MPC in the head using the subfield bilinear collision problem

Janik Huth and Antoine Joux

CISPA – Helmholtz Center for Information Security, Saarbrücken, Germany.
janik.huth@cispa.de, joux@cispa.de

Abstract. In this paper, we introduce the subfield bilinear collision problem and use it to construct an identification protocol and a signature scheme. This construction is based on the MPC-in-the-head paradigm and uses the Fiat-Shamir transformation to obtain a signature.

1 Introduction

The use of the Multi-Party Computation in the Head (MPCitH) paradigm for Zero-Knowledge (ZK) protocols was introduced in [16]. The general idea is to use any NP-relation $\mathcal{R}(x, w)$ to obtain a ZK-protocol in which a prover $\mathcal{P}$ convinces a verifier $\mathcal{V}$ that she knows a valid witness w for a given (public) value of x without revealing any information about x. In this paper, we introduce a problem which naturally arises by using the standard techniques for finding discrete logarithms in small characteristic finite fields to construct such a protocol. We then use the Fiat-Shamir heuristic [12] to turn the corresponding MPCitH protocol into a signature scheme. The advantage of introducing this new problem is that it can easily be transformed into an MPC protocol which yields a smaller signature size compared to existing schemes.
The problem we consider throughout this paper uses three parameters: A prime power q and two positive integers k, n.

The subfield bilinear collision (SBC) Problem.

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

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

We denote an instance of the SBC problem given by the two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$ by $\mathrm{SBC}[\vec{u}, \vec{v}]$. If the vectors $\vec{x}, \vec{y} \in \left(\mathbb{F}_q\right)^n$ are a solution of $\mathrm{SBC}[\vec{u}, \vec{v}]$, we use the notation $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$.
Note that we use the canonical embedding from $\mathbb{F}_q$ to $\mathbb{F}_{q^k}$ for the vectors $\vec{x}$ and $\vec{y}$. Therefore, the operations in (1) are well-defined.
This problem originates from a heuristic discrete logarithm algorithm in small characteristic finite fields and seems hard to solve for large n when $n \approx \frac{k}{2}$.

2 Preliminaries

2.1 Notations

In this section, we introduce some standard (cryptographic) notation which we need throughout this paper. The target security level of our scheme is denoted by λ. For any positive integer $m \in \mathbb{Z}^+$, we denote the set $\{1, \ldots, m\}$ by $[m]$. Let D denote any probability distribution. Then the notation $s \leftarrow_\$ D$ indicates that s is sampled from D. If S is a finite set, then the notation $s \leftarrow_\$ S$ means that s is uniformly sampled at random from S. By using the notation $s \xleftarrow{r}_\$ S$, we indicate that s is sampled pseudorandomly from S based on the seed r. Additionally, we assume the random oracle model and consider any hash function used in this article to be a random oracle.

Let $\mathbb{F}$ be a finite field and let k, n be positive integers. Then we denote the set of all $n \times k$ matrices over $\mathbb{F}$ by $\mathcal{M}_{n,k}(\mathbb{F})$. For a fixed vector $\vec{x} \in \mathbb{F}^n$, the subspace generated by $\vec{x}$ is denoted by $\langle \vec{x} \rangle$ and the projective space over $\mathbb{F}$ of dimension n by $\mathbb{P}_n(\mathbb{F})$. We denote the polynomial ring over $\mathbb{F}$ in X by $\mathbb{F}[X]$.

2.2 Basic considerations about the SBC problem

Colinear collisions are easy to find. For any vector $\vec{x} \in (\mathbb{F}_q)^n$ and for any scalar $\alpha \in \mathbb{F}_q$, we see that

$$(\vec{u} \cdot \vec{x})\,(\vec{v} \cdot (\alpha\vec{x})) = \alpha\,(\vec{u} \cdot \vec{x})\,(\vec{v} \cdot \vec{x}) = (\vec{u} \cdot (\alpha\vec{x}))\,(\vec{v} \cdot \vec{x})\,.$$

Therefore, the pair $(\vec{x}, \alpha\vec{x})$ always yields a collision. This is why we insist on non-colinear vectors $\vec{x}, \vec{y}$ in the definition of the SBC problem.

Choice of the parameters k and n. In this section, we examine the relation between the parameters k and n and the hardness of the corresponding instances of the SBC problem. Let $\mathrm{SBC}[\vec{u}, \vec{v}]$ be an instance of the SBC problem. By rewriting Equation (1), we want to find non-colinear vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ such that

$$\frac{\vec{u} \cdot \vec{x}}{\vec{v} \cdot \vec{x}} = \frac{\vec{u} \cdot \vec{y}}{\vec{v} \cdot \vec{y}},$$

assuming that $\vec{v} \cdot \vec{x} \neq 0$ and $\vec{v} \cdot \vec{y} \neq 0$. Consider the map

$$\begin{aligned} \phi\colon \mathbb{P}_{n-1}(\mathbb{F}_q) &\to \mathbb{F}_{q^k} \\ \vec{x} &\mapsto \frac{\vec{u} \cdot \vec{x}}{\vec{v} \cdot \vec{x}}. \end{aligned}$$

We do a heuristic analysis of the expected number of collisions of ϕ. For simplicity, we assume that the range of ϕ is $\mathbb{F}_{q^k}^*$ and that the scalar product $\vec{v} \cdot \vec{x} \neq 0$

for every $\vec{x} \in \mathbb{P}_{n-1}(\mathbb{F}_q)$. We consider $Q \coloneqq \frac{q^n-1}{q-1} \approx q^{n-1}$ vectors in the domain of ϕ. Therefore, there are

$$\binom{Q}{2} = \frac{Q(Q-1)}{2} \approx \frac{Q^2}{2}$$

possible pairs of vectors which could be part of a collision. Assuming that the range of ϕ is $\mathbb{F}_{q^k}^*$, the expected number of collisions depending on k and n can be estimated by

$$\frac{Q(Q-1)}{2(q^k-1)} \approx \frac{Q^2}{2q^k} \approx \frac{q^{2(n-1)}}{2q^k} = \frac{q^{2n-k-2}}{2}.$$

First, we consider the case $n > k+1$. Fix a random vector $\vec{x} \in \mathbb{P}_{n-1}(\mathbb{F}_q)$ and let $c = \phi(\vec{x})$. Finding a collision $(\vec{x}, \vec{y})$ is then equivalent to finding a vector $\vec{y} \in \mathbb{P}_{n-1}(\mathbb{F}_q)$ with $\vec{y} \neq \vec{x}$ such that

$$(c\vec{v} - \vec{u}) \cdot \vec{y} = 0.$$

Each entry of the vectors $\vec{u}$ and $\vec{v}$ is an element of $\mathbb{F}_{q^k}$. Let $(\alpha_1, \ldots, \alpha_k)$ be an $\mathbb{F}_q$ basis of $\mathbb{F}_{q^k}$. We can express each entry of $(c\vec{v} - \vec{u})$ using this basis to obtain a matrix of the form

$$\mathrm{M} = \begin{pmatrix} a_1^{(1)} & \ldots & a_n^{(1)} \\ \vdots & \ddots & \vdots \\ a_1^{(k)} & \ldots & a_n^{(k)} \end{pmatrix} \in \mathcal{M}_{k,n}(\mathbb{F}_q),$$

where $cv_i - u_i = \sum_{j=1}^{k} a_i^{(j)} \alpha_j$ for $i = 1, \ldots, n$. By construction, $\vec{x}$ is in the right kernel of M. To obtain a collision, the goal is to find another vector $\vec{y}$ in the right kernel of M with $\vec{y} \neq \vec{x}$ in $\mathbb{P}_{n-1}(\mathbb{F}_q)$. If $n > k+1$, such a vector $\vec{y}$ has to exist and can be computed by solving the linear system $\mathrm{M}\vec{y} = 0$.

Next, consider the case $n \leq k+1$. We pick N pairwise non-colinear vectors $\vec{x}_1, \ldots, \vec{x}_N$ and want to estimate the probability that at least one of these N distinct vectors is part of a collision with the remaining $Q - N \approx Q$ vectors. If one of these N vectors, say $\vec{x}_j$, is part of a collision, we can compute $c_j = \phi(\vec{x}_j)$ and again solve the linear system

$$(c_j\vec{v} - \vec{u}) \cdot \vec{y} = 0$$

to find the vector $\vec{y}$. Following a standard heuristic argument, we expect such a collision to appear if $NQ > q^k$, i.e. if $N > q^{k+1-n}$. By using this technique, we get an attack against the SBC problem in $\mathcal{O}(q^{k+1-n})$, which is quite efficient for n close to k.

If we consider $n \ll \frac{k}{2}$, we do not expect any collisions to exist. Therefore, a setup where $n \approx \frac{k}{2}$ seems to be a good parameter choice for the SBC problem: We expect collisions to exist, but the technique for finding collisions described above is not efficient.

