VOLE-in-the-head signatures from Subfield Bilinear Collisions

Janik Huth[1,2] and Antoine Joux[1]

[1] CISPA - Helmholtz Center for Information Security, Saarbrücken, Germany
`janik.huth@cispa.de, joux@cispa.de`
[2] Saarland University, Saarbrücken, Germany

Abstract. In this paper, we introduce a new method to construct a signature scheme based on the subfield bilinear collision problem published at Crypto 2024. We use techniques based on vector oblivious linear evaluation (VOLE) to significantly improve the running time and signature size of the scheme compared to the MPC-in-the-head version.

1 Introduction

The *Multi-Party Computation in the Head* (MPCitH) paradigm was first introduced in [18] as a new approach to design *Zero-Knowledge* (ZK) protocols. This approach leverages a multiparty protocol checking an NP-relation $\mathcal{R}(x, w)$ to construct a ZK protocol where a prover $\mathcal{P}$ convinces a verifier $\mathcal{V}$ that she knows a witness w for a given (public) value of x, without disclosing any information about w. In the MPCitH protocol, the prover emulates the execution of the underlying MPC protocol checking the relation being proven. The verifier $\mathcal{V}$ then checks parts of this execution in a way that allows him to verify the correctness of the relation while preserving the secrecy of the witness w. As other identification protocols, those based on MPCitH can be turned into signature schemes by using the Fiat-Shamir transform [14]. In recent years, a large number of MPCitH-based signature schemes have been proposed [12,13,5,3,11]. These schemes are quite efficient in terms of speed, but the corresponding signatures remain quite large compared to classical signature schemes such as [22,24,2]. In order to limit this problem and reduce the size of signatures, Baum et al. introduced an additional twist on MPCitH, called *VOLE-in-the-head* [4], which relies on *vector oblivious linear evaluation* (VOLE) and uses a non-interactive version of the SoftspokenOT technique [23]. This technique significantly reduces the signature size and, as a side bonus, also improves the running times of signature schemes, for example, see [8,10,6].

In this paper, we consider the *subfield bilinear collision* (SBC) problem proposed at Crypto 2024 [17] and develop a *VOLE-in-the-head* signature scheme based on this problem. In [17], it already appeared that the structure of the problem was highly suited to the MPCitH paradigm leading to a simple and quite compact signature. In the present paper, we show that the structure of the SBC problem also works very well in the context of *VOLE-in-the-head*. In fact, because of this, we do not need to use the full extent of the functionalities presented in [4]. Instead, we reconstruct a specific version of VOLEitH from scratch dedicated to the SBC problem in order to obtain a simplified scheme.

2 Preliminaries

2.1 Notations

We denote the target security level of our scheme by λ. For any positive integer $m \in \mathbb{Z}^+$, we denote the set $\{0, \ldots, m-1\}$ by $[m]$. For $m > 1$, we denote the set $\{1, \ldots, m-1\}$ by $[m]^*$.

Let S be a finite set. Then the notation $s \overset{\$}{\leftarrow} S$ indicates that s is sampled uniformly at random from S.

Let $\mathbb{F}$ be a finite field. We denote the polynomial ring over $\mathbb{F}$ with indeterminate X by $\mathbb{F}[X]$. We indicate vectors by using the arrow notation, e.g. $\vec{x}$. Coordinates of $\vec{x}$ are denoted x_i, with indices starting at 0.

We denote the dot product between two vectors $\vec{u} \in \mathbb{F}^n$ and $\vec{x} \in \mathbb{F}^n$ by $\vec{u} \cdot \vec{x} = \sum_{i=0}^{n-1} u_i x_i$. Let $\vec{a} \in \mathbb{F}^{n_1}$ and $\vec{b} \in \mathbb{F}^{n_2}$ be two vectors of dimensions n_1 and n_2 respectively. We denote the vector of dimension $n_1 + n_2$ resulting from the concatenation of the coordinates of $\vec{a}$ and $\vec{b}$ by $(\vec{a}, \vec{b})$.

Additionally, we place ourselves in the random oracle model and consider every hash function used in this paper as a random oracle. In particular, $\mathcal{H}$ is always modeled by a random oracle.

2.2 The SBC Problem

The *subfield bilinear collision* (SBC) problem was introduced in [17]. We recall it using the same notations as in [17]:

Definition 1. *Let q be a prime power and let k, n be positive integers.*

- *Problem instance: Two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, which are linearly independent over $\mathbb{F}_{q^k}$.*
- *Solution: Two vectors $\vec{x}, \vec{y} \in \left(\mathbb{F}_q\right)^n$, linearly independent over $\mathbb{F}_q$, that satisfy the relation:*

$$(\vec{u} \cdot \vec{x})\,(\vec{v} \cdot \vec{y}) = (\vec{u} \cdot \vec{y})\,(\vec{v} \cdot \vec{x})\,. \tag{1}$$

We denote such an instance of the SBC problem by $\mathrm{SBC}[\vec{u}, \vec{v}]$ *and write* $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$ *when the vectors $\vec{x}, \vec{y} \in \left(\mathbb{F}_q\right)^n$ are a solution of the instance specified by $\vec{u}$ and $\vec{v}$.*

Remark 1. In (1), we use the canonical embedding from $\mathbb{F}_q$ to $\mathbb{F}_{q^k}$. Therefore, all operations are well-defined. As mentioned in [17], it is simpler and preferable to use normalized solutions for the SBC problem of the form $\vec{x}' = (\vec{x}, 1, 0)$ and $\vec{y} = (\vec{y}', 0, 1)$ for $\vec{x}, \vec{y} \in \left(\mathbb{F}_q\right)^{n-2}$. Furthermore, [17] shows how to transform a regular solution into a normalized one using simple linear algebra. In the normalized case, we use the notation $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ to denote a solution.

Generating instances with a planted solution. In order to design identification and signature schemes from the SBC problem, we need the ability to efficiently create instances together with a solution. The following method from [17] solves this issue.

Given q, k and n, we can generate an instance of the SBC problem together with a normalized solution in the following way: Choose, uniformly at random two vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$ and let $\vec{x}' := (\vec{x}, 1, 0)$ and $\vec{y}' := (\vec{y}, 0, 1)$. Then randomly select the vector coordinates $\vec{u} \in \mathbb{F}_{q^k}^n$ and the $n-1$ coordinates $v_0, \ldots, v_{n-2} \in \mathbb{F}_{q^k}$ of $\vec{v}$. Finally, compute the missing coordinate v_{n-1} as

$$v_{n-1} = \frac{(\vec{u} \cdot \vec{y}') \left(\sum_{i=0}^{n-2} v_i x_i'\right) - (\vec{u} \cdot \vec{x}') \left(\sum_{i=0}^{n-2} v_i y_i'\right)}{y_n' \, (\vec{u} \cdot \vec{x}')}.$$

If the denominator $y_n' \, (\vec{u} \cdot \vec{x}')$ happens to be zero, the computation of v_n fails. This occurs with negligible probability. In this case, we can restart the selection of $\vec{u}$ and $\vec{v}$. Similarly, in the very unlikely case that $\vec{u}, \vec{v}$ are linearly dependent, we select new vectors instead. The construction guarantees that $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$.

Hardness of SBC. Following the heuristic analysis from [17], the NSBC problem is hard for $n \approx \frac{k}{2}$. Note that, as mentioned in [17], any advance concerning the SBC problem could be leveraged into a faster bilinear descent step for the computation of discrete logarithms in small characteristic finite fields [19].

To design our scheme, we follow the parameters suggested in [17] for a security level of $\lambda = 128$ bits, namely we use $q = 2, n = 130$ and $k = 257$.

2.3 The MPC-in-the-head paradigm

The construction of a ZK proof in [17] relies on the MPCitH paradigm introduced in [18]. We first define notations for basic secure multiparty computations which we use throughout this paper. We use additive sharings of finite field elements and finite field vectors. Let N be the number of parties. Then an *N-sharing* of a finite field element $x \in \mathbb{F}$ is an N-tuple

$$[\![x]\!] = \left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right)$$

such that

$$x = \sum_{i=0}^{N-1} x^{[\![i]\!]} \quad (\text{in } \mathbb{F}).$$

We call $x^{[\![i]\!]}$ the i-th *share* of x. Similarly, we use the notation

$$[\![\vec{x}]\!] = \left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![N-1]\!]}\right)$$

for a sharing of a finite field vector $\vec{x} \in \mathbb{F}^n$. In our setting, each party receives one of the N shares.

A sharing of a given value x is usually obtained by computing $N-1$ random values $x^{[\![0]\!]}, \ldots, x^{[\![N-2]\!]}$ and by setting $x^{[\![N-1]\!]} = x - \sum_{i=0}^{N-2} x^{[\![i]\!]} \mod |\mathbb{F}|$. In this paper, we often use a random oracle to derive the shares appearing in the protocol. In this situation, each share can be viewed as a random value, and summed together they would share a random value. Thus, to obtain a sharing of the desired value, we need an *offset* value δ_x, which is computed as

$$\delta_x = x - \sum_{i=0}^{N-1} x^{[\![i]\!]}.$$

We can then reconstruct x using the formula

$$x = \delta_x + \sum_{i=0}^{N-1} x^{[\![i]\!]}.$$

This could also be viewed as an (exact) sharing of x between $N+1$ parties. By introducing these offset values, we can use a random oracle to derive sharings if needed and can simplify the notation we use in our protocol description.

