature scheme. *Submission to NIST Post-Quantum Cryptography project*, 2020.

CLY$^+$24. H. Cui, H. Liu, D. Yan, K. Yang, Y. Yu, and K. Zhang. Resolved: Shorter signatures from regular syndrome decoding and vole-in-the-head. Cryptology ePrint Archive, Paper 2024/040, 2024. `https://eprint.iacr.org/2024/040`.

CRR21. G. Couteau, P. Rindal, and S. Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO 2021, Part III*, *LNCS* 12827, pages 502–534, Virtual Event, August 2021. Springer, Heidelberg.

CS14. S. Chen and J. P. Steinberger. Tight security bounds for key-alternating ciphers. In *EUROCRYPT 2014*, *LNCS* 8441, pages 327–350. Springer, Heidelberg, May 2014.

DDOS19. C. Delpech de Saint Guilhem, L. De Meyer, E. Orsini, and N. P. Smart. BBQ: Using AES in picnic signatures. In *SAC 2019*, *LNCS* 11959, pages 669–692. Springer, Heidelberg, August 2019.

ES23. A. Esser and P. Santini. Not just regular decoding: Asymptotics and improvements of regular syndrome decoding attacks. *Cryptology ePrint Archive*, 2023.

FGS07. M. Finiasz, P. Gaborit, and N. Sendrier. Improved fast syndrome based cryptographic hash functions. In *Proceedings of ECRYPT Hash Workshop*, page 155. Citeseer, 2007.

FJR22. T. Feneuil, A. Joux, and M. Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In *CRYPTO 2022, Part II*, *LNCS* 13508, pages 541–572. Springer, Heidelberg, August 2022.

GGM86. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

GI14. N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, *LNCS* 8441, pages 640–658. Springer, Heidelberg, May 2014.

GKWY20. C. Guo, J. Katz, X. Wang, and Y. Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841. IEEE Computer Society Press, May 2020.

GYW+23. X. Guo, K. Yang, X. Wang, W. Zhang, X. Xie, J. Zhang, and Z. Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In *EUROCRYPT 2023, Part I*, *LNCS* 14004, pages 330–362. Springer, Heidelberg, April 2023.

HOSS18. C. Hazay, E. Orsini, P. Scholl, and E. Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In *CRYPTO 2018, Part III*, *LNCS* 10993, pages 3–33. Springer, Heidelberg, August 2018.

IKOS07. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

KPTZ13. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

KZ20. D. Kales and G. Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In *CANS 20*, *LNCS* 12579, pages 3–22. Springer, Heidelberg, December 2020.

MDCYA11. M. Meziani, Ö. Dagdelen, P.-L. Cayrel, and S. M. E. Yousfi Alaoui. S-fsb: An improved variant of the fsb hash family. In *International Conference on Information Security and Assurance*, pages 132–145. Springer, 2011.

MFG+23. C. A. Melchor, T. Feneuil, N. Gama, S. Gueron, J. Howe, D. Joseph, A. Joux, E. Persichetti, T. H. Randrianarisoa, M. Rivain, et al. The syndrome decoding in the head (sd-in-the-head) signature scheme–algorithm specifications and supporting documentation, 2023.

MSY21. J.-P. Münch, T. Schneider, and H. Yalame. Vasa: Vector aes instructions for security applications. In *Annual Computer Security Applications Conference*, pages 131–145, 2021.

Pat09. J. Patarin. The "coefficients h" technique. In *Selected Areas in Cryptography*, pages 328–345, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

Red27. J. H. Redfield. The theory of group-reduced distributions. *American Journal of Mathematics*, 49(3):433–455, 1927.

RS21. P. Rindal and P. Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 901–930. Springer, Heidelberg, October 2021.

WYKW21. C. Weng, K. Yang, J. Katz, and X. Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.

YSWW21. K. Yang, P. Sarkar, C. Weng, and X. Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021.

YWL+20. K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

Supplementary Material

A Further Applications of Multi-Instance PPRFs

We expect our tight analysis of fast PPRFs in the multi-instance setting to find applications beyond the realm of MPC-in-the-head signatures. As a sample application, pseudorandom correlation functions [BCG+20a, BCG+22], which are used to efficiently generate correlated randomness in secure computation, are typically constructed using a large number of distributed point functions (DPFs). DPFs are very similar to GGM-style PPRFs, and we expect our analysis to extend almost immediately to multi-instance DPFs. Using to the attack which we describe in Section 3, the concrete security of PCFs using N copies of a DPF with a tree of depth D is $2^\lambda/(N \cdot D)$. In many settings, this turns out to be a significant security loss: for example, using the PCF of [BCG+22] to generate 2^{30} degree-2 correlations requires $N = 664^2$ copies of a GGM tree of depth $\log_2(2^{30}/664)^2$ (in fact, a 2-dimensional GGM tree). With the collision attack, this translates to a concrete loss of 27 bits of security. Using our methodology to extend GGM PPRFs to the multi-instance setting, the security loss could be reduced to 7 bits, without any sacrifice on efficiency.