Normalization of solutions. Consider a solution $(\vec{x}, \vec{y}) \in \mathrm{SBC}[\vec{u}, \vec{v}]$ of the SBC problem. We define the matrix

$$N = \begin{pmatrix} x_{n-1} & y_{n-1} \\ x_n & y_n \end{pmatrix} \in \mathcal{M}_{2,2}(\mathbb{F}_q).$$

First, we consider the case that N is singular. Let $\vec{K} \neq (0,0)$ be an element of the left kernel of N, i.e. $\vec{K} \cdot (x_{n-1}, x_n) = 0$ and $\vec{K} \cdot (y_{n-1}, y_n) = 0$. We distinguish two cases:

- $\vec{K} \in \langle (1,0) \rangle$: In this case, the entries x_{n-1} and y_{n-1} to not contribute towards the sum in the scalar product computations. Therefore, we could decrease the size of the vectors $\vec{u}$ and $\vec{v}$ to $n-1$ by removing the entries u_{n-1} and v_{n-1} from $\vec{u}$ and $\vec{v}$ respectively. Afterwards, we can still obtain a solution $\left(\vec{x'}, \vec{y'}\right)$ to the SBC problem with smaller dimension $n-1$ by removing the entries x_{n-1} and y_{n-1} from $\vec{x}$ and $\vec{y}$ respectively.
- $\vec{K} \in \langle (\alpha, 1) \rangle$ for an $\alpha \in \mathbb{F}_q$: For simplicity, we normalize $\vec{K}$ to be of the form $\vec{K} = (c, 1)$. Then the following equations hold:

$$\begin{aligned}
(u_{n-1}, u_n) \cdot (x_{n-1}, x_n) &= (u_{n-1}, u_n) \cdot (x_{n-1}, x_n) - u_n \vec{K} \cdot (x_{n-1}, x_n), \\
(u_{n-1}, u_n) \cdot (y_{n-1}, y_n) &= (u_{n-1}, u_n) \cdot (y_{n-1}, y_n) - u_n \vec{K} \cdot (y_{n-1}, y_n), \\
(v_{n-1}, v_n) \cdot (x_{n-1}, x_n) &= (v_{n-1}, v_n) \cdot (x_{n-1}, x_n) - v_n \vec{K} \cdot (x_{n-1}, x_n), \\
(v_{n-1}, v_n) \cdot (y_{n-1}, y_n) &= (v_{n-1}, v_n) \cdot (y_{n-1}, y_n) - v_n \vec{K} \cdot (y_{n-1}, y_n).
\end{aligned}$$

  If we replace u_{n-1} by $(u_{n-1} - cu_n)$ and v_{n-1} by $(v_{n-1} - cv_n)$, we can remove the entries u_n and v_n from $\vec{u}$ and $\vec{v}$ respectively to obtain a new instance of the SBC problem with smaller dimension $n-1$. This new system still has a solution of the form $\vec{x} = (x_1, \ldots, x_{n-1})$ and $\vec{y} = (y_1, \ldots, y_{n-1})$.

In both cases, we can reduce the dimension n of the vectors but still obtain a solution of the SBC problem if N is singular. Since we do not want the dimension of the problem to be easily reduced, we assume that the matrix N is non-singular when we consider solutions of the SBC problem.
Assume that N is non-singular. Then, we can simplify the solutions in the following way: Compute the inverse of N, which we denote by

$$W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \in \mathcal{M}_{2,2}(\mathbb{F}_q).$$

We can use W to normalize the solution vectors $\vec{x}, \vec{y}$ and obtain

$$W \cdot \begin{pmatrix} \vec{x} \\ \vec{y} \end{pmatrix} = \begin{pmatrix} \vec{x'} & 1 & 0 \\ \vec{y'} & 0 & 1 \end{pmatrix}$$

for some $\vec{x'}, \vec{y'} \in (\mathbb{F}_q)^{n-2}$. The normalized vectors $\left(\left(\vec{x'}, 1, 0\right), \left(\vec{y'}, 0, 1\right)\right)$ are also a solution of the same SBC problem $\mathrm{SBC}[\vec{u}, \vec{v}]$. To see that, we check the

equation

$$\begin{aligned}\left[\vec{u}\cdot\left(\vec{x'},1,0\right)\right]\left[\vec{v}\cdot\left(\vec{y'},0,1\right)\right] &= \left[\vec{u}\cdot\left(w_{11}\vec{x}+w_{21}\vec{y}\right)\right]\left[\vec{v}\cdot\left(w_{21}\vec{x}+w_{22}\vec{y}\right)\right]\\ &= w_{11}w_{21}\left(\vec{u}\cdot\vec{x}\right)\left(\vec{v}\cdot\vec{x}\right)+w_{11}w_{22}\left(\vec{u}\cdot\vec{x}\right)\left(\vec{v}\cdot\vec{y}\right)\\ &\quad +w_{12}w_{21}\left(\vec{u}\cdot\vec{y}\right)\left(\vec{v}\cdot\vec{x}\right)+w_{12}w_{22}\left(\vec{u}\cdot\vec{y}\right)\left(\vec{v}\cdot\vec{y}\right)\\ &= w_{21}w_{11}\left(\vec{u}\cdot\vec{x}\right)\left(\vec{v}\cdot\vec{x}\right)+w_{22}w_{11}\left(\vec{v}\cdot\vec{y}\right)\left(\vec{u}\cdot\vec{x}\right)\\ &\quad +w_{21}w_{12}\left(\vec{v}\cdot\vec{x}\right)\left(\vec{u}\cdot\vec{y}\right)+w_{22}w_{12}\left(\vec{v}\cdot\vec{y}\right)\left(\vec{u}\cdot\vec{y}\right)\\ &= \left[\vec{u}\cdot\left(w_{21}\vec{x}+w_{22}\vec{y}\right)\right]\left[\vec{v}\cdot\left(w_{11}\vec{x}+w_{12}\vec{y}\right)\right]\\ &= \left[\vec{u}\cdot\left(\vec{y'},0,1\right)\right]\left[\vec{v}\cdot\left(\vec{x'},1,0\right)\right].\end{aligned}$$

In the third equation, we used the commutativity of the multiplication in $\mathbb{F}_{q^k}$ and the fact that $(\vec{x},\vec{y}) \in \mathrm{SBC}[\vec{u},\vec{v}]$. By this argument, we can always normalize a solution of the SBC problem to obtain vectors of the form $\vec{x} = \left(\vec{x'},1,0\right)$ and $\vec{y} = \left(\vec{y'},0,1\right)$. In this case, $\vec{x}$ and $\vec{y}$ are non-colinear. Throughout the rest of this paper, we therefore assume any solution of the SBC problem to be of this normalized form.

The normalized subfield bilinear collision (NSBC) Problem.

- *Problem instance:* Two vectors $\vec{u},\vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, which are linearly independent over $\mathbb{F}_q$.
- *Solution:* Two vectors $\vec{x},\vec{y} \in \left(\mathbb{F}_q\right)^n$ of the form $\vec{x} = \left(\vec{x'},1,0\right)$, $\vec{y} = \left(\vec{y'},0,1\right)$ such that

$$\left(\vec{u}\cdot\vec{x}\right)\left(\vec{v}\cdot\vec{y}\right) = \left(\vec{u}\cdot\vec{y}\right)\left(\vec{v}\cdot\vec{x}\right). \tag{2}$$

Similar to before, we denote an instance of the NSBC problem given by the two vectors $\vec{u},\vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$ by $\mathrm{NSBC}[\vec{u},\vec{v}]$. If $\vec{x},\vec{y} \in \left(\mathbb{F}_q\right)^n$ are a solution of $\mathrm{NSBC}[\vec{u},\vec{v}]$, we use the notation $(\vec{x},\vec{y}) \in \mathrm{NSBC}[\vec{u},\vec{v}]$.

Generation of normalized instances with a solution. Given q, k and n, we can generate an instance of this problem that has a solution in the following way: Uniformly at random choose two vectors $\vec{x'},\vec{y'} \in \left(\mathbb{F}_q\right)^{n-2}$, set $\vec{x} \coloneqq \left(\vec{x'},1,0\right)$ and $\vec{y} \coloneqq \left(\vec{y'},0,1\right)$. Also, randomly choose the coordinates $u_1,\ldots,u_n \in \mathbb{F}_{q^k}$ and $v_1,\ldots,v_{n-1} \in \mathbb{F}_{q^k}$ of $\vec{u}$ and $\vec{v}$. Lastly, compute v_n as

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

If the denominator $y_n\left(\vec{u}\cdot\vec{x}\right)$ happens to be zero, the computation of v_n fails. In this case, we can choose new random vectors and start the computation

again or we can change the value of u_{n-1} and compute v_n successfully. If the unlikely case that $\vec{u}, \vec{v}$ are linearly dependent over $\mathbb{F}_q$ appears, we can easily adapt the vectors accordingly to avoid this case. By this construction, we have that $(\vec{x}, \vec{y}) \in \mathrm{NSBC}[\vec{u}, \vec{v}]$.

Origin of the SBC problem. Consider the discrete logarithm problem (DLP) in the group $G = \mathbb{F}_{q^k}^*$ for some large extension degree k. Let $\mathbb{F}_q[X]$ denote the set of polynomials in X with coefficients in $\mathbb{F}_q$. We can represent $\mathbb{F}_{q^k}^*$ as the quotient ring $\mathbb{F}_q[X]/(I_k(X))$, where I_k is an arbitrary irreducible polynomial of degree k. We briefly describe a heuristic algorithm to solve the DLP in this group, which is based on the function field sieve. This family of algorithms, called Frobenius Representation algorithms, was for example studied in [19]. In the representation phase of the algorithm from [19], the goal is to find two polynomials $h_0, h_1 \in \mathbb{F}_q[X]$ of degree at most 2 such that there exists an irreducible polynomial I_k of degree $k > 2$ with $I_k(X) | (h_1(X)X^q - h_0(X))$. Let θ be a root of I_k in the algebraic closure of $\mathbb{F}_q$. Using I_k, we can represent $\mathbb{F}_{q^k} \cong \mathbb{F}_q[\theta]$ as $\mathbb{F}_q[X]/(I_k(X))$. In this representation, we see that

$$\theta^q = \frac{h_0(\theta)}{h_1(\theta)}.$$

We follow the approach in [19]: In the descent phase of the algorithm, the goal is to find the discrete logarithm of an element $T(\theta) \in \mathbb{F}_{q^k}$. Without loss of generality, we can assume that T is an irreducible polynomial over $\mathbb{F}_q[X]$.[1] Let $\xi \in \mathbb{F}_{q^{\deg(T)}}$ be a root of T. We try to find polynomials $A, B \in \mathbb{F}_q[X, Y]$ of degree $d \approx k/2$ such that

$$B(\xi, 1)A(h_0(\xi), h_1(\xi)) - A(\xi, 1)B(h_0(\xi), h_1(\xi)) = 0 \quad \text{in } \mathbb{F}_{q^{\deg(T)}},$$

or equivalently

$$T(x) | [B(x, 1)A(h_0(x), h_1(x)) - A(x, 1)B(h_0(x), h_1(x))],$$

where A and B are the two homogeneous polynomials

$$A(x, y) \coloneqq \sum_{i=0}^{d} a_i x^i y^{d-i} \quad \text{and} \quad B(x, y) \coloneqq \sum_{i=0}^{d} b_i x^i y^{d-i}.$$

This is in fact a special case of the SBC problem: Define

$$\vec{\mathcal{H}} \coloneqq \begin{pmatrix} h_1(\xi)^d \\ h_1(\xi)^{d-1} h_0(\xi) \\ \vdots \\ h_0(\xi)^d \end{pmatrix} \quad \text{and} \quad \vec{\mathcal{G}} \coloneqq \begin{pmatrix} 1 \\ \xi \\ \vdots \\ \xi^d \end{pmatrix}.$$

[1] Otherwise, we can factor T into irreducible polynomials and consider each factor individually.

Then the goal is to find two non-colinear vectors $\vec{a}, \vec{b} \in (\mathbb{F}_q)^d$ with

$$\left(\vec{\mathcal{H}} \cdot \vec{a}\right)\left(\vec{\mathcal{G}} \cdot \vec{b}\right) - \left(\vec{\mathcal{H}} \cdot \vec{b}\right)\left(\vec{\mathcal{G}} \cdot \vec{a}\right) = 0 \quad \text{in } \mathbb{F}_{q^{\deg(T)}},$$

i.e. $\left(\vec{a}, \vec{b}\right) \in \mathrm{SBC}[\vec{\mathcal{H}}, \vec{\mathcal{G}}]$. In this case, the vectors $\vec{a}, \vec{b}$ represent the coefficients of the polynomials A and B, respectively. In [18], the limiting factor of the discrete logarithm problem computation comes from this variant of the SBC problem. Improving attacks against the SBC problem would therefore lead to improvements of the discrete logarithm computations using the method described above.

Known attacks. Consider an instance $\mathrm{NSBC}[\vec{u}, \vec{v}]$ of the normalized SBC problem. By rewriting Equation (2), the goal is to find vectors $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$ of the form $\vec{x} = \left(\vec{x'}, 1, 0\right)$, $\vec{y} = \left(\vec{y'}, 0, 1\right)$ such that

$$(\vec{u} \cdot \vec{x})(\vec{v} \cdot \vec{y}) - (\vec{u} \cdot \vec{y})(\vec{v} \cdot \vec{x}) = 0.$$

This is a bilinear equation given by the polynomial

$$\begin{aligned} g(x_1, \ldots, x_{n-2}, y_1, \ldots, y_{n-2}) \coloneqq & \left(\sum_{i=1}^{n-2} u_i x_i + u_{n-1}\right)\left(\sum_{i=1}^{n-2} v_i y_i + v_n\right) \\ & - \left(\sum_{i=1}^{n-2} u_i y_i + u_n\right)\left(\sum_{i=1}^{n-2} v_i x_i + v_{n-1}\right). \end{aligned}$$

Since $\vec{u}, \vec{v} \in (\mathbb{F}_{q^k})^n$, the polynomial g is an element of the polynomial ring $\mathbb{F}_{q^k}[X_1, \ldots, X_{n-2}, Y_1, \ldots, Y_{n-2}]$ in $(2n-2)$ variables. We can express this polynomial with k polynomials in the base field $\mathbb{F}_q$ by choosing an $\mathbb{F}_q$ basis $(\alpha_1, \ldots, \alpha_k)$ of $\mathbb{F}_{q^k}$. Therefore, we obtain a system of k bilinear equations of the form

$$\begin{aligned} g_1(x_1, \ldots, x_{n-2}, y_1, \ldots, y_{n-2}) &= 0, \\ &\vdots \\ g_k(x_1, \ldots, x_{n-2}, y_1, \ldots, y_{n-2}) &= 0, \end{aligned}$$

where $g_i \in \mathbb{F}_q[X_1, \ldots, X_{n-2}, Y_1, \ldots, Y_{n-2}]$ for $i \in [k]$. We can solve this bilinear system using Gröbner basis algorithms, for example the so-called F_5 algorithm [9]. For details about Gröbner basis algorithms, see for example [10,25]. We recall the following result from [10]:

Theorem 1 ([10, Corollary 3]). *The complexity of computing a Gröbner basis of a generic bilinear system $g_1, \ldots, g_{n_x+n_y} \in \mathbb{K}[x_1, \ldots, x_{n_x}, y_1, \ldots, y_{n_y}]$ with the F_5 algorithm is upper bounded by*

$$\mathcal{O}\left(\binom{n_x - 1 + n_y - 1 + \min(n_x, n_y)}{\min(n_x, n_y)}^{\omega}\right),$$

where $2 \leq \omega \leq 3$ is the linear algebra constant.

In out setting, we have that $n_x = n_y \approx \frac{k}{2}$, which means we can upper bound the complexity of solving the obtained system of bilinear equations by

$$\mathcal{O}\left(\binom{\frac{3k}{2}}{\frac{k}{2}}^{\omega}\right).$$

By using Stirling's formula, we can asymptotically estimate the binomial coefficient by $2^{H\left(\frac{1}{3}\right)\frac{3k}{2}}$, where $H(p) = -p\log p - (1-p)\log(1-p)$ is the *binary entropy function*, which yields an estimate of roughly $2^{1.38k\omega}$ for the upper bound of the complexity. For a security level of $\lambda = k = 128$ and the best case scenario for the linear algebra constant of $\omega = 2$, this would yield a complexity of 2^{354}.

The upper bound given in Theorem 1 is not tight, there are practical examples of Gröbner basis computations which are faster. In [18, Section 6], a discrete logarithm computation based on the same bilinear system is feasible to compute up to the extension degree of $k = 36$. However, our suggested parameter choice of $k \geq 128$ seems completely out of the reach of these techniques.

2.3 Notations for Multi-Party Computations (MPC)

We briefly introduce the notation we use in the Multi-Party Computation (MPC) protocols throughout this paper. We use additive sharings of finite field elements. Let N be the number of parties. Then an *N-sharing* of a finite field element $x \in \mathbb{F}$ is an N-tuple

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

such that

$$x = \sum_{i=1}^{N} x^{[\![i]\!]} \mod |\mathbb{F}|.$$

We call each $x^{[\![i]\!]}$ a *share* of x. In the MPC protocol which we use, each party receives one of the N shares. With these shares, the parties can then perform computations independently: Assume each party $i \in [N]$ receives the shares $x^{[\![i]\!]}$ and $y^{[\![i]\!]}$ corresponding to sharings of x and y. Let α be a constant. Then the parties can perform the following operations:

- **Addition:** they locally compute $[\![x+y]\!]$ by adding their shares:

  $$(x+y)^{[\![i]\!]} \coloneqq x^{[\![i]\!]} + y^{[\![i]\!]}$$

  for $i \in [N]$. We denote this by $[\![x+y]\!] = [\![x]\!] + [\![y]\!]$.
- **Multiplication by a constant:** they locally compute $[\![\alpha x]\!]$ by multiplying their respective shares by α:

  $$(\alpha x)^{[\![i]\!]} \coloneqq \alpha x^{[\![i]\!]}$$

  for $i \in [N]$. We denote this by $[\![\alpha x]\!] = \alpha[\![x]\!]$.

– **Adding a constant:** the constant α can be shared in a trivial way as $[\![\alpha]\!]_T \coloneqq (\alpha, 0, \ldots, 0)$, where the subscript indicates a trivial sharing. The parties then compute $[\![x+\alpha]\!] = [\![x]\!] + [\![\alpha]\!]_T$. We denote this by $[\![x+\alpha]\!] = [\![x]\!] + \alpha$.

In practice, a sharing of x is usually computed by choosing $N-1$ random values $x^{[\![1]\!]}, \ldots, x^{[\![N-1]\!]}$ and by setting

$$x^{[\![N]\!]} = x - \sum_{i=1}^{N-1} x^{[\![i]\!]} \mod |\mathbb{F}|$$

afterwards. Then the N-tuple $\left(x^{[\![1]\!]}, \ldots, x^{[\![N]\!]}\right)$ is actually a sharing of x. In this paper, we instead mostly use completely random sharings by using N random values $R_x^{[\![1]\!]}, \ldots, R_x^{[\![N]\!]}$ in our MPC protocol to obtain a sharing of x. If we do that, we need an auxiliary value

$$\delta_x \coloneqq x - \sum_{i=1}^{N} R_x^{[\![i]\!]} \mod |\mathbb{F}|,$$

to be able to reconstruct x. The value of x can then be obtained by calculating the sum

$$x = \delta_x + \sum_{i=1}^{N} R_x^{[\![i]\!]} \mod |\mathbb{F}|.$$

In the usual setting, this can be viewed as a sharing of the value x between $N+1$ parties. By using auxiliary values of the form δ_x instead, we can simplify the notation which we need in the identification scheme in Section 3 and choose random values for the sharings.

MPC-in-the-Head Paradigm. Our construction of the identification protocol uses the *MPC-in-the-Head* (MPCitH) paradigm introduced in [16]. Consider an MPC protocol in which N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ securely and correctly compute the output of a function f, given a secret input x. The secret x is hereby given as a sharing $[\![x]\!]$ and each party $\mathcal{P}_i$ receives the share $x^{[\![i]\!]}$. Additionally, the function f should output either 1 or 0, corresponding to accept or reject, respectively. For our protocol, we also require that the views of $N-1$ parties do not reveal any information about x. If this is the case, we say that the protocol is $(N-1)$-*private*. This type of MPC protocol can be used to construct a Zero-Knowledge proof of an x for which $f(x) = 1$. The prover proceeds in the following way:

– she generates a random sharing $[\![x]\!]$ of x.
– she simulates privately ("in the head") all N parties of the MPC protocol.
– she sends commitments to the views of each party to the verifier.
– she sends the output shares $[\![f(x)]\!]$ of the parties to the verifier.

The verifier then randomly chooses $N-1$ parties for which the prover has to reveal the views. Since the protocol is $(N-1)$-private, this does not reveal any information about the secret. The verifier can then check if these views are consistent with an honest execution of the MPC protocol as well as with the commitments from the prover. Since the choice of the $N-1$ opened parties was random, a malicious prover might be able to cheat with probability $\frac{1}{N}$ by corrupting the computation of one (unopened) party. Therefore, the Zero-Knowledge protocol constructed using this paradigm has soundness error $\frac{1}{N}$.

Puncturable PRFs. We use a puncturable pseudo-random function, or puncturable PRF for short, for the MPC protocol in Section 3. A family F of *puncturable PRFs* on $[N]$ is a PRF family F indexed by a key K with domain $[N]$ satisfying the following properties:

- For each key K and index $i \in [N]$ there exists a punctured key K_{i^*} and an algorithm $\mathcal{A}$ such that

$$\text{for each } j \in [N] \setminus \{i\} : \mathcal{A}(K_{i^*}, j) = F_K(j).$$

- The punctured key K_{i^*} does not reveal any information about $F_K(i)$.

In practice, puncturable PRFs are usually constructed using tree PRFs (or GGM trees [14]), as mentioned for example in [7,22]. In this construction, the idea is to build a binary tree of depth $\lceil \log_2(N) \rceil$. The root of this tree is labeled with a master root seed, while the other nodes are labeled inductively by using PRFs on the parent node to get to the left and right children. To reveal all the N leaves except one leave $i^* \in [N]$, the idea is to reveal all the labels of the siblings of the path from the root seed to the leave i^*. By using this method, all leaves except i^* can be reconstructed by communicating $\lceil \log_2(N) \rceil$ seeds instead of $N-1$ seeds.

3 Main Zero-Knowledge Protocol

In this section, we describe an MPC protocol based on the SBC problem. Let $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$, consider the NSBC problem NSBC$[\vec{u}, \vec{v}]$ and let $\vec{x}, \vec{y} \in (\mathbb{F}_q)^n$. Let $X_1, X_2, Y_1, Y_2 \in \mathbb{F}_{q^k}$ be random values. Consider the following polynomial in t:

$$\begin{aligned} F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t) &= (X_1 + t\,(\vec{u}\cdot\vec{x}))\,(Y_1 + t\,(\vec{v}\cdot\vec{y})) - (X_2 + t\,(\vec{v}\cdot\vec{x}))\,(Y_2 + t\,(\vec{u}\cdot\vec{y})) \\ &= X_1Y_1 - X_2Y_2 \\ &\quad + [X_1\,(\vec{v}\cdot\vec{y}) + Y_1\,(\vec{u}\cdot\vec{x}) - X_2\,(\vec{u}\cdot\vec{y}) - Y_2\,(\vec{v}\cdot\vec{x})]\,t \\ &\quad + [(\vec{u}\cdot\vec{x})\,(\vec{v}\cdot\vec{y}) - (\vec{u}\cdot\vec{y})\,(\vec{v}\cdot\vec{x})]\,t^2. \end{aligned}$$

Remark. For this polynomial, we see that $\deg\left(F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}\right) < 2$ if and only if $(\vec{x}, \vec{y}) \in \text{NSBC}[\vec{u}, \vec{v}]$. In that case, we can write

$$F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t) = A + Bt$$

with $A = A_{X_1,X_2,Y_1,Y_2} = X_1Y_1 - X_2Y_2$ and $B = B^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2} = X_1(\vec{v}\cdot\vec{y}) + Y_1(\vec{u}\cdot\vec{x}) - X_2(\vec{u}\cdot\vec{y}) - Y_2(\vec{v}\cdot\vec{x})$. This property of $F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t)$ is a crucial part of our MPC protocol.

3.1 Basic multiparty protocol for the SBC problem

In this protocol, we share the two coefficients A and B and want to prove that $F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t) = A + Bt$, which means that $(\vec{x},\vec{y}) \in \mathrm{NSBC}[\vec{u},\vec{v}]$. Let $\mathcal{P}_1,\ldots,\mathcal{P}_N$ be a group of N provers which want to verify that $(\vec{x},\vec{y})$ is a solution of (2). The basic protocol contains the following steps:

- The parties run an auxiliary protocol to obtain a random value $t_0 \in \mathbb{F}_{q^k}$.
- The parties evaluate F_{X_1,X_2,Y_1,Y_2} at t_0.
- The parties evaluate $A + Bt_0$.
- The parties output accept if $F_{X_1,X_2,Y_1,Y_2}(t_0) = A + Bt_0$ and reject otherwise.

Assume that $\vec{x}$ and $\vec{y}$ are shared as

$$\vec{x} = \vec{\delta}_x + [\![\vec{R}_x]\!],$$
$$\vec{y} = \vec{\delta}_y + [\![\vec{R}_y]\!]$$

and the coefficients A and B are shared as

$$A = \delta_A + [\![R_A]\!],$$
$$B = \delta_B + [\![R_B]\!].$$

We use (exact) sharings given by $[\![X_1]\!], [\![X_2]\!], [\![Y_1]\!], [\![Y_2]\!]$ for the random values X_1, X_2, Y_1, Y_2. Since we only consider normalized solutions of the SBC problem, the last two entries of $\vec{x}$ and $\vec{y}$ can be excluded from the sharing as they are known already.
In particular, the protocol runs as follows:

- Each party $i \in [N]$ receives their private inputs

  $$\vec{R}_x{}^{[\![i]\!]}, \vec{R}_y{}^{[\![i]\!]}, X_1^{[\![i]\!]}, X_2^{[\![i]\!]}, Y_1^{[\![i]\!]}, Y_2^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}$$

  as well as the (public) auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$.
- Each party $\mathcal{P}_i$ chooses a random $t_0^{[\![i]\!]} \in \mathbb{F}_{q^k}$ and all parties simultaneously broadcast their shares.
- The parties reconstruct $t_0 \coloneqq \sum_{i=1}^N t_0^{[\![i]\!]}$.
- Each party $i \in [N]$ computes

$$(X_1 + t_0(\vec{u}\cdot\vec{x}))^{[\![i]\!]} = X_1^{[\![i]\!]} + t_0\left(\vec{u}\cdot\vec{R}_x{}^{[\![i]\!]}\right),$$
$$(X_2 + t_0(\vec{v}\cdot\vec{x}))^{[\![i]\!]} = X_2^{[\![i]\!]} + t_0\left(\vec{v}\cdot\vec{R}_x{}^{[\![i]\!]}\right),$$
$$(Y_1 + t_0(\vec{v}\cdot\vec{y}))^{[\![i]\!]} = Y_1^{[\![i]\!]} + t_0\left(\vec{v}\cdot\vec{R}_y{}^{[\![i]\!]}\right),$$
$$(Y_2 + t_0(\vec{u}\cdot\vec{y}))^{[\![i]\!]} = Y_2^{[\![i]\!]} + t_0\left(\vec{u}\cdot\vec{R}_y{}^{[\![i]\!]}\right),$$
$$(A + Bt_0)^{[\![i]\!]} = R_A^{[\![i]\!]} + R_B^{[\![i]\!]} t_0$$

and broadcasts their shares.

– The parties reconstruct

$$X_1 + t_0\,(\vec{u}\cdot\vec{x}) = \sum_{i=1}^{N} (X_1 + t_0\,(\vec{u}\cdot\vec{x}))^{[\![i]\!]} + t_0 u_{n-1} + t_0\left(\vec{u}\cdot\vec{\delta}_x\right),$$

$$X_2 + t_0\,(\vec{v}\cdot\vec{x}) = \sum_{i=1}^{N} (X_2 + t_0\,(\vec{v}\cdot\vec{x}))^{[\![i]\!]} + t_0 v_{n-1} + t_0\left(\vec{v}\cdot\vec{\delta}_x\right),$$

$$Y_1 + t_0\,(\vec{v}\cdot\vec{y}) = \sum_{i=1}^{N} (Y_1 + t_0\,(\vec{v}\cdot\vec{y}))^{[\![i]\!]} + t_0 v_n + t_0\left(\vec{v}\cdot\vec{\delta}_y\right),$$

$$Y_2 + t_0\,(\vec{u}\cdot\vec{y}) = \sum_{i=1}^{N} (Y_2 + t_0\,(\vec{u}\cdot\vec{y}))^{[\![i]\!]} + t_0 u_n + t_0\left(\vec{u}\cdot\vec{\delta}_y\right),$$

$$A + Bt_0 = \sum_{i=1}^{N} (A + Bt_0)^{[\![i]\!]} + \delta_A + \delta_B t_0.$$

The parties output accept if

$$\begin{aligned} A + Bt_0 &= F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t_0) \\ &= (X_1 + t_0\,(\vec{u}\cdot\vec{x}))\,(Y_1 + t_0\,(\vec{v}\cdot\vec{y})) - (X_2 + t_0\,(\vec{v}\cdot\vec{x}))\,(Y_2 + t_0\,(\vec{u}\cdot\vec{y})) \end{aligned}$$

and reject otherwise.

The protocol is correct: Every party accepts if the shares are correct and if each party is honest. The protocol accepts a wrong instance of vectors if t_0 is a root of the degree 2 polynomial

$$(A + Bt) - F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t).$$

If at least one of the N parties is honest, the value of t_0 obtained in the protocol is random. Therefore, the probability of accepting a wrong pair of vectors $(\vec{x},\vec{y})$ is bounded by $\frac{2}{q^k}$. We also note that the protocol is $(N-1)$-private. Indeed, if the sharing of the values is performed uniformly at random, no information about the secret vectors $\vec{x},\vec{y}$ is revealed if at least one of the N parties is honest.

The view of each party. During the protocol execution, each party keeps a view of their computation. The view of party i consists of the following six elements:

$$\begin{aligned} \text{View}_i = (&t_0^{[\![i]\!]}, (X_1 + t_0\,(\vec{u}\cdot\vec{x}))^{[\![i]\!]}, (X_2 + t_0\,(\vec{v}\cdot\vec{x}))^{[\![i]\!]}, \\ &(Y_1 + t_0\,(\vec{v}\cdot\vec{y}))^{[\![i]\!]}, (Y_2 + t_0\,(\vec{u}\cdot\vec{y}))^{[\![i]\!]}, (A + Bt_0)^{[\![i]\!]}). \end{aligned}$$

Security improvement against adversarial selection of t_0. We want to slightly adapt the basic MPC protocol to be able to use it in an identification protocol. If the parties do not broadcast their shares simultaneously, a malicious

party can wait until he receives all the other shares of t_0 and choose his share such that t_0 is a root of the polynomial

$$(A + Bt) - F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t).$$

In this section, we adapt the protocol to prevent this. For a first new version, instead of randomly choosing the value of $t_0^{[\![i]\!]}$, party i uses the random oracle $\mathcal{H}$ on their private input shares and the public auxiliary values to compute

$$t_0^{[\![i]\!]} = \mathcal{H}_t\left(i \parallel \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B\right) \quad (3)$$

Here, the subscript t indicates that we use domain separation for the random oracle $\mathcal{H}$ to derive t_0. In particular, for any input string $\texttt{str}$, define $\mathcal{H}_t(\texttt{str}) \coloneqq \mathcal{H}(\text{“t commit”}\|\texttt{str})$.

In a second new version, the dealer distributes purely random shares of each private input to the provers. He achieves this by sending each party a random value r_i, which party i can use to derive the shares of

$$\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}.$$

To still have valid sharings, the dealer also publishes the auxiliary values

$$\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B.$$

By using the random oracle to obtain the shares of t_0, we can make sure that the value of t_0 was not maliciously chosen by a prover to be a root of the polynomial

$$(A + Bt) - F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t),$$

which would make the protocol succeed even if $(\vec{x}, \vec{y}) \notin \text{NSBC}[\vec{u}, \vec{v}]$.

3.2 Identification protocol using MPCitH

We can transform the multiparty protocol from Section 3.1 into an interactive Zero-Knowledge proof of knowledge of a solution $(\vec{x}, \vec{y}) \in \text{NSBC}[\vec{u}, \vec{v}]$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. In a basic version of this protocol, the prover creates sharings of all the inputs of the MPC protocol. In particular, the sharings

$$\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}$$

as well as the auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$. The prover executes the MPC protocol and commits to the views of each party. She sends the commitment hash $h = \mathcal{H}_v(\text{View}_1, \ldots, \text{View}_N)$ together with the values

$$X_1 + t_0\left(\vec{u} \cdot \vec{x}\right), X_2 + t_0\left(\vec{v} \cdot \vec{x}\right), Y_1 + t_0\left(\vec{v} \cdot \vec{y}\right), Y_2 + t_0\left(\vec{u} \cdot \vec{y}\right), A + Bt_0, t_0$$

and the auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$ to the verifier. The subscript v for the random oracle indicates domain separation, i.e. $\mathcal{H}_v(\texttt{str}) \coloneqq \mathcal{H}(\text{“view commit”}\|\texttt{str})$

for any string `str`. The verifier randomly chooses an index $i^* \in [N]$ and sends it to the prover. The prover then sends all the shares of the values $\vec{R_x}^{[\![i]\!]}$, $\vec{R_y}^{[\![i]\!]}$, ${X_1}^{[\![i]\!]}$, ${X_2}^{[\![i]\!]}$, ${Y_1}^{[\![i]\!]}$, ${Y_2}^{[\![i]\!]}$, $R_A^{[\![i]\!]}$, $R_B^{[\![i]\!]}$ for $i \neq i^*$ to the verifier. Using these values, the verifier can recompute View_i of each party for $i \neq i^*$. In particular, the values of $t_0^{[\![i]\!]}$, $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]}$, $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]}$, $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]}$, $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]}$, $(A + Bt_0)^{[\![i]\!]}$ for $i \neq i^*$. Note that for this basic protocol, the prover uses the random oracle to derive $t_0^{[\![i]\!]}$ as described in (3). For the missing shares corresponding to party i^*, the verifier can recompute the shares by using the received values from the verifier. In particular, she computes

$$
\begin{aligned}
(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i^*]\!]} &= (X_1 + t_0 (\vec{u} \cdot \vec{x})) - t_0 u_{n-1} - t_0 \left(\vec{u} \cdot \vec{\delta}_x\right) \\
&\quad - \sum_{i \neq i^*} (X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]}, \\
(X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i^*]\!]} &= (X_2 + t_0 (\vec{v} \cdot \vec{x})) - t_0 v_{n-1} - t_0 \left(\vec{v} \cdot \vec{\delta}_x\right) \\
&\quad - \sum_{i \neq i^*} (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]}, \\
(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i^*]\!]} &= (Y_1 + t_0 (\vec{v} \cdot \vec{y})) - t_0 v_n - t_0 \left(\vec{v} \cdot \vec{\delta}_y\right) \\
&\quad - \sum_{i \neq i^*} (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]}, \\
(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i^*]\!]} &= (Y_2 + t_0 (\vec{u} \cdot \vec{y})) - t_0 u_n - t_0 \left(\vec{u} \cdot \vec{\delta}_y\right) \\
&\quad - \sum_{i \neq i^*} (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]}, \\
(A + Bt_0)^{[\![i^*]\!]} &= (A + Bt_0) - \delta_A - \delta_B t_0 - \sum_{i \neq i^*} (A + Bt_0)^{[\![i]\!]}, \\
t_0^{[\![i^*]\!]} &= t_0 - \sum_{i \neq i^*} t_0^{[\![i]\!]}.
\end{aligned}
$$

With these values, the verifier can compute $h' = \mathcal{H}_v(\text{View}_1, \ldots, \text{View}_N)$, check if it matches the commitment hash h and accept or reject accordingly.

Reducing the communication. To reduce the required communication in the basic protocol described above, we can use punctured PRFs. The prover chooses a random key $\mathcal{K}$ and sets $r_i = \text{PRF}_{\mathcal{K}}(i)$ using a tree PRF. Using r_i, the prover pseudo-randomly generates all of her shares, namely:

$$
\vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}
$$

and computes the auxiliary values

$$
\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B.
$$

In this case, the prover uses the random oracle to derive $t_0^{[\![i]\!]}$ as described in (3). After receiving the value $i^* \in [N]$ from the verifier, the prover returns the punctured PRF key $\mathcal{K}_{i^*}$ to the verifier. With this, the verifier can compute every r_i except r_{i^*} and derive every input share of the MPC protocol except the shares of party i^*. By using the auxiliary values, the verifier can still compute the shares of the missing party and verify the commitment hash as before. We also note that the prover does not have to send the value of $A+Bt_0$, as it can be computed by the verifier from the other received values by using the formula

$$A + Bt_0 = (X_1 + t_0 (\vec{u} \cdot \vec{x})) (Y_1 + t_0 (\vec{v} \cdot \vec{y})) - (X_2 + t_0 (\vec{v} \cdot \vec{x})) (Y_2 + t_0 (\vec{u} \cdot \vec{y})) .$$

A malicious prover $\mathcal{P}'$ who does not have a valid sharing of the secret has to cheat by creating an invalid sharing in at least one position. This only remains undetected by the verifier if this position equals i^*. We have seen in Section 3.1 that the protocol has a false positive probability of $\frac{2}{q^k}$. Therefore, the soundness of the Zero-Knowledge protocol is $\frac{1}{N} + \frac{2}{q^k}$. We prove this formally in Theorem 2.

Hypercube technique. A common idea in MPCitH schemes is to use the hypercube technique introduced in [2]. Applied to the SBC identification protocol, it can reduce the number of scalar products that need to be computed during a protocol execution. To apply this technique, assume that N is a power of two, i.e. $N = 2^m$. Denote the j-th bit of the binary decomposition of an integer i by $B_j(i)$. The idea is to transform one instance of the protocol with $N = 2^m$ parties into m instances of the protocol with 2 parties. Consider the sharing of an element s of the form

$$s = \delta_s + \sum_{i=1}^{N} R_s^{[\![i]\!]}.$$

For any fixed value $j \in \{0, \ldots, m-1\}$, we see that

$$s = \delta_s + \sum_{B_j(i)=0} R_s^{[\![i]\!]} + \sum_{B_j(i)=1} R_s^{[\![i]\!]}.$$

After receiving the punctured PRF key, the verifier only knows one of the two sums. This reduces the amount of scalar products that are needed in the protocol execution: In the basic version, four scalar products are computed by each of the N parties. In particular, these scalar products are needed to compute the values

$$(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]}, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]}, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]}, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]} .$$

In the commitment step of the hypercube version, these $4N$ scalar product computations are replaced by $8m$ scalar product computations. For the verification, the verifier only has to compute $4m$ scalar products instead of $4(N-1)$. By using this technique, the amount of multiplications between elements of $\mathbb{F}_{q^k}$ and $\mathbb{F}_q$ during the protocol execution decreases. The computations can therefore be performed faster in this version.

Inputs:
- Public key $(\vec{u}, \vec{v})$ (Verifier)
- Secret key $(\vec{x}, \vec{y}) \in \mathrm{NSBC}[\vec{u}, \vec{v}]$ (Prover)

Step 1: Commitment
1. Sample $\mathcal{K} \leftarrow_{\$} \{0,1\}^{\lambda}$
2. For $i \in [N]$:
   - $r_i \leftarrow \mathcal{K}_i$
   - $X_1{}^{[\![i]\!]}, X_2{}^{[\![i]\!]}, Y_1{}^{[\![i]\!]}, Y_2{}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \xleftarrow{r_i}_{\$} \mathbb{F}_{q^k}$
   - $\vec{R_x}{}^{[\![i]\!]}, \vec{R_y}{}^{[\![i]\!]} \xleftarrow{r_i}_{\$} (\mathbb{F}_q)^n$
3. $X_1 \leftarrow \sum_{i=1}^{N} X_1^{[\![i]\!]}$
   $X_2 \leftarrow \sum_{i=1}^{N} X_2^{[\![i]\!]}$
   $Y_1 \leftarrow \sum_{i=1}^{N} Y_1^{[\![i]\!]}$
   $Y_2 \leftarrow \sum_{i=1}^{N} Y_2^{[\![i]\!]}$
4. $A \leftarrow X_1 Y_1 - X_2 Y_2$
   $B \leftarrow X_1 (\vec{v} \cdot \vec{y}) + Y_1 (\vec{u} \cdot \vec{x}) - X_2 (\vec{u} \cdot \vec{y}) - Y_2 (\vec{v} \cdot \vec{x})$
5. $\delta_A \leftarrow A - \sum_{i=1}^{N} R_A^{[\![i]\!]}$
   $\delta_B \leftarrow B - \sum_{i=1}^{N} R_B^{[\![i]\!]}$
   $\vec{\delta_x} \leftarrow \vec{x} - \sum_{i=1}^{N} \vec{R_x}{}^{[\![i]\!]}$
   $\vec{\delta_y} \leftarrow \vec{y} - \sum_{i=1}^{N} \vec{R_y}{}^{[\![i]\!]}$
6. $t_0^{[\![i]\!]} \leftarrow \mathcal{H}_t \left(i \parallel \vec{R_x}{}^{[\![i]\!]}, \vec{R_y}{}^{[\![i]\!]}, X_1{}^{[\![i]\!]}, X_2{}^{[\![i]\!]}, Y_1{}^{[\![i]\!]}, Y_2{}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B \right)$
7. $t_0 \leftarrow \sum_{i=1}^{N} t_0^{[\![i]\!]}$
8. $\mathrm{View}_i \leftarrow (t_0^{[\![i]\!]}, (X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i]\!]}, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i]\!]}, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i]\!]}, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i]\!]}, (A + t_0 B)^{[\![i]\!]})$
9. $h \leftarrow \mathcal{H}_v(\mathrm{View}_1, \ldots, \mathrm{View}_N)$
10. $\mathrm{msg} \leftarrow (X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0)$
11. Send $(h, \mathrm{msg}, \vec{\delta_x}, \vec{\delta_y}, \delta_A, \delta_B)$ to the verifier

Step 2: Challenge
1. Sample $i^* \leftarrow_{\$} [N]$
2. Send i^* to the prover

Step 3: Response
1. Compute punctured PRF key $\mathcal{K}_{i^*}$
2. Send $\mathcal{K}_{i^*}$ to the verifier

Step 4: Verification
1. For $i \in [N] \setminus \{i^*\}$:
   - Compute r_i from $\mathcal{K}_{i^*}$
   - Compute $X_1{}^{[\![i]\!]}, X_2{}^{[\![i]\!]}, Y_1{}^{[\![i]\!]}, Y_2{}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]}, \vec{R_x}{}^{[\![i]\!]}, \vec{R_y}{}^{[\![i]\!]}, t_0^{[\![i]\!]}$ from r_i and $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$
2. Compute $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{[\![i^*]\!]}, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^{[\![i^*]\!]}, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{[\![i^*]\!]}, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{[\![i^*]\!]}, (A + t_0 B)^{[\![i^*]\!]}, t_0^{[\![i^*]\!]}$ from msg and $\vec{\delta_x}, \vec{\delta_y}, \delta_A, \delta_B$
3. For $i \in [N]$: Compute View_i
4. Compute $h_1 = \mathcal{H}_v(\mathrm{View}_1, \ldots, \mathrm{View}_N)$
5. Output accept if and only if $h_1 = h$

Table 1. Identification protocol for the SBC problem

4 Signature Scheme

We can turn the Honest Verifier Zero Knowledge (HVZK) protocol from Section 3 into a signature scheme by using the Fiat-Shamir transform [12] with r repetitions to obtain a security level of $\lambda = r \log_2(N)$. The public key consists of the two vectors $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$ and the private key is $(\vec{x}, \vec{y}) \in \mathrm{NSBC}[\vec{u}, \vec{v}]$. The prover executes r rounds of the MPC protocol and commits to all of them by sending one commitment hash. After receiving the commitment hash to the r round protocol, the auxiliary values and the broadcast shares for each round, the verifier picks a random $i^* \in [N]$ for each round and sends each of these r values to the prover. The prover sends the punctured PRF keys for each round to the verifier, which allows the verifier to validate the commitment hash by recomputing the missing shares for each round.

4.1 Key and Signature size

Signature size. To avoid collisions in the hash function, we need outputs on 2λ bits. Since we use the GGM construction for the puncturable PRF, we can instead use a random global salt on λ bits and outputs of size λ for the hash function in the tree. By doing this, we reduce the PRF key size to $\lambda \log_2(N)$ bits. To avoid generic attacks, we also need the bitsize of the elements of $\mathbb{F}_{q^k}$ to be 2λ. Our assumption that $n \approx \frac{k}{2}$ implies that the bitsize of the elements of $(\mathbb{F}_q)^n$ is λ. The signature consists of the following elements:

- One hash value h corresponding to a global commitment to the r round protocol of size 2λ and one salt of size λ.
- One punctured PRF key for each round.
- The auxiliary values $\delta_A, \delta_B \in \mathbb{F}_{q^k}$ and $\vec{\delta}_x, \vec{\delta}_y \in (\mathbb{F}_q)^n$ for each round.
- The values $X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0 \in \mathbb{F}_{q^k}$ for each round.

The total communication cost in bits of the protocol is therefore

$$r \cdot (\lambda \log_2(N)) + 6r\lambda + 10r\lambda + 3\lambda = \lambda^2 + 16r\lambda + 3\lambda \text{ bits},$$

where we use the fact that $\lambda = r \log_2(N)$ in the equation. For $\lambda = 128$, we can for example choose $N = 2^{16}, r = 8$ and obtain a signature of size 4.144 KB.

Key size. The public key can be derived from a seed of size λ together with one element in $\mathbb{F}_{q^k}$. By using the seed, the elements $u_1, \ldots, u_n$ and $v_1, \ldots, v_{n-1}$ can be derived using a PRF and the missing element v_n is sent separately. The key size in bits is therefore 3λ. For $\lambda = 128$, this yields a public key size of 48 bytes.

4.2 Comparison with other MPCitH-based signature schemes

We compare our scheme with the current state of the art MPCitH-based signatures for the 128-bit security level and list the corresponding public key sizes

|pk| and signature sizes |sgn|. The SBC signature scheme yields smaller signature sizes and smaller public key sizes compared to previously known signature schemes based on the MPCitH paradigm.

Name	Year	|pk|	|sgn|
Syndrome Decoding in the Head [11]	2022	0.144 KB	8.481 KB
RYDE [6]	2023	0.086 KB	5.956 KB
MinRank in the Head [1]	2022	0.129 KB	5.673 KB
MIRA [3]	2023	0.084 KB	5.640 KB
Biscuit [5]	2023	0.050 KB	4.758 KB
SBC	2023	0.048 KB	4.144 KB

Table 2. Comparison of the SBC scheme with signatures from the literature for 128-bits security.

5 Formal schemes

In this section, we give a formal description of the identification scheme proposed in Section 3 with a target security of λ bits. We assume that the random oracle $\mathcal{H}$ has an output size of λ bits and that the size of the finite field $\mathbb{F}_{q^k}$ is $2^{2\lambda}$ to avoid generic attacks faster than 2^λ. We first give the description of the puncturable PRF which we use in our construction. In these functions, a salt of λ bits and internal nodes of size λ in the GGM tree are used. The number of leaves of the puncturable PRF is a power of two, i.e. $N = 2^m$. To obtain a finite field element, we expand the leaves of the tree to size 3λ before applying the map γ_{Field} to convert the leaves to field elements. Hereby, we assume that there exists an efficient bijective map γ_{Field} from $\mathbb{Z}_{q^k}$ to $\mathbb{F}_{q^k}$. Similarly, we assume there is an efficient bijective map γ_{Vec} from $\mathbb{Z}_{q^n}$ to $(\mathbb{F}_q)^n$. To obtain vectors in $(\mathbb{F}_q)^n$, where $n \approx k/2$, we apply the map γ_{Vec} to a leaf. Before applying these maps, we first expand the bitstring of the respective leaf and then map the extended bitstring to an integer using the function `ToInteger`. By doing this, we can obtain pseudorandom field elements and vectors. We use domain separation to be able to use the same random oracle $\mathcal{H}$ for each hash function.

Algorithm 1 is the implementation of the random oracle, Algorithms 2 to 10 are the implementations of the puncturable PRF. In Algorithms 11 to 14, we provide implementations of the four step identification protocol described in Section 3.2, i.e. Commitment, Challenge, Response and Verification.

Algorithm 1 Random Oracle $\mathcal{H}$

Input
A bitstring **Message**
Output
A bitstring of size λ

1: Let **Dict** be a global dictionary initially empty
2: **if** Message appears in Dict **then**
3: **return** Dict[Message]
4: **else**
5: Let **Value** be a uniformly random λ-bitstring
6: Define Dict[Message] = Value
7: **return** Value

Algorithm 2 Descendant

Input
An integer **Level** corresponding to the current level of the node in the tree
A bitstring **Salt** of size λ
A bitstring **Node** of size λ corresponding to the current node in the tree
A bitstring **SubPath** of the path from the root to a sibling of the node
Output
A bitstring of size λ corresponding to the sibling node along the subpath

1: **return** $\mathcal{H}$("Descendant"‖Level‖Salt‖Node‖SubPath)

Algorithm 3 RecursiveDescend

Input
An integer **Level** corresponding to the current level of the node in the tree
A bitstring **Salt** of size λ
A bitstring **Node** of size λ corresponding to the current node in the tree
A bitstring **Path** corresponding to the descend path from the root to the leaf
Output
A bitstring of size λ corresponding to the leaf node along the subpath

1: **if** Level is equal to the length of Path **then**
2: **return** Node
3: **else**
4: Let **SubPath** = Path[1 . . . Level + 1]
5: Let NextNode = **Descendant**(Level, Salt, Node, SubPath)
6: **return** **RecursiveDescent**(Level + 1, Salt, NextNode, Path)

Algorithm 4 ExpandNode

Input

A bitstring **Sep** for domain separation
A bitstring **Salt** of size λ
A bitstring **Leaf** of size λ corresponding to a leaf of the tree
A bitstring **PathLeaf** corresponding to the path from the root to the leaf

Output

A bistring of size 3λ

1: Let **ExpandedNode** $= \mathcal{H}(\text{“Expand0”}\|\text{Sep}\|\text{Salt}\|\text{Leaf}\|\text{PathLeaf})$
$\|\mathcal{H}(\text{“Expand1”}\|\text{Sep}\|\text{Salt}\|\text{Leaf}\|\text{PathLeaf})\|\mathcal{H}(\text{“Expand2”}\|\text{Sep}\|\text{Salt}\|\text{Leaf}\|\text{PathLeaf})$
2: **return** ExpandedNode

Algorithm 5 FieldEltFromLeaf

Input

A bitstring **Sep** for domain separation
A bitstring **Salt** of size λ
A bitstring **Leaf** of size λ corresponding to a leaf of the tree
A bitstring **PathLeaf** corresponding to the path from the root to the leaf

Output

An element of the finite field $\mathbb{F}$

1: Let **Value** $=$ `ToInteger`(**ExpandNode**(Sep, Salt, Leaf, PathLeaf))
2: **return** $\gamma_{\text{Field}}(\text{Value mod } |\mathbb{F}|)$

Algorithm 6 VecFromLeaf

Input

A bitstring **Sep** for domain separation
A bitstring **Salt** of size λ
A bitstring **Leaf** of size λ corresponding to a leaf of the tree
A bitstring **PathLeaf** corresponding to the path from the root to the leaf

Output

A vector in $\mathbb{F}^n$

1: Let **Value** $=$ `ToInteger`(**ExpandNode**(Sep, Salt, Leaf, PathLeaf))
2: **return** $\gamma_{\text{Vec}}(\text{Value mod } |\mathbb{F}^n|)$

Algorithm 7 PuncturedKey

Input

A bitstring **Salt** of size λ
A bitstring **RootKey** of size λ corresponding to the root of the tree
A bitstring **PathLeaf** corresponding to the path from the root to a leaf

Output

A bitstring corresponding to the punctured key along the path

---

Init **PuncturedKey** (as an empty array)
Let PuncturedKey[0] = PathLeaf
for Level **from** 1 **to** TreeDepth **do**
    Set **Path** = PathLeaf[1 . . . Level]
    Flip Bit Path[Level]
    Set PuncturedKey[Level] = **RecursiveDescend**(0, Salt, RootKey, Path)
return PuncturedKey

Algorithm 8 LeafFromPuncKey

Input

A bitstring **Salt** of size λ
A **PuncturedKey**
A bitstring **PathLeaf** corresponding to the path from the root to a leaf

Output

A bitstring of size λ corresponding to the leaf node along the path

---

Set **ForbiddenPath** = PuncturedKey[0]
if PathLeaf is equal to ForbiddenPath **then**
    return $\perp$
Let **Level** be the first bit position where PathLeaf differs from ForbiddenPath
Let **Leaf** = **RecursiveDescend**(Level, Salt, PuncturedKey[Level], PathLeaf)
return Leaf

Algorithm 9 FieldEltFromPuncKey

Input

A bitstring **Sep** for domain separation
A bitstring **Salt** of size λ
A **PuncturedKey**
A bitstring **PathLeaf** corresponding to the path from the root to a leaf

Output

An element of the finite field $\mathbb{F}$ corresponding to the leaf of PathLeaf

1: Set **ForbiddenPath** = PuncturedKey[0]
2: **if** PathLeaf is equal to ForbiddenPath **then**
3: **return** $\perp$
4: Let **Level** be the first bit position where PathLeaf differs from ForbiddenPath
5: Let **Leaf** = **RecursiveDescend**(Level, Salt, PuncturedKey[Level], PathLeaf)
6: **return FieldEltFromLeaf**(Sep, Salt, Leaf, PathLeaf)

Algorithm 10 VecFromPuncKey

Input

A bitstring **Sep** for domain separation
A bitstring **Salt** of size λ
A **PuncturedKey**
A bitstring **PathLeaf** corresponding to the path from the root to a leaf

Output

A vector in $\mathbb{F}^n$ corresponding to the leaf of PathLeaf

1: Set **ForbiddenPath** = PuncturedKey[0]
2: **if** PathLeaf is equal to ForbiddenPath **then**
3: **return** $\perp$
4: Let **Level** be the first bit position where PathLeaf differs from ForbiddenPath
5: Let **Leaf** = **RecursiveDescend**(Level, Salt, PuncturedKey[Level], PathLeaf)
6: **return VecFromLeaf**(Sep, Salt, Leaf, PathLeaf)

Algorithm 11 Step 1: Commitment

Let **RootKey** be a uniform random bitstring (of the expected size λ)
Let **Salt** be a uniform random bitstring (of the expected size λ)
Let **CommitString** be the empty string
Let $X_1, X_2, Y_1, Y_2, t_0 = 0$
for i **from** 1 **to** N **do**
    Let **Path** be the m-bit binary encoding of $i-1$
    Let $r_i = \textbf{RecursiveDescent}(0, \text{Salt}, \text{RootKey}, \text{Path})$
    Let ${X_1}^i = \textbf{FieldEltFromLeaf}(\text{“X1 derivation”}, \text{Salt}, r_i, \text{Path})$
    Let ${X_2}^i = \textbf{FieldEltFromLeaf}(\text{“X2 derivation”}, \text{Salt}, r_i, \text{Path})$
    Let ${Y_1}^i = \textbf{FieldEltFromLeaf}(\text{“Y1 derivation”}, \text{Salt}, r_i, \text{Path})$
    Let ${Y_2}^i = \textbf{FieldEltFromLeaf}(\text{“Y2 derivation”}, \text{Salt}, r_i, \text{Path})$
    Set $X_1 = X_1 + {X_1}^i$
    Set $X_2 = X_2 + {X_2}^i$
    Set $Y_1 = Y_1 + {Y_1}^i$
    Set $Y_2 = Y_2 + {Y_2}^i$
    Let $R_A^i = \textbf{FieldEltFromLeaf}(\text{“A derivation”}, \text{Salt}, r_i, \text{Path})$
    Let $R_B^i = \textbf{FieldEltFromLeaf}(\text{“B derivation”}, \text{Salt}, r_i, \text{Path})$
    Let $\vec{R_x}^i = \textbf{VecFromLeaf}(\text{“x derivation”}, \text{Salt}, r_i, \text{Path})$
    Let $\vec{R_y}^i = \textbf{VecFromLeaf}(\text{“y derivation”}, \text{Salt}, r_i, \text{Path})$
Let $A = X_1 Y_1 - X_2 Y_2$
Let $B = X_1 (\vec{v} \cdot \vec{y}) + Y_1 (\vec{u} \cdot \vec{x}) - X_2 (\vec{u} \cdot \vec{y}) - Y_2 (\vec{v} \cdot \vec{x})$
Let $\delta_A = A - \sum_{i=1}^{N} R_A^i$
Let $\delta_B = B - \sum_{i=1}^{N} R_B^i$
Let $\vec{\delta_x} = \vec{x} - \sum_{i=1}^{N} \vec{R_x}^i$
Let $\vec{\delta_y} = \vec{y} - \sum_{i=1}^{N} \vec{R_y}^i$
for i **from** 1 **to** N **do**
    Let $h_{t_0}^i = \mathcal{H}_t\left(i \parallel \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B\right)$
    Let $t_0^i = \gamma_{\text{Field}}(\texttt{ToInteger}(h_{t_0}^i) \mod |\mathbb{F}|)$
    Set $t_0 = t_0 + t_0^i$
for i **from** 1 **to** N **do**
    Let $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^i = X_1^i + t_0(\vec{u} \cdot \vec{R_x}^i)$
    Let $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^i = X_2^i + t_0(\vec{v} \cdot \vec{R_x}^i)$
    Let $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i = Y_1^i + t_0(\vec{v} \cdot \vec{R_y}^i)$
    Let $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i = Y_2^i + t_0(\vec{u} \cdot \vec{R_y}^i)$
    Let $(A + Bt_0)^i = R_A^i + R_B^i t_0$
    Let $\text{View}^i = ({t_0}^i, (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i, (A + Bt_0)^i)$
    Append encoding of View^i to CommitString
Let $X_1 + t_0 (\vec{u} \cdot \vec{x}) = \sum_{i=1}^{N} (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i + t_0 u_{n-1} + t_0(\vec{u} \cdot \vec{\delta}_x)$
Let $X_2 + t_0 (\vec{v} \cdot \vec{x}) = \sum_{i=1}^{N} (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i + t_0 v_{n-1} + t_0(\vec{v} \cdot \vec{\delta}_x)$
Let $Y_1 + t_0 (\vec{v} \cdot \vec{y}) = \sum_{i=1}^{N} (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i + t_0 v_n + t_0(\vec{v} \cdot \vec{\delta}_y)$
Let $Y_2 + t_0 (\vec{u} \cdot \vec{y}) = \sum_{i=1}^{N} (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i + t_0 u_n + t_0(\vec{u} \cdot \vec{\delta}_y)$
Let $\textbf{Commitment} = (\mathcal{H}_v(\text{CommitString}), (X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0), \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B)$
return Commitment

Algorithm 12 Step 2: Challenge

Let **Query** be a uniformly random integer in $[N]$
return Query

Algorithm 13 Step 3: Response

Input
An integer **Query** in $[N]$

Output
A punctured key
A salt

Import **RootKey** and **Salt** from Step 1
Convert Query to a binary **QueryPath**
return Salt$\|$**PuncturedKey**(Salt, RootKey, QueryPath)

Algorithm 14 Step 4: Verification

Input

A bitstring **Salt** of size λ and a **PuncturedKey** from Step 3

A **Commitment** from Step 1 of the form

$(h, X_1 + t_0 (\vec{u} \cdot \vec{x}), X_2 + t_0 (\vec{v} \cdot \vec{x}), Y_1 + t_0 (\vec{v} \cdot \vec{y}), Y_2 + t_0 (\vec{u} \cdot \vec{y}), t_0), \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B))$

Output

A boolean corresponding to accept or reject

Let **Query** = ToInteger(PuncturedKey[0])
Let **CommitVerifyString** be the empty string
for i **in** $[N] \setminus \{\text{Query}\}$ **do**
    Let **Path** be the m-bit binary encoding of $i - 1$
    Let r_i = **LeafFromPuncKey**(Salt, PuncturedKey, Path)
    Let ${X_1}^i$ = **FieldEltFromPuncKey**("X1 derivation", Salt, PuncturedKey, Path)
    Let ${X_2}^i$ = **FieldEltFromPuncKey**("X2 derivation", Salt, PuncturedKey, Path)
    Let ${Y_1}^i$ = **FieldEltFromPuncKey**("Y1 derivation", Salt, PuncturedKey, Path)
    Let ${Y_2}^i$ = **FieldEltFromPuncKey**("Y2 derivation", Salt, PuncturedKey, Path)
    Let R_A^i = **FieldEltFromPuncKey**("A derivation", Salt, PuncturedKey, Path)
    Let R_B^i = **FieldEltFromPuncKey**("B derivation", Salt, PuncturedKey, Path)
    Let $\vec{R_x}^i$ = **VecFromPuncKey**("x derivation", Salt, PuncturedKey, Path)
    Let $\vec{R_y}^i$ = **VecFromPuncKey**("y derivation", Salt, PuncturedKey, Path)
    Let $h_{t_0}^i = \mathcal{H}_t \left(i \parallel \vec{R_x}^{[\![i]\!]}, \vec{R_y}^{[\![i]\!]}, {X_1}^{[\![i]\!]}, {X_2}^{[\![i]\!]}, {Y_1}^{[\![i]\!]}, {Y_2}^{[\![i]\!]}, R_A^{[\![i]\!]}, R_B^{[\![i]\!]} \parallel \vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B \right)$
    Let $t_0^i = \gamma_{\text{Field}}(\texttt{ToInteger}(h_{t_0}) \mod |\mathbb{F}|)$
    Let $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^i = X_1^i + t_0(\vec{u} \cdot \vec{R_x}^i)$
    Let $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^i = X_2^i + t_0(\vec{v} \cdot \vec{R_x}^i)$
    Let $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i = Y_1^i + t_0(\vec{v} \cdot \vec{R_y}^i)$
    Let $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i = Y_2^i + t_0(\vec{u} \cdot \vec{R_y}^i)$
    Let $(A + Bt_0)^i = R_A^i + R_B^i t_0$
Let $A + Bt_0 = (X_1 + t_0 (\vec{u} \cdot \vec{x})) (Y_1 + t_0 (\vec{v} \cdot \vec{y})) - (X_2 + t_0 (\vec{v} \cdot \vec{x})) (Y_2 + t_0 (\vec{u} \cdot \vec{y}))$
Let $t_0^{\text{Query}} = t_0 - \sum_{i \neq \text{Query}} t_0^i$
Let $(X_1 + t_0 (\vec{u} \cdot \vec{x}))^{\text{Query}} = (X_1 + t_0 (\vec{u} \cdot \vec{x})) - t_0 u_{n-1} - t_0(\vec{u} \cdot \vec{\delta}_x) - \sum_{i \neq \text{Query}} (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i$
Let $(X_2 + t_0 (\vec{v} \cdot \vec{x}))^{\text{Query}} = (X_2 + t_0 (\vec{v} \cdot \vec{x})) - t_0 v_{n-1} - t_0(\vec{v} \cdot \vec{\delta}_x) - \sum_{i \neq \text{Query}} (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i$
Let $(Y_1 + t_0 (\vec{v} \cdot \vec{y}))^{\text{Query}} = (Y_1 + t_0 (\vec{v} \cdot \vec{y})) - t_0 v_n - t_0(\vec{v} \cdot \vec{\delta}_y) - \sum_{i \neq \text{Query}} (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i$
Let $(Y_2 + t_0 (\vec{u} \cdot \vec{y}))^{\text{Query}} = (Y_2 + t_0 (\vec{u} \cdot \vec{y})) - t_0 u_n - t_0(\vec{u} \cdot \vec{\delta}_y) - \sum_{i \neq \text{Query}} (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i$
Let $(A + Bt_0)^{\text{Query}} = (A + Bt_0) - \delta_A - \delta_B t_0 - \sum_{i \neq \text{Query}} (A + Bt_0)^i$
for i **from** 1 **to** N **do**
    Let $\text{View}^i = ({t_0}^i, (X_1 + t_0 (\vec{u} \cdot \vec{x}))^i, (X_2 + t_0 (\vec{v} \cdot \vec{x}))^i, (Y_1 + t_0 (\vec{v} \cdot \vec{y}))^i, (Y_2 + t_0 (\vec{u} \cdot \vec{y}))^i, (A + Bt_0)^i)$
    Append encoding of View^i to CommitVerifyString
if $\mathcal{H}_v$(CommitVerifyString) is equal to h **then**
    return true
else
    return false

Theorem 2. *Consider an instance* NSBC$[\vec{u}, \vec{v}]$ *of the NSBC problem for* $\vec{u}, \vec{v} \in \left(\mathbb{F}_{q^k}\right)^n$. *The identification protocol described above is a statistical honest verifier Zero-Knowledge proof of knowledge of* $(\vec{x}, \vec{y}) \in$ NSBC$[\vec{u}, \vec{v}]$ *with soundness* $1/N + 2/q^k$.

Proof. **Completeness.** By the discussion of the polynomial $F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t)$ in Section 3, we see that an execution of the protocol by an honest prover and an honest verifier is always accepted.
Statistical honest Verifier Zero-Knowledge. We construct a simulator $\mathcal{S}$ in the following way:

- Uniformly at random pick a RootKey, a Salt, a Query and auxiliary values $\vec{\delta}_x, \vec{\delta}_y, \delta_A, \delta_B$.
- Compute a PuncturedKey from RootKey, Salt and Query.
- Use the Verify algorithm to get the expected Commitment of the input (without using the Commitment hash in the algorithm).
- Output a simulated transcript of the protocol using PuncturedKey, Salt, Query, Commitment and the auxiliary values.

We claim that this distribution of this output is statistically indistinguishable from a valid transcript of the protocol. To see that, first note that if a valid prover runs the protocol with the same values for RootKey, Salt and Query, all the values for $X_1^i, X_2^i, Y_1^i, Y_2^i, \vec{R}_x^i, \vec{R}_Y^i, \vec{R}_A^i, \vec{R}_B^i, t_0^i$, $\left(X_1 + t_0 \left(\vec{u} \cdot \vec{x}\right)\right)^i$, $\left(X_2 + t_0 \left(\vec{v} \cdot \vec{x}\right)\right)^i$, $\left(Y_1 + t_0 \left(\vec{v} \cdot \vec{y}\right)\right)^i$, $\left(Y_2 + t_0 \left(\vec{u} \cdot \vec{y}\right)\right)^i, \left(A + Bt_0\right)^i$ for $i \neq$ Query are the same as in the run of the simulator $\mathcal{S}$. Only the values for the index Query are missing. There exists a configuration for these values for which the simulated and honest transcripts would be equal. By programming the random oracle at the respective points, the transcript produced by $\mathcal{S}$ could also come from a legitimate prover. These two transcripts only differ because the auxiliary values were chosen uniformly at random in the simulator but are obtained using a summation of outputs of the functions FieldEltFromLeaf and VecFromLeaf on random values for the prover. The outputs of the functions FieldEltFromLeaf and VecFromLeaf are statistically close to a random map. Therefore, the distribution obtained by the simulated transcript is statistically close to the honest one.
Proof of Knowledge. Let $\mathcal{P}^*$ be a prover that convinces a verifier with probability $\geq 1/N + 2/q^k + \varepsilon$ for a non-negligible ε. Then we can build an extractor which, with read access to the random oracle memory, can learn the secret key $(\vec{x}, \vec{y})$.
First, note that all random oracle queries made in many executions of the verify function are pairwise distinct with extremely high probability. Since we use a salt and domain separators, identical queries can only occur if identical salt values are selected. After $\mathcal{P}^*$ runs the commitment step of the protocol, the extractor can obtain the input to the random oracle that produced the commitment hash h. If this commitment hash was not produced by an honest protocol execution, the verification algorithm can only succeed with exponentially small probability. With the input string, the extractor can construct a candidate tree for the puncturable PRF used by the prover to derive the values for r_i. For each of these

values, the extractor knows the corresponding value of t_0^i from the input string. He searches through the oracle queries whose input could have come from a call of the Descendant function at the corresponding