Hypercube technique. A prevalent tool to improve efficiency of MPCitH-based schemes is the hypercube technique introduced in [1]. Let $N = 2^D$ be the number of parties. This approach transforms one instance of the protocol with $N = 2^D$ parties into D instances of the protocol, each involving only 2 parties, while still having the same total soundness error as the original protocol. Let the j-th bit of the binary decomposition of an integer i be $B_j(i)$. The process of converting one protocol instance with $N = 2^D$ parties into D instances with 2 parties is referred to as *folding* of the shares.

Consider the sharing of an element x between N parties of the form

$$x = \sum_{i=0}^{N-1} x^{[\![i]\!]}.$$

For any fixed value $j \in [D]$, we see that

$$\begin{aligned} x &= \sum_{B_j(i)=0} x^{[\![i]\!]} + \sum_{B_j(i)=1} x^{[\![i]\!]} \\ &= {x_j}^{[\![0]\!]} + {x_j}^{[\![1]\!]} \end{aligned}$$

is a sharing of x between two parties. In the hypercube technique, D such protocols between two parties are run in parallel, each having a soundness error of $\frac{1}{2}$. We use the notation $\left({x_j}^{[\![0]\!]}, {x_j}^{[\![1]\!]}\right)_{j \in [D]} \leftarrow \mathsf{Folding}\left(x^{[\![0]\!]}, \ldots, x^{[\![N-1]\!]}\right)$ to denote this construction. For efficiency, we use the fast folding algorithm with running time $\mathcal{O}(N)$ described in [17, Section 6] in our implementation.

The key point of the hypercube technique is that the choice of one missing share among N is equivalent to the choice of one missing share in each derived binary sharing. Note that the correspondence between the missing positions simply comes from the binary decomposition of the missing index among N. Furthermore, the derived binary sharings are mutually independent.

2.4 Puncturable PRFs

To reduce communication cost, the N-sharings appearing in the construction are based on the use of puncturable pseudo-random functions (PPRFs). More specifically, we use the classical PPRF based on GGM trees [15], which is mentioned for example in [7,21]. Starting from a root seed, the idea is to build a binary tree with $N = 2^D$ leaves by recursively applying a length-doubling *pseudo random generator* (PRG) to obtain the left and right children of a node. To reveal all N leaves of such a binary tree except one leaf $i^* \in [N]$, the idea is to reveal all the nodes of the siblings of the path from the root seed to the leaf i^*. By using this method, all leaves except i^* can be reconstructed by communicating $\lceil \log_2(N) \rceil$ nodes instead of $N - 1$ leaves.

In our scheme, we use *correlated* GGM (or cGGM) trees, as introduced in [16]. The idea is to build a tree where the sum of all nodes on each level is preserved as the tree is being constructed. In order to do that, it suffices to modify the length-doubling PRG so that the sum of the label of the left and right descendants of any node is equal to the label of the node itself. Given the (salted) hash function $\mathcal{H}$ and any node T, this is easily achieved by setting the left child of T to $\mathcal{H}(T)$ and the right child of T to $T - \mathcal{H}(T)$. In the binary case $q = 2$, the right child is rewritten as $T \oplus \mathcal{H}(T)$ and we speak of a *XOR-preserving* GGM tree. Note that correlated GGM trees have a limitation. Namely, the size of the correlated value is limited by the size of the inner nodes of the tree, usually equal to λ bits. As a consequence, if the tree is used to share more than λ bits between the parties, it is necessary to compute and transmit offsets for the values beyond the first λ bits. Of course, this constraint of having offsets does not apply to purely random values shared between the parties.

In order to improve the running time of the scheme for $\lambda = 128$, we use a cGGM tree construction based on the AES block cipher instead of a salted hash function. This technique was introduced in [9] and takes full advantage of the AES instruction set available on many recent CPU architectures. In the standard GGM, we need two keys K_0 and K_1, the left child is set to $T \oplus \mathrm{AES}_{K_0}(T)$ and the right child is set to $T \oplus \mathrm{AES}_{K_1}(T)$. It is important to note that a direct application of AES would not lead to secure PPRF construction because decryption would allow one to climb back to the root node. In order to achieve a XOR preserving tree, we still use two keys and define the left node to be $\mathrm{AES}_{K_0}(T) \oplus \mathrm{AES}_{K_1}(T)$, which leads to a right node of $T \oplus \mathrm{AES}_{K_0}(T) \oplus \mathrm{AES}_{K_1}(T)$.

3 Identification protocol based on SBC

Consider an NSBC instance $\mathrm{NSBC}[\vec{u}, \vec{v}]$ for $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$. Based on the cGGM tree construction described in Section 2.4, we build an interactive Zero-Knowledge proof of knowledge of a solution $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. As previously explained, the normalized vectors $\vec{x}'$ and $\vec{y}'$ are written in the form $\vec{x}' = (\vec{x}, 1, 0)$ and $\vec{y}' = (\vec{y}, 0, 1)$ for $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$. For simplicity, we only consider the case $q = 2$ for this construction. In other words, the secret key of our scheme consists of the $2n - 4$ bits forming $\vec{x}$ and $\vec{y}$.

3.1 Correlating multiple cGGM trees

To design our signature scheme, we need multiple (different) sharings of the vectors $\vec{x}$ and $\vec{y}$ between 2 parties in the hypercube setting. More precisely, the number of binary sharings can never be lower than the desired security level λ. Furthermore, depending on the details of the scheme and of its security proof, a few extra sharings might be needed.

In particular, we need to be able to construct a large number of (independent) sharings of $\vec{x}$ between 2 parties. When doing that, it is essential to ensure that the same value of $\vec{x}$ is used everywhere. With a standard cGGM tree with N parties, we obtain $\log_2(N)$ such sharings. And, by construction, both the prover and verifier have the guarantee that all sharings are for the same vector $\vec{x}$. Unfortunately, N cannot be arbitrarily large, since it would make the computation costs too high. As a consequence, repetition is needed to obtain a large enough number of sharings. Using τ repetitions, we obtain a total of $\tau \cdot D$ binary sharings, and to ensure the desired security level, we require $\tau \cdot D \geq \lambda$.

In this section, we introduce a technique that allows us to guarantee the equality of the vector $\vec{x}$ across several distinct cGGM trees. This is achieved by a layered construction: first, an initial cGGM tree is built, followed by the hypercube folding, which is used to initialize a second level consisting of τ distinct cGGM trees.

First, we construct the initial cGGM tree, which we also call the pre-tree, that contains 2^τ leaves, where every level sums to $\vec{x}$, using a bitwise XOR on λ bits. To initialize this tree, we generate a random initial value $R_{\vec{x}}$ and set the two sibling elements on the second level of the tree as $R_{\vec{x}}$ and $R_{\vec{x}} \oplus \vec{x}$. The root node is unused and can remain unassigned. The XOR of these two nodes is $\vec{x}$ by design. From them, we compute the next levels of a XOR-preserving cGGM tree $T_{\vec{x}_{\mathrm{pre}}}$ with 2^τ leaves. We call this functionality the generation of the Pre-Tree and denote it by $T_{\vec{x}_{\mathrm{pre}}} \leftarrow \mathsf{cGGM}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$. Let $(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau - 1]\!]})$ be the leaves of $T_{\vec{x}_{\mathrm{pre}}}$.

Using the hypercube folding for these leaves, we obtain τ (pre) sharings of $\vec{x}$ between two parties of the form $[\![\vec{x}]\!]_j = \left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}\right)$ for $j \in [\tau]$ by using the function $\mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau - 1]\!]}\right)$. By construction, we have that $\vec{x}^{[\![0]\!]_j} \oplus \vec{x}^{[\![1]\!]_j} = \vec{x}$ for $j \in [\tau]$.

We can use each of these sharings to initialise a second level containing τ new correlated trees $T_{\vec{x}_j}$ for $j \in [\tau]$. Each of these trees can then be fully computed with the cGGM technique. We thus obtain τ trees $T_{\vec{x}_0}, \dots, T_{\vec{x}_{\tau-1}}$ with 2^D leaves each using the function $T_{\vec{x}_j} \leftarrow \mathsf{cGGM}(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}, 2^D)$ for each $j \in [\tau]$. The leaves of tree $T_{\vec{x}_j}$ are denoted by $\left(\vec{x}_j^{[\![0]\!]}, \dots, \vec{x}_j^{[\![2^D-1]\!]}\right)$. By using the hypercube folding on the leaves of each of these trees again, we obtain D sharings of $\vec{x}$ between 2 parties for each of the τ trees. We denote this by $\left(\vec{x}_j^{[\![0]\!]_i}, \vec{x}_j^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow$ $\mathsf{Folding}\left(\vec{x}_j^{[\![0]\!]}, \dots, \vec{x}_j^{[\![2^D-1]\!]}\right)$ for $j \in [\tau]$.

Since we used XOR preserving cGGM trees on both layers of our construction, this is guaranteed to yield a total of $\tau \cdot D$ hypercube sharings of the same value $\vec{x}$.

To simplify the notation, we renumber the $\tau \cdot D$ sharings obtained in this way and simply write them as:

$$\vec{x}_m^{[\![0]\!]} \oplus \vec{x}_m^{[\![1]\!]} = \vec{x} \quad \text{for each } m \in [\tau \cdot D].$$

In addition, by using a pseudo-random function to expand the leaves of each tree $T_{\vec{x}_j}$ into more than λ bits, it is easy to produce shares for additional values in each tree. Note, however, that these extra values are not correlated. As a consequence, they might require offsets and extra precautions to prove equality across trees.

Punctured keys for the correlated cGGM family. When giving out the punctured key for a correlated cGGM family, we need to do it in a way that guarantees to the verifier that the construction was properly used. Thus, it is not sufficient to independently give a punctured key for each tree from the second level. Indeed, such an opening would be possible even with completely independent trees. Thus, equality across trees would not be guaranteed. Instead, we first give a punctured key for the pre-tree. From this punctured key, the verifier can recover all the leaves of the pre-tree but one. Thus, after hypercube folding he learns exactly one node on the second level of each of the τ trees. Note that these nodes would normally be part of the punctured keys of the second level trees, so this technique allows us to remove one element from each of the second level punctured keys. Therefore, proceeding in this manner does not modify the overall size of the global punctured key. Indeed, we remove τ elements and replace them by the punctured key of the pre-tree, which contains τ nodes. To locate the missing position that is need in the pre-tree, it suffices to assemble the high order bits of the missing nodes of each second-level tree into a τ-bit number.

3.2 From sharings to Vector Oblivious Linear Evaluation (VOLE)

In our use of the trees described in the above section, the prover always knows the complete trees, while the verifier only lacks one leaf of every tree (including the pre-tree). At the level of the sharings, it means that for every $m \in [\tau \cdot D]$, the verifier knows only one value $\vec{x}_m^{[\![b_m]\!]}$ (out of two) for a bit b_m of its choice, while the prover knows both.

It is equivalent and convenient to say that the prover creates an affine vectorial function:

$$f_m(b_m) = \vec{x}_m^{[\![0]\!]} \oplus b_m \cdot \vec{x}$$

and that the verifier chooses b_m and learns $f_m(b_m)$.

With this view in mind, we can now pick a series of random coefficients α_m in the large field $\mathbb{F}_{2^k}$ and consider the function:

$$\vec{F} = \sum_{m=0}^{\tau D-1} \alpha_m f_m(b_m) = \sum_{m=0}^{\tau D-1} \alpha_m \vec{x}_m^{[\![0]\!]} + \left[\sum_{m=0}^{\tau D-1} \alpha_m b_m\right] \cdot \vec{x}.$$

Remark that $\vec{F}$ is an affine function known to the prover and that the verifier chooses the evaluation point $\Delta = \sum_{m=0}^{\tau D-1} \alpha_m b_m$ and learns the value $\vec{F}(\Delta)$ obtained by evaluating $\vec{F}$ at this point.

In principle, the coefficients α_m could be pre-specified as part of the scheme. However, to avoid casting suspicion on the choice of coefficients, we feel that it is preferable to have them chosen at random by the honest verifier.

3.3 A simple but inefficient construction

As already mentioned, cGGM trees with internal nodes of size λ bits are limited and can only produce correlated values up to λ bits. Unfortunately, the secret key of the SBC problem contains a total of 2λ bits, namely $\vec{x}$ and $\vec{y}$. To illustrate our construction in a simple but inefficient way, let us use the construction of Section 3.1 twice, in order to independently get $\tau \cdot D$ sharings of each of $\vec{x}$ and $\vec{y}$. We write them as:

$$\vec{x}_m^{[\![0]\!]} \oplus \vec{x}_m^{[\![1]\!]} = \vec{x} \quad \text{and} \quad \vec{y}_m^{[\![0]\!]} \oplus \vec{y}_m^{[\![1]\!]} = \vec{y} \quad \text{for each } m \in [\tau \cdot D]. \tag{2}$$

Proceeding as in Section 3.2, we use these two families of correlated trees to define two vectorial functions $\vec{F}_x, \vec{F}_y : \mathbb{F}_{2^k} \to (\mathbb{F}_{2^k})^{n-2}$:

$$\vec{F}_x(\Delta) = \vec{A}_{F_x} + \Delta \cdot \vec{x},$$
$$\vec{F}_y(\Delta) = \vec{A}_{F_y} + \Delta \cdot \vec{y},$$

whose constant coefficients are given by

$$\vec{A}_{F_x} = \vec{F}_x(0) = \sum_{m=0}^{\tau \cdot D-1} \alpha_m \, \vec{x}_m^{[\![0]\!]} \quad \text{and} \quad \vec{A}_{F_y} = \vec{F}_y(0) = \sum_{m=0}^{\tau \cdot D-1} \alpha_m \, \vec{y}_m^{[\![0]\!]}.$$

Note that, since we want to evaluate both functions at the same Δ, the missing tree leaves in both families are located in the same manner.

Before learning the evaluation point Δ, the prover computes the polynomial Φ in Δ defined as:

$$\Phi(\Delta) := \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta) \\ 1 \\ 0 \end{pmatrix}\right) \left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta) \\ 0 \\ 1 \end{pmatrix}\right) \left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta) \\ 1 \\ 0 \end{pmatrix}\right)$$

$$
\begin{aligned}
&= \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)-\left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\\
&\quad+\Bigg[\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)+\left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)\\
&\quad-\left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)-\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)\Bigg]\Delta\\
&\quad+\left[\left(\vec{u}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)-\left(\vec{u}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)\right]\Delta^2.
\end{aligned}
$$

Remark 2. Note that Φ is of the form

$$\Phi(\Delta) = A + B\Delta$$

if and only if $(\vec{x}', \vec{y}') \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ for $\vec{x}' = (\vec{x}, 1, 0)$, $\vec{y}' = (\vec{y}, 0, 1)$. The coefficients $A, B \in \mathbb{F}_{2^k}$ are given by

$$
\begin{aligned}
A &= \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)-\left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right),\\
B &= \left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)+\left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)\\
&\quad-\left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right)-\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right).
\end{aligned}
$$

We use this property of Φ in our ZK protocol.

The prover sends the coefficients $A, B \in \mathbb{F}_{2^k}$ to the verifier. The verifier responds by sending back random challenges $i_0^*, \ldots, i_{\tau-1}^* \in [2^D]$, one for each tree. The prover returns the PPRF keys $\mathcal{K}_{i_j^*}$ for the trees $T_{\vec{x}_j}$ and $T_{\vec{y}_j}$ for each $j \in [\tau]$ to the verifier.

As explained in Section 3.2, this corresponds to the verifier choosing an evaluation point

$$\Delta^* := \sum_{m=0}^{\tau\cdot D-1} b_m \alpha_m,$$

and obtaining the values $\vec{F}_x(\Delta^*)$ and $\vec{F}_y(\Delta^*)$. Equipped with this knowledge, the verifier can compute $\Phi(\Delta^*)$ as

$$\Phi(\Delta^*) = \left(\vec{u}\cdot\begin{pmatrix}\vec{F}_x(\Delta^*)\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_y(\Delta^*)\\0\\1\end{pmatrix}\right)-\left(\vec{u}\cdot\begin{pmatrix}\vec{F}_y(\Delta^*)\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{F}_x(\Delta^*)\\1\\0\end{pmatrix}\right).$$

He then checks whether $\Phi(\Delta^*) = A + B\Delta^*$ and accepts or rejects accordingly.

Remark 3. The protocol accepts a wrong instance of vectors $(\vec{x}', \vec{y}') \notin \mathrm{NSBC}[\vec{u}, \vec{v}]$ if Δ^* is a root of the polynomial

$$\Phi(\Delta) - B\Delta - A.$$

For $(\vec{x}', \vec{y}') \notin \mathrm{NSBC}[\vec{u}, \vec{v}]$, $\Phi(\Delta)$ is a quadratic polynomial in Δ. Therefore, $\Phi(\Delta) - B\Delta - A$ has at most two roots. Since the evaluation point is chosen in a set of cardinality $2^{\tau D}$, the probability of getting a false positive is bounded by $2^{1-\tau D}$.

3.4 Consistency checks

In Section 3.3, we use two independent cGGM trees. One to derive $\tau\, D$ sharings of $\vec{x}$ and one to derive $\tau\, D$ sharings of $\vec{y}$. Despite its simplicity, this approach has a very high cost since the need for opening two puncturable PRF families doubles the amount of data. In this section, we address the problem of implementing the same basic idea while using a single family of cGGM trees, based on $\vec{x}$. This leads to a significant decrease in the required communication cost for the scheme but requires additional techniques.

As mentioned in Section 3.1, we can use the tree family provided to share $\vec{x}$ to create sharings of extra values, simply by applying a PRF to extend the size of individual leaves. However, this creates two problems. First, every tree induces sharings of random values, thus offsets are required to set the sharings to the desired value. As a consequence, each of the τ cGGM trees created to share $\vec{x}$ can be made to share $\vec{y}$ by providing a corresponding offset value. Since the communication costs for these offsets are much lower than for an extra family of cGGM trees, this is worthwhile in terms of signature size.

The second problem is more subtle and harder to fix. By providing an offset for $\vec{y}$, we can make sure that each of the τ cGGM trees provides D sharings of a value $\vec{y}_i$. However, nothing guarantees, from the verifier's viewpoint, that all these values are identical. This could be exploited by a cheating prover to boost his cheating probability. To remedy this, we need to add a consistency check that lets the verifier check that all the $\vec{y}_i$ for $i \in [\tau]$ are really equal. This idea of performing a (probabilistic) consistency check is known as the SoftSpokenOT technique [23]. In our case, it can be somewhat simplified and we now describe our version in detail.

For the purpose of realizing the equality testing, we extend each leaf in the family of cGGM trees created to share $\vec{x}$ in order to produce additional sharings of two additional elements: the second secret value $\vec{y}$ and an extra element z from $\mathbb{F}_{2^k}$. Since $\vec{y}$ is fixed, one offset is required for each of the τ trees. For z, since a random element is requested, it is tempting to assume that no offsets are needed. However, to prove equality between sharings of $\vec{y}$, we also need the sharings of z to be equal across the trees. As a consequence, we can only skip the offset corresponding to z for the first of the cGGM trees in the family.

More precisely, from each leaf $\vec{x}_j^{[\![i]\!]} \in (\mathbb{F}_2)^{n-2}$, the prover derives the shares $\vec{y}_j^{[\![i]\!]} \in (\mathbb{F}_2)^{n-2}$ and $z_j{}^{[\![i]\!]} \in \mathbb{F}_{2^k}$ for each $(i,j) \in [2^D] \times [\tau]$, using the random oracle $\mathcal{H}$. To commit to the family of τ subtrees, he sends the offset values $\delta_{\vec{y}_j} \in (\mathbb{F}_2)^{n-2}$ and $\delta_{z_j} \in \mathbb{F}_{2^k}$ such that

$$\vec{y}_j = \sum_{i=0}^{2^D-1} \vec{y}_j^{[\![i]\!]} + \delta_{\vec{y}_j} \text{ and } z_j = \sum_{i=0}^{2^D-1} z_j{}^{[\![i]\!]} + \delta_{z_j} \text{ for } j \in [\tau], \tag{3}$$

to the verifier. Note that we set δ_{z_0} to 0 and do not send this value.

After the prover has committed to the cGGM trees, the verifier computes a random vector $\vec{\mu} \in (\mathbb{F}_{2^k})^{n-2}$ and sends it to the prover. The goal of $\vec{\mu}$ is to provide an efficient check that the pairs $(\vec{y}_j, z_j)$ are equal among the rounds. Since $\vec{\mu}$ is selected randomly and not under the control of the prover, it suffices to check that $z_j + \vec{\mu} \cdot \vec{y}_j$ is constant across the rounds.

Remark 4. Note that the goal of z is to act as a one-time-pad and hide the value $\vec{\mu} \cdot \vec{y}$ which would, if revealed, leak the secret $\vec{y}$.

This equality check relies on the following lemma.

Lemma 1. *Let $\vec{y}, \vec{y'} \in (\mathbb{F}_2)^{n-2}$ and $z, z' \in \mathbb{F}_{2^k}$ with $(\vec{y}, z) \neq (\vec{y'}, z')$ and let $\vec{\mu} \xleftarrow{\$} (\mathbb{F}_{2^k})^{n-2}$. Then $\Pr[z + \vec{\mu} \cdot \vec{y} = z' + \vec{\mu} \cdot \vec{y'}] \leq 2^{-k}$, where the probability is on the choice of $\vec{\mu}$.*

Proof. Since $(\vec{y}, z) \neq (\vec{y'}, z')$, we have:

$$(\vec{y} \oplus \vec{y'}, z \oplus z') \neq (\vec{0}, 0).$$

To analyze the probability $\Pr[z + \vec{\mu} \cdot \vec{y} = z' + \vec{\mu} \cdot \vec{y'}]$, we consider two cases:

1. Assume $\vec{y} \oplus \vec{y'} = \delta_{\vec{y}} \neq \vec{0}$. In this situation, the probabilistic event occurs when $\delta_{\vec{y}} \cdot \vec{\mu} = z \oplus z'$. Equivalently, $\vec{\mu}$ belongs to a given affine hyperplane orthogonal to $\delta_{\vec{y}}$, which occurs with (conditional) probability 2^{-k}.
2. Assume, $\vec{y} = \vec{y'}$ and $z \neq z'$, thus $z + \vec{\mu} \cdot \vec{y} \neq z' + \vec{\mu} \cdot \vec{y'}$. In this case, the (conditional) probability is 0.

The statement follows by combining both cases. □

We now go back to the construction of the VOLE for $\vec{y}$ and z. A priori, the verifier is not sure that $\vec{y}$ and z are equal in all trees, and the construction from Section 3.2 cannot be used without precaution. However, all the binary sharings derived from an individual tree correspond to equal values $(\vec{y}_j, z_j)$, where $j \in [\tau]$ denotes the index of the tree in its family. Thus, the prover can compute a VOLE instance (with smaller evaluation domain) for $(\vec{y}_j, z_j)$ for each $j \in [\tau]$. More precisely, after folding, each tree provides D binary sharings:

$$\left(\vec{y}_j^{[\![0]\!]_i}, z_j{}^{[\![0]\!]_i}\right) \oplus \left(\vec{y}_j^{[\![1]\!]_i}, z_j{}^{[\![1]\!]_i}\right) \oplus (\delta_{\vec{y}_j}, \delta_{z_j}) = (\vec{y}_j, z_j).$$

As in Section 3.3, for each value $(i,j) \in [D] \times [\tau]$ the prover defines a linear polynomial:

$$f_{i,j}(b) = \left(\vec{y}_j^{[\![b]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + b \cdot (\vec{y}_j, z_j) \, .$$

We have grouped $\vec{y}_j$ and z_j in the above polynomial for compactness. However, by abuse of notation and for the sake of convenience, we might refer to it as a polynomial in $\vec{y}_j$ or z_j alone. We continue as in Section 3.3, and combine together the D polynomials in $\vec{y}_j$ and z_j. This defines a polynomial:

$$\vec{F}^{(j)}_{y,z}(\Delta_j) = \sum_{i=0}^{D-1} \alpha_{D\,j+i}\, f_{i,j}(b_{i,j}) = \sum_{i=0}^{D-1} \alpha_{D\,j+i} \left(\vec{y}_j^{[\![b]\!]_i}, {z_j}^{[\![0]\!]_i}\right) + \Delta_j \cdot (\vec{y}_j, z_j) \, ,$$

where $\Delta_j = \sum_{i=0}^{D-1} \alpha_{D\,j+i}\, b_{i,j}$. As before, we can write this as

$$\vec{F}^{(j)}_{y,z}(\Delta_j) = \vec{A}^{(j)}_{F_{y,z}} + \Delta_j \cdot (\vec{y}_j, z_j), \tag{4}$$

where the coefficient $\vec{A}^{(j)}_{F_{y,z}} = \vec{F}^{(j)}_{y,z}(0)$ is given by the evaluation at 0.

To implement the test provided by Lemma 1, we combine $\vec{\mu}$ with Equation (4) and find:

$$\begin{aligned} \vec{F}^{(j)}_{y,z}(\Delta_j) \cdot (\vec{\mu}, 1) &= \vec{A}^{(j)}_{F_{y,z}} \cdot (\vec{\mu}, 1) + \Delta_j (\vec{y}_j, z_j) \cdot (\vec{\mu}, 1) \\ &= \vec{A}^{(j)}_{F_{y,z}} \cdot (\vec{\mu}, 1) + \Delta_j [z_j + \vec{y}_j \cdot \vec{\mu}]. \end{aligned}$$

To understand the goal of the protocol, let us first **assume** that $\vec{y}_j$ and z_j are **equal across trees**. Then the coefficient in front of Δ_j is the same for all values of j. As a consequence, we ask that the prover computes and transmit this coefficient $b = z + \vec{y} \cdot \vec{\mu} \in \mathbb{F}_{2^k}$, where $\vec{y}$ and z denote the (putative) common values across tress. The prover also computes and transmits the coefficients $a^{(j)} = \vec{A}^{(j)}_{F_{y,z}} \cdot (\vec{\mu}, 1) \in \mathbb{F}_{2^k}$ for all $j \in [\tau]$.

As usual, the verifier sends the challenges $i^*_j \in [2^D]$ for each subtree T_j and obtains the corresponding punctured key from the prover, thus learning the evaluations $\vec{F}^{(j)}_{y,z}(\Delta^*_j)$ for each of the τ trees. Let us remark that if $\vec{y}_j$ and z_j are equal across trees, then we can add these evaluations and obtain:

$$\vec{F}_{y,z}(\Delta^*) = \sum_{j=0}^{\tau-1} \vec{F}^{(j)}_{y,z}(\Delta^*_j) = \vec{F}_{y,z}(0) + \Delta^* \, (\vec{y}, z) \quad , \text{where } \Delta^* = \sum_{j=0}^{\tau-1} \Delta^*_j .$$

In particular, the verifier learns $\vec{F}_y(\Delta^*)$ and also $\vec{F}_x(\Delta^*)$ (from the correlated part of the family of trees).

Thus, as in the simplified version of Section 3.3, the verifier computes

$$\Phi(\Delta^*) = \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right) \left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) \left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right) ,$$

checks if it is equal to $A + B\Delta^*$ and accepts or rejects accordingly.

If we **no longer assume equality across trees**, the verifier has to perform a little extra work. Namely, after obtaining each partial evaluation $\vec{F}^{(j)}_{y,z}(\Delta^*_j)$, he computes $\vec{F}^{(j)}_{y,z}(\Delta^*_j) \cdot (\vec{\mu}, 1)$ and checks that the value is equal to $a^{(j)} + b\Delta^*_j$. If any of these checks fail, the verifier rejects. If all checks succeed, he continues with the computation described in the case of equality, computing and checking $\Phi(\Delta^*)$. The following theorem states that this provides a sound protocol.

Theorem 1. *The soundness error of the NSBC identification scheme described above using $N = 2^D$ parties and τ rounds is bounded by $\tau 2^{-k} + 2^{1-\tau \cdot D}$.*

Proof. Consider a prover who does not share the same value for $(\vec{y}, z)$ in every round. The values $(\vec{y}_i, z_i)$ can be partitioned into at most τ distinct subsets. In particular, we have that

$$\{(\vec{y}_0, z_0), \ldots, (\vec{y}_{\tau-1}, z_{\tau-1})\} = S^{(0)} \dot{\cup} \ldots \dot{\cup} S^{(m-1)}, \tag{5}$$

where $m \leq \tau$. For any fixed index $j \in [m]$, we have that $(\vec{y}_{i_0}, z_{i_0}) = (\vec{y}_{i_1}, z_{i_1})$ for all $(\vec{y}_{i_0}, z_{i_0}), (\vec{y}_{i_1}, z_{i_1}) \in S^{(j)}$ by definition. Moreover, for any $j_0, j_1 \in [m]$ with $j_0 \neq j_1$, $S^{(j_0)} \cap S^{(j_1)} = \emptyset$.

From Lemma 1, we know that

$$\Pr[\exists i \neq j | (\vec{y}_i, z_i) \in S^{(i)}, (\vec{y}_j, z_j) \in S^{(j)} \text{ and } z_i + \vec{y}_i \cdot \vec{\mu} = z_j + \vec{y}_j \cdot \vec{\mu}] \leq \tau 2^{-k}.$$

This *bad event* provides the term $\tau 2^{-k}$ in the soundness error.

We now assume that the bad event has not occurred and without loss of generality, we consider that the reference value is $(\vec{y}_0, z_0) \in S^{(0)}$ and that during the protocol the prover sends the value $b = z_0 + \vec{y}_0 \cdot \vec{\mu}$. Since we excluded the above bad event, we know that $z_j + \vec{y}_j \cdot \vec{\mu} \neq b$ for all $(\vec{y}_j, z_j) \in S^{(i)}$, where $i \in \{1, \ldots, m-1\}$.

Consider a subtree T_j for $j \in [\tau]$ with $(a^{(j)}, b^{(j)}) \neq (a^{(j)}, b)$. For this index, we see that $a^{(j)} + b\Delta_j = z_j + \vec{\mu} \cdot \vec{F}^{(j)}_y(\Delta_j)$ for at most one value Δ_j. Indeed, the affine function $a^{(j)} + b\Delta_j - z_j - \vec{\mu} \cdot \vec{F}^{(j)}_y(\Delta_j)$ is non zero and thus has at most one root.

In order for the verifier to validate the equality testing for this index j, a cheating prover needs to guess the evaluation point Δ^*_j in advance and provide a corresponding $a^{(j)}$. We denote the subset of $[\tau]$ containing the indices j with $z_j + \vec{y}_j \cdot \vec{\mu} \neq b$ as I_{bad}. In particular,

$$I_{\text{bad}} \coloneqq \{j \in [\tau] | z_j + \vec{y}_j \cdot \vec{\mu} \neq b\}.$$

Since each set of possible evaluation points for the partial VOLE evaluation of any single tree contains 2^D points, the probability for the prover to correctly guess all evaluation points for I_{bad} is:

$$\Pr[\text{Prover guesses all } \Delta_j \text{ for } j \in I_{\text{bad}}] = 2^{-D \cdot |I_{\text{bad}}|}.$$

If the prover guesses correctly, we can treat all the Δ_j for $j \in I_{\text{bad}}$. As a consequence, the linear polynomials that are constructed in this situation only depend on the remaining indices $I_{\text{good}} \coloneqq [\tau] \setminus I_{\text{bad}}$. More precisely, everything can be expressed in terms of the variable:

$$\Delta_g = \sum_{j \in I_{\text{good}}} \Delta_j.$$

In particular, we can view Φ as a polynomial in Δ_g. Furthermore, we can see that $\Phi(\Delta_g)$ is an affine polynomial if and only if $(\vec{x}', \vec{y}'_g) \in \text{NSBC}[\vec{u}, \vec{v}]$ for $\vec{x}' = (\vec{x}, 1, 0), \vec{y}'_g = (\vec{y}_g, 0, 1)$.

As a consequence, for a cheating prover that does not know a solution to the NSBC instance, we know that $\Phi(\Delta_g)$ is a non-zero polynomial of degree 2. Thus, the linearity check on Φ can only work at two exceptional points (coming from a quadratic equation). Thus, the probability that this test succeeds for a cheating prover is upper bounded by $\frac{2}{2^{D \cdot |I_{\text{good}}|}} = 2^{1-D\cdot|I_{\text{good}}|}$. Since $\tau = |I_{\text{good}}| + |I_{\text{bad}}|$ and since the equality test for I_{bad} and the linearity test for I_{good} are independent, the overall success probability of both is upper bounded by:

$$2^{-D\cdot|I_{\text{bad}}|} \cdot 2^{1-D\cdot|I_{\text{good}}|} = 2^{1-\tau\cdot D}.$$

Adding this to the probability of the bad event described earlier, we conclude that the soundness error of the identification scheme is bounded by:

$$\tau 2^{-k} + 2^{1-\tau\cdot D}.$$

□

4 Detailed description of the identification scheme

For completeness, we now provide a detailed description of our five-round identification protocol between the prover $\mathcal{P}$ and the honest verifier $\mathcal{V}$. This protocol uses six algorithmic sub-routines, executed in alternation by the prover and the verifier. These algorithms are: Commitment, First Challenge, First Response, Second Challenge, Second Response, and Verification.

The main cryptographic tool that we use is the correlated cGGM trees family from Section 3.1. In this section, we base the construction on a random oracle $\mathcal{H}$ with output size λ. Following the state of the art, we use a seed and also do full domain separation, providing a tree number and a node position to derive the left child of each node using $\mathcal{H}$. The right child is obtained by XORing the left child and the parent. At the level of the leaf, we also use the random oracle (with proper domain separation) to derive additional shares for $\vec{y}$ and z. With our choice of security parameters, we need to create $3\lambda + 1$ additional bits for each leaf.

In the algorithms, we use the notations for the cGGM function introduced in Section 2.4 and the $\mathsf{Folding}$ function described in Section 2.3.

The protocol for the Commitment is given in Figure 1, the First Challenge is in Figure 2, the First Response in Figure 3. The Second Challenge and Second Response are given in Figure 4 and Figure 5. Finally, the Verification protocol is described in Figure 6.

```
Commitment

1. Sample $R_{\vec{x}} \xleftarrow{\$} (\mathbb{F}_2)^{n-2}$ and a random seed for the cGGM trees derivation
2. Compute Pre-Tree: $\left(\vec{x}^{\llbracket 0 \rrbracket}, \dots, \vec{x}^{\llbracket 2^\tau - 1 \rrbracket}\right) \leftarrow \mathsf{cGGM}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$
3. Fold Pre-tree: $\left(\vec{x}^{\llbracket 0 \rrbracket_j}, \vec{x}^{\llbracket 1 \rrbracket_j}\right)_{j \in [\tau]} \leftarrow \mathsf{Folding}\left(\vec{x}^{\llbracket 0 \rrbracket}, \dots, \vec{x}^{\llbracket 2^\tau - 1 \rrbracket}\right)$
4. For $j \in [\tau]$:
   – Get Tree $T_j$: $\left(\vec{x}_j^{\llbracket 0 \rrbracket}, \dots, \vec{x}_j^{\llbracket 2^D - 1 \rrbracket}\right) \leftarrow \mathsf{cGGM}\left(\vec{x}^{\llbracket 0 \rrbracket_j}, \vec{x}^{\llbracket 1 \rrbracket_j}, 2^D\right)$
   – Expand $T_j$ and get $\left(\vec{y}_j^{\llbracket 0 \rrbracket}, \dots, \vec{y}_j^{\llbracket 2^D - 1 \rrbracket}\right)$ and $\left(z_j{}^{\llbracket 0 \rrbracket}, \dots, z_j{}^{\llbracket 2^D - 1 \rrbracket}\right)$
   – Fold Tree $T_j$:
      • $\left(\vec{x}_j^{\llbracket 0 \rrbracket_i}, \vec{x}_j^{\llbracket 1 \rrbracket_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{x}_j^{\llbracket 0 \rrbracket}, \dots, \vec{x}_j^{\llbracket 2^D - 1 \rrbracket}\right)$
      • $\left(\vec{y}_j^{\llbracket 0 \rrbracket_i}, \vec{y}_j^{\llbracket 1 \rrbracket_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{y}_j^{\llbracket 0 \rrbracket}, \dots, \vec{y}_j^{\llbracket 2^D - 1 \rrbracket}\right)$
      • $\left(z_j{}^{\llbracket 0 \rrbracket_i}, z_j{}^{\llbracket 1 \rrbracket_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(z_j{}^{\llbracket 0 \rrbracket}, \dots, z_j{}^{\llbracket 2^D - 1 \rrbracket}\right)$
   – Let $z \leftarrow z_0{}^{\llbracket 0 \rrbracket_0} \oplus z_0{}^{\llbracket 1 \rrbracket_0}$
   – Compute offsets:
      • $\delta_{\vec{y}_j} \leftarrow \vec{y} \oplus \vec{y}_j^{\llbracket 0 \rrbracket_0} \oplus \vec{y}_j^{\llbracket 1 \rrbracket_0}$
      • $\delta_{z_j} \leftarrow z \oplus z_j{}^{\llbracket 0 \rrbracket_0} \oplus z_j{}^{\llbracket 1 \rrbracket_0}$
5. Send $\left(\delta_{\vec{y}_j}\right)_{j \in [\tau]}, \left(\delta_{z_j}\right)_{j \in [\tau]^*}$ and seed to the verifier
```

Fig. 1. Commitment algorithm of the identification protocol for the NSBC problem

```
Challenge 1

1. Sample $\vec{\mu} \xleftarrow{\$} (\mathbb{F}_{2^k})^{n-2}$
2. For $m \in [\tau \cdot D]$: Sample $\alpha_m \xleftarrow{\$} \mathbb{F}_{2^k}$
3. Send $\vec{\mu}, (\alpha_i)_{i \in [\tau \cdot D]}$ to the prover
```

Fig. 2. First Challenge algorithm of the identification protocol for the NSBC problem

Response 1

```
1. $\vec{A}_{F_x} \leftarrow \sum_{j=0}^{\tau-1}\sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{x}_j^{[\![0]\!]_i}\alpha_i$
2. $\vec{A}_{F_y} \leftarrow \sum_{j=0}^{\tau-1}\sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i}\alpha_i$
3. $A \leftarrow \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right) - \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)$
4. $B \leftarrow \left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right) + \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)$
   $- \left(\vec{u}\cdot\begin{pmatrix}\vec{A}_{F_y}\\0\\1\end{pmatrix}\right)\left(\vec{v}\cdot\begin{pmatrix}\vec{x}\\1\\0\end{pmatrix}\right) - \left(\vec{v}\cdot\begin{pmatrix}\vec{A}_{F_x}\\1\\0\end{pmatrix}\right)\left(\vec{u}\cdot\begin{pmatrix}\vec{y}\\0\\1\end{pmatrix}\right)$
5. Compute $b \leftarrow z + \vec{y}\cdot\vec{\mu}$
6. For $j \in [\tau]$:
   – Compute $\vec{A}_{F_y}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \vec{y}_j^{[\![0]\!]_i}\alpha_i$ and $\vec{A}_{F_z}^{(j)} \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} z_j{}^{[\![0]\!]_i}\alpha_i$
   – Compute $a^{(j)} \leftarrow \vec{A}_{F_z}^{(j)} + \vec{A}_{F_y}^{(j)}\cdot\vec{\mu}$
7. Send $A, B, \left(a^{(j)}\right)_{j\in[\tau]}$ and $b$ to the verifier
```

Fig. 3. First Response algorithm of the identification protocol for the NSBC problem

Challenge 2

```
1. Sample $\left(i_j^*\right)_{j\in[\tau]} \xleftarrow{\$} [2^D]^\tau$ and send it to the prover
```

Fig. 4. Second Challenge algorithm of the identification protocol for the NSBC problem

Response 2

```
1. Compute the Pre-Challenge $i_p^* \leftarrow \sum_{j=0}^{\tau-1} \cdot\lfloor i_j^*/2^{D-1}\rfloor 2^j$
2. Compute the Pre-Tree PPRF key $K_{i_p^*}$
3. Compute the (truncated) PPRF key $K_{i_0^*,\ldots,i_{\tau-1}^*}$
4. Send $\left(K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}\right)$ to the verifier
```

Fig. 5. Second Response algorithm of the identification protocol for the NSBC problem

```
Verification

 1. Recompute and fold all trees (with one missing leaf each)
 2. For $(i,j) \in [D] \times [\tau]$, let $b_{i,j}$ denote the known position of the respective
    hypercube binary sharing
 3. $\delta_{z_0} \leftarrow 0$
 4. For $j \in [\tau]$:
    – Compute $\Delta_j^* \leftarrow \sum_{i=j \cdot D}^{(j+1)\cdot D-1} b_{i,j}\alpha_i$
    – Compute $\vec{F}_y^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j \cdot D}^{(j+1)\cdot D-1} \alpha_i \left(\vec{y}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{\vec{y}_j}\right)$
    – Compute $Z^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j \cdot D}^{(j+1)\cdot D-1} \alpha_i \left(z_j{}^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{z_j}\right)$
    – $a^{(j)'} \leftarrow Z^{(j)}\left(\Delta_j^*\right) \oplus \vec{F}_y^{(j)}(\Delta_j^*) \cdot \vec{\mu} \oplus b\Delta_j^*$
 5. $\Delta^* \leftarrow \sum_{j=0}^{\tau-1} \Delta_j^*$
 6. $\vec{F}_x(\Delta^*) \leftarrow \sum_{m=0}^{\tau \cdot D-1} \vec{x}_m^{[\![b_m]\!]}\alpha_m$
 7. $\vec{F}_y(\Delta^*) \leftarrow \sum_{j=0}^{\tau-1} \vec{F}_y^{(j)}(\Delta_j^*)$
 8. Compute $\Phi(\Delta^*)$ as
    $\left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)$
 9. $A' \leftarrow \Phi(\Delta^*) - B\Delta^*$
10. If $A = A'$ and $(a^{(j)})_{j\in[\tau]} = (a^{(j)'})_{j\in[\tau]}$ output ACCEPT, otherwise output REJECT
```

Fig. 6. Verification algorithm of the identification protocol for the NSBC problem

5 Signature scheme

We can now proceed to transform the Honest Verifier Zero Knowledge (HVZK) protocol from Section 3 into a signature scheme by using the Fiat-Shamir transform [14] to obtain the desired security level of λ bits.

The public key is given by the two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, while the secret key consists of $\vec{x}, \vec{y} \in (\mathbb{F}_q)^{n-2}$ with $(\vec{x}', \vec{y}') \in \text{NSBC}[\vec{u}, \vec{v}]$ for $\vec{x}' \coloneqq (\vec{x}, 1, 0)$ and $\vec{y}' \coloneqq (\vec{y}, 0, 1)$.

5.1 Removing the interactivity of the protocol

In the signature scheme, all algorithms of the prover and verifier but the final verification are executed by the signer. Only the final verification is performed by the verifier.

Using the five-round Fiat-Shamir transform, we simply replace the first and second challenge by hash values based on the random oracle. To derive the first challenge, we compute:

$$h_0 \leftarrow \mathcal{H}\left(\left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}\right)$$

and expand it into $\vec{\mu}$ and $(\alpha_i)_{i\in[\tau\cdot D]}$. Note that neither h_0 nor the derived $\vec{\mu}$ and $(\alpha_i)_{i\in[\tau\cdot D]}$ need to be sent as part of the signature since they can be recomputed from the offsets $\delta_{\vec{y}_j}$ and δ_{z_j} using $\mathcal{H}$.

To derive the second challenge, we continue hashing, including now A, B, $\left(a^{(j)}\right)_{j\in[\tau]}$ and b into a hash h_1 from which we obtain the second challenge $\left(i_j^*\right)_{j\in[\tau]}$ using $\mathcal{H}$.

This also permits additional reduction of the signature size. Indeed, since A and $\left(a^{(j)}\right)_{j\in[\tau]}$ are recomputed during the verification phase, they do not need to be sent. Instead, sending (and verifying) the hash h_1 is enough. For a detailed description of the resulting Signing and Verification algorithms, see Appendix A.

5.2 Signature size

As mentioned in [17], we need the bitsize of the elements of $\mathbb{F}_{q^k}$ to be at least 2λ to avoid generic attacks based on collision finding. Therefore, since $n \approx \frac{k}{2}$, the bitsize of the elements of $(\mathbb{F}_q)^{n-2}$ is λ. The signature consists of the following elements:

- One hash value corresponding to the global commitment of the protocol of size 2λ.
- One salt of size λ.
- The punctured key of the correlated PPRF family of size $\lambda\tau\log_2(N)$.
- The coefficient $B \in \mathbb{F}_{q^k}$.
- The coefficient $b = z_0 + \vec{y}_0 \cdot \vec{\mu} \in \mathbb{F}_{q^k}$.
- The offset values $\delta_{\vec{y}_0}, \ldots, \delta_{\vec{y}_{\tau-1}} \in (\mathbb{F}_q)^{n-2}$ and $\delta_{z_1}, \ldots, \delta_{z_{\tau-1}} \in \mathbb{F}_{q^k}$.

The total communication cost in bits of the protocol is therefore

$$\begin{aligned}\mathsf{size} &\geq \lambda\tau\log_2(N) + \tau\lambda + (\tau-1)2\lambda + 2\lambda + 2\lambda + 2\lambda + \lambda \\ &= \lambda\tau\log_2(N) + 3\tau\lambda + 5\lambda \\ &\approx \lambda^2 + 3\tau\lambda + 6\lambda.\end{aligned}$$

The final approximate size depends on the exact value of $\tau\log_2(N)$. We need $\tau\log_2(N) > \lambda$ and take $\tau\log_2(N) \approx \lambda + 1$, as far as factorization of numbers close to $\lambda + 1$ permits.

From Theorem 1, we know that the total soundness error of the identification protocol we use is bounded by

$$\tau 2^{-k} + 2^{1-\tau\cdot D}. \tag{6}$$

In general, for a signature scheme based on the Fiat-Shamir transformation of a 5 round protocol, the cost of forgery is given by the Kales and Zaverucha formula, based on their attack from [20]:

$$\text{cost} = \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau}\left\{\frac{1}{\sum_{i=\tau_1}^{\tau}\binom{\tau}{i}\mathrm{p}^i(1-\mathrm{p})^{\tau-i}} + N^{\tau_2}\right\},$$

where p denotes the false positive probability of the underlying identification scheme.

In our case, the false positive probability in the sense of Kales and Zaverucha is $\mathrm{p} = 2^{-k}$ from Lemma 1. Since this is extremely small compared to the desired security level, the Kales and Zaverucha attack does not apply in our case. In fact, our proof of Theorem 1 uses a union bound argument to prevent any false positive from happening and the same proof directly shows the security level of our signature scheme in the random oracle model.

Consider the security level of $\lambda = 128$ bits and the parameters $q = 2, k = 257$ and $n = 130$. For these parameters, the values for D and τ, including the respective signature sizes are displayed in Table 1. We note that, unlike the standard SBC scheme [17], using $(D, \tau) = (16, 8)$ and $(D, \tau) = (8, 16)$ does not yield a signature with 128 bits security level, since (6) yields a security parameter smaller than 127 bits for these cases. This one bit gap comes from the fact that the evaluation point of the quadratic polynomial is selected from a smaller set of values. For this reason, the potential existence of two roots boosts the probability of a cheating prover by a factor of two.

Table 1. Signature size for $N = 2^D$ parties and τ rounds with parameters $q = 2, k = 257$ and $n = 130$ for the security parameter $\lambda = 128$ bits.

D	τ	$\lambda^2 + 3\tau\lambda + 6\lambda$	|sgn|
9	15	2 864 B	2 962 B
10	13	2 768 B	2 786 B
11	12	2 720 B	2 770 B
12	11	2 672 B	2 722 B
13	10	2 624 B	2 642 B
15	9	2 576 B	2 674 B

6 Implementation and performance

In this section, we provide the signature sizes and running times of the scheme based on our C implementation, which is an adaptation of the artifact from [17]. The total communication cost of the standard SBC paper is $\lambda^2 + 16\tau\lambda + 3\lambda$ bits, while the communication cost of our scheme is $\lambda^2 + 3\tau\lambda + 6\lambda$ bits.

We compare our scheme with the MPCitH signature scheme for SBC proposed in [17] in Table 2. We see that the case $(D, \tau) = (13, 10)$ yields the smallest signature size of 2 642 bytes, which results in a decrease of the signature size of $\approx 30\%$ compared to the smallest signature size obtained by the standard SBC signature scheme from [17], i.e. for $(D, \tau) = (16, 8)$. The running time of the smallest $\mathrm{SBC}_{\mathrm{VOLE}}$ instance $(D, \tau) = (13, 10)$ is reduced by 90% compared to the smallest instance of the standard SBC version for $(D,\tau)=(16,8)$. We also note that the $\mathrm{SBC}_{\mathrm{VOLE}}$ scheme for $(D,\tau)=(9,15)$ and $(D,\tau)=(15,9)$ is redundant since it is slower and larger compared to the $(13,10)$ version.

Improving signature size with variable-sized trees. The main restriction on finding good parameter choices is to find multiples of the desired number of trees, i.e. the desired value of τ, slightly above 128. As seen in Table 2, this leads to very limited good choices. We can improve the accessible range of parameters by allowing the second level trees in the correlated family to have variable sizes. For example, with $\tau=9$, we could use four trees of size 2^{15} and five of size 2^{14}. In this situation, the number of binary sharings would be 130, i.e. the same as in the $\tau=10$ case. This approach can potentially reduce the signature sizes to the estimated value $\lambda^2+3\tau\lambda+6\lambda$ given in Table 1.

Table 2. Signature sizes and running times for $\lambda=128$ using $N=2^D$ parties and τ rounds with the parameters $q=2$, $k=257$ and $n=130$ for standard SBC and $\mathrm{SBC}_{\mathrm{VOLE}}$ using an AMD EPYC 9374F processor running at 3.85 GHz.

Parameters		standard SBC [17]			$\mathrm{SBC}_{\mathrm{VOLE}}$		
D	τ	|sgn|	Sign	Verify	|sgn|	Sign	Verify
8	16	5 436 B	0.77 ms	0.68 ms	—	—	—
9	15	5 340 B	0.90 ms	0.81 ms	2 962 B	2.41 ms	2.40 ms
10	13	4 842 B	1.29 ms	1.21 ms	2 786 B	1.58 ms	1.57 ms
11	12	4 665 B	1.60 ms	1.51 ms	2 770 B	1.60 ms	1.59 ms
12	11	4 457 B	2.25 ms	2.15 ms	2 722 B	1.79 ms	1.77 ms
13	10	4 216 B	3.47 ms	3.34 ms	**2 642** B	**2.21 ms**	**2.19 ms**
15	9	4 087 B	11.27 ms	10.94 ms	2 674 B	5.13 ms	5.02 ms
16	8	3 766 B	19.67 ms	19.17 ms	—	—	—

References

1. Aguilar-Melchor, C., Gama, N., Howe, J., Hülsing, A., Joseph, D., Yue, D.: The return of the SDitH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 564–596. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4˙20
2. American National Standards Institute, Inc.: ANSI X9.62 public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA) (Nov 16, 2005), `https://standards.globalspec.com/std/1955141/ANSI%20X9.62`
3. Aragon, N., Bardet, M., Bidoux, L., Chi-Domínguez, J., Dyseryn, V., Feneuil, T., Gaborit, P., Neveu, R., Rivain, M., Tillich, J.: MIRA. Tech. rep., National Institute of Standards and Technology (2023), available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`

4. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 581–615. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4˙19
5. Bettale, L., Kahrobaei, D., Perret, L., Verbel, J.: Biscuit. Tech. rep., National Institute of Standards and Technology (2023), available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`
6. Bidoux, L., Feneuil, T., Gaborit, P., Neveu, R., Rivain, M.: Dual support decomposition in the head: Shorter signatures from rank SD and MinRank. Cryptology ePrint Archive, Paper 2024/541 (2024), `https://eprint.iacr.org/2024/541`, `https://eprint.iacr.org/2024/541`
7. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Berlin, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0˙15
8. Bui, D.: Shorter VOLEitH signature from multivariate quadratic. Cryptology ePrint Archive, Paper 2024/465 (2024), `https://eprint.iacr.org/2024/465`, `https://eprint.iacr.org/2024/465`
9. Bui, D., Carozza, E., Couteau, G., Goudarzi, D., Joux, A.: Short Signatures from Regular Syndrome Decoding, Revisited. Cryptology ePrint Archive, Paper 2024/252 (2024), `https://eprint.iacr.org/2024/252`
10. Cui, H., Liu, H., Yan, D., Yang, K., Yu, Y., Zhang, K.: 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`, `https://eprint.iacr.org/2024/040`
11. Feneuil, T.: Building MPCitH-based signatures from MQ, MinRank, rank SD and PKP. Cryptology ePrint Archive, Report 2022/1512 (2022), `https://eprint.iacr.org/2022/1512`
12. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 541–572. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4˙19
13. Feneuil, T., Rivain, M.: MQOM — MQ on my Mind. Tech. rep., National Institute of Standards and Technology (2023), available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`
14. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Berlin, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7˙12
15. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986). https://doi.org/10.1145/6490.6503
16. Guo, X., Yang, K., Wang, X., Zhang, W., Xie, X., Zhang, J., Liu, Z.: Half-tree: Halving the cost of tree expansion in COT and DPF. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part I. LNCS, vol. 14004, pp. 330–362. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30545-0˙12
17. Huth, J., Joux, A.: MPC in the head using the subfield bilinear collision problem. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part I. LNCS, vol. 14920, pp. 39–70. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68376-3˙2

18. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250794
19. Joux, A., Pierrot, C.: Algorithmic aspects of elliptic bases in finite field discrete logarithm algorithms (2024). https://doi.org/10.3934/amc.2022085, `https://www.aimsciences.org/article/id/6368515a6aa93c395b347970`
20. Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 20. LNCS, vol. 12579, pp. 3–22. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-65411-5˙1
21. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. Cryptology ePrint Archive, Report 2018/475 (2018), `https://eprint.iacr.org/2018/475`
22. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the Association for Computing Machinery **21**(2), 120–126 (Feb 1978). https://doi.org/10.1145/359340.359342
23. Roy, L.: SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 657–687. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15802-5˙23
24. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0˙22

A Algorithms for the signature scheme

Below, we provide pseudocode for the algorithms of the SBC signature scheme described in Section 5. The Signing algorithm is given in Figure 7, while the Verification algorithm is given in Figure 8.

```
Inputs: Secret key $\mathsf{sk} = (\vec{x}, \vec{y})$, public key $\mathsf{pk} = (\vec{u}, \vec{v})$ and a message $\mathsf{msg} \in \{0,1\}^*$

 1. Sample $R_{\vec{x}} \xleftarrow{\$} (\mathbb{F}_2)^{n-2}$ and a random seed for the cGGM trees derivation
 2. Compute Pre-Tree: $\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau - 1]\!]}\right) \leftarrow \mathsf{cGGM}(R_{\vec{x}}, R_{\vec{x}} \oplus \vec{x}, 2^\tau)$
 3. Fold Pre-tree: $\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}\right)_{j \in [\tau]} \leftarrow \mathsf{Folding}\left(\vec{x}^{[\![0]\!]}, \ldots, \vec{x}^{[\![2^\tau - 1]\!]}\right)$
 4. For $j \in [\tau]$:
    – Get Tree $T_j$: $\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D - 1]\!]}\right) \leftarrow \mathsf{cGGM}\left(\vec{x}^{[\![0]\!]_j}, \vec{x}^{[\![1]\!]_j}, 2^D\right)$
    – Expand $T_j$ and get $\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D - 1]\!]}\right)$ and $\left({z_j}^{[\![0]\!]}, \ldots, {z_j}^{[\![2^D - 1]\!]}\right)$
    – Fold Tree $T_j$:
       • $\left(\vec{x}_j^{[\![0]\!]_i}, \vec{x}_j^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{x}_j^{[\![0]\!]}, \ldots, \vec{x}_j^{[\![2^D - 1]\!]}\right)$
       • $\left(\vec{y}_j^{[\![0]\!]_i}, \vec{y}_j^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left(\vec{y}_j^{[\![0]\!]}, \ldots, \vec{y}_j^{[\![2^D - 1]\!]}\right)$
       • $\left({z_j}^{[\![0]\!]_i}, {z_j}^{[\![1]\!]_i}\right)_{i \in [D]} \leftarrow \mathsf{Folding}\left({z_j}^{[\![0]\!]}, \ldots, {z_j}^{[\![2^D - 1]\!]}\right)$
    – Let $z \leftarrow {z_0}^{[\![0]\!]_0} \oplus {z_0}^{[\![1]\!]_0}$
    – Compute offsets $\delta_{\vec{y}_j} \leftarrow \vec{y} \oplus \vec{y}_j^{[\![0]\!]_0} \oplus \vec{y}_j^{[\![1]\!]_0}$ and $\delta_{z_j} \leftarrow z \oplus {z_j}^{[\![0]\!]_0} \oplus {z_j}^{[\![1]\!]_0}$
 5. $h_0 \leftarrow \mathcal{H}\left(\mathsf{msg}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j \in [\tau]}\right)$
 6. $\vec{\mu} \leftarrow \mathcal{H}(h_0)$
 7. For $m \in [\tau \cdot D]$: $\alpha_m \leftarrow \mathcal{H}(m, h_0)$
 8. Compute $\vec{A}_{F_x} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j \cdot D}^{(j+1) \cdot D - 1} \vec{x}_j^{[\![0]\!]_i} \alpha_i$
 9. Compute $\vec{A}_{F_y} \leftarrow \sum_{j=0}^{\tau-1} \sum_{i=j \cdot D}^{(j+1) \cdot D - 1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$
10. Compute the coefficients $A$ and $B$ of $\Phi$ as in identification protocol
11. Compute $b \leftarrow z + \vec{y} \cdot \vec{\mu}$
12. For $j \in [\tau]$:
    – Compute $\vec{A}_{F_y}^{(j)} \leftarrow \sum_{i=j \cdot D}^{(j+1) \cdot D - 1} \vec{y}_j^{[\![0]\!]_i} \alpha_i$ and $\vec{A}_{F_z}^{(j)} \leftarrow \sum_{i=j \cdot D}^{(j+1) \cdot D - 1} {z_j}^{[\![0]\!]_i} \alpha_i$
    – Compute $a^{(j)} \leftarrow \vec{A}_{F_z}^{(j)} + \vec{A}_{F_y}^{(j)} \cdot \vec{\mu}$
13. $h_1 \leftarrow \mathcal{H}\left(h_0, \mathsf{msg}, A, B, \left(a^{(j)}\right)_{j \in [\tau]}, b\right)$
14. For $j \in [\tau]$: $i_j^* \leftarrow \mathcal{H}(j, h_1)$
15. Compute the Pre-Challenge $i_p^* = \sum_{j=0}^{\tau-1} \cdot \lfloor i_j^* / 2^{D-1} \rfloor 2^j$
16. Compute the Pre-Tree PPRF key $K_{i_p^*}$ and the PPRF key $K_{i_0^*, \ldots, i_{\tau-1}^*}$
17. Output the signature $\sigma \leftarrow \left(h_1, \mathsf{seed}, \left(\delta_{\vec{y}_j}\right)_{j \in [\tau]}, \left(\delta_{z_j}\right)_{j \in [\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*, \ldots, i_{\tau-1}^*}\right)$
```

Fig. 7. NSBC signature scheme - Signing algorithm

```
Inputs: Public key $\mathsf{pk} = (\vec{u}, \vec{v})$, a message $\mathsf{msg} \in \{0,1\}^*$ and a signature $\sigma = \left(h_1, \mathsf{seed}, \left(\delta_{\vec{y}_j}\right)_{j\in[\tau]}, \left(\delta_{z_j}\right)_{j\in[\tau]^*}, B, b, K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}\right)$

 1. For $j \in [\tau]$: $i_j^* \leftarrow \mathcal{H}(j, h_1)$
 2. Recompute and fold all trees (with one missing leaf each) using $K_{i_p^*}, K_{i_0^*,\ldots,i_{\tau-1}^*}$ and $\mathsf{seed}$
 3. For $(i,j) \in [D] \times [\tau]$, let $b_{i,j}$ denote the known position of the respective hypercube binary sharing
 4. $\delta_{z_0} \leftarrow 0$
 5. $h_0' \leftarrow \mathcal{H}\left(\mathsf{msg}, \left(\delta_{\vec{y}_j}, \delta_{z_j}\right)_{j\in[\tau]}\right)$
 6. $\vec{\mu} \leftarrow \mathcal{H}(h_0')$
 7. For $m \in [\tau \cdot D]$: $\alpha_m \leftarrow \mathcal{H}(m, h_0)$
 8. For $j \in [\tau]$:
      – Compute $\Delta_j^* \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} b_{i,j}\alpha_i$
      – Compute $\vec{F}_y^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i \left(\vec{y}_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{\vec{y}_j}\right)$
      – Compute $Z^{(j)}\left(\Delta_j^*\right) \leftarrow \sum_{i=j\cdot D}^{(j+1)\cdot D-1} \alpha_i \left(z_j^{[\![b_{i,j}]\!]_i} + b_{i,j}\delta_{z_j}\right)$
      – $a^{(j)'} \leftarrow Z^{(j)}\left(\Delta_j^*\right) \oplus \vec{F}_y^{(j)}(\Delta_j^*) \cdot \vec{\mu} \oplus b\Delta_j^*$
 9. $\Delta^* \leftarrow \sum_{j=0}^{\tau-1} \Delta_j^*$
10. $\vec{F}_x(\Delta^*) \leftarrow \sum_{m=0}^{\tau\cdot D-1} \vec{x}_m^{[\![b_m]\!]}\alpha_m$
11. $\vec{F}_y(\Delta^*) \leftarrow \sum_{j=0}^{\tau-1} \vec{F}_y^{(j)}(\Delta_j^*)$
12. Compute $\Phi(\Delta^*)$ as
    $\left(\vec{u} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right) - \left(\vec{u} \cdot \begin{pmatrix} \vec{F}_y(\Delta^*) \\ 0 \\ 1 \end{pmatrix}\right)\left(\vec{v} \cdot \begin{pmatrix} \vec{F}_x(\Delta^*) \\ 1 \\ 0 \end{pmatrix}\right)$
13. $A' \leftarrow \Phi(\Delta^*) - B\Delta^*$
14. $h_1' \leftarrow \mathcal{H}\left(h_0', \mathsf{msg}, A', B, \left(a^{(j)'}\right)_{j\in[\tau]}, b\right)$
15. If $h_1' = h_1$ output ACCEPT, otherwise output REJECT
```

Fig. 8. NSBC signature scheme - Verification algorithm