We also expect that our proof technique could be used to improve the parameters of other schemes. For example, the recent work of [BCdSG24] also takes advantage of AES using half-tree construction [GYW+23] to improve MPCitH signature schemes. The scheme of [BCdSG24] uses a direct construction of half-tree based on the circular collision-resistant hash function (CCR), and due to the limitation of the security level of CCR hash (to 128-bit blocks and key size) when using an AES-based instantiation, the size of tree leaves have been extended to $2\lambda = 256$ bits to reach 128 bits of security. We expect that the techniques developed in the concrete security analysis our our multi-instance PPRF using the H-coefficient technique can be applied to security proof of [BCdSG24] construction to get rid of the need to expand the size of the last layer.

B Python Script

Below, we provide the Python script used to compute the value of $k \geq |X| + \lambda$ using the bound on $|X|$ from Lemma 26. The code for listing all partitions of an integer into i parts was taken from Stackoverflow.[11]

```
import numpy as np
import math
from math import factorial
from math import gcd
from math import comb
from math import log

def phi(n):
    amount = 0
    for k in range(1, n + 1):
        if gcd(n, k) == 1:
            amount += 1
    return amount

def L(blocksize):
  num = 0
  for k in range(1,blocksize+1):
    if k%2 == 1:
      D = gcd(blocksize-k,k)
      val = 0
      for d in range(1, D+1):
        if math.gcd(d,D)==d:
```

[11] https://stackoverflow.com/questions/18503096/python-integer-partitioning-with-given-k-partitions

```
            val += phi(d)*factorial(int(blocksize/d))/(factorial(int((blocksize
                                    -k)/d))*factorial(int(k/d)))
        num += val
    return int(num/blocksize)

def part(n, k):
    def memoize(f):
        cache = [[[None] * n for j in range(k)] for i in range(n)]
        def wrapper(n, k, pre):
            if cache[n-1][k-1][pre-1] is None:
                cache[n-1][k-1][pre-1] = f(n, k, pre)
            return cache[n-1][k-1][pre-1]
        return wrapper
    @memoize
    def _part(n, k, pre):
        if n <= 0:
            return []
        if k == 1:
            if n <= pre:
                return [(n,)]
            return []
        ret = []
        for i in range(min(pre, n), 0, -1):
            ret += [(i,) + sub for sub in _part(n-i, k-1, i)]
        return ret
    return _part(n, k, n)

def max_numblocks(w,B):              # return the largest N such that u can
                                        have N distinct blocks yet generates
                                        less than B permutations
  N = 1
  numb = 1
  for j in range(w-N+2,w+1):
    numb *= j
  while numb <= B:
    N += 1
    numb = 1
    for j in range(w-N+2,w+1):
      numb *= j
  return N-1

def min_k(w,B):                   # returns the smallest k (above w/2, by
                                        symmetry) such that c = (w choose k)
                                        is less than or equal to B
  k = int(w/2)
  c = math.comb(w,k)
  while c > B:
    k += 1
    c = math.comb(w,k)
  return (c,k)

def gvbound(w,bs,B,b=1,secpar=128):
  K = w*bs
  l = L(bs)
  N = max_numblocks(w,B)
  res = 0
  for n in range(2,N+1):             # starts at 2 to avoid the empty
                                        partition later when we isolate the
                                        biggest element
    val = 0
```

```
  (c,k) = min_k(w,B)            # computes the smallest k (above w/2, by
                                   symmetry) such that c = (w choose k)
                                   is less than or equal to B
  for i in range(k, w-n+2):       # for all possible values i of the first
                                   term in the partition of w (from k to
                                   w-(N-1))
    partitions = part(w-i,n-1)      # stores all partitions of the integer
                                   w-i into N-1 parts
    fact = 1
    for j in range(i+1, w+1):
      fact *= j              # computes w!/i!, which is a common factor of the
                                    terms summed over all partitions of w
                                   -i
    for partition in partitions:
      term = fact
      for v in partition:
        term = term//factorial(v)    # computes w!/(i! * prod_v v!), where v
                                    goes over the elements of a partition
                                    of w-i
      if term <= B:            # checks that the term w!/(i! * prod_v v!) is
                                   below the bound B
        val += term            # val accumulates the terms of the sum accross
                                   all partitions of w-i, then accross
                                   all values of i
  res += factorial(n)*math.comb(l, n)*val         # res accumulates the
                                   result accross all possible numbers n
                                   of blocks, from 1 to N = max_numblocks
                                   (w,B)
dim = math.log((res+1)*(bs**w),2)      # dim is the base 2 logarithm of (K/w
                                   )^w * (the sum computed so far). /!\
                                   will need to add lambda later
                       # also adds 1 to the result because we excluded n=1
return dim + b*secpar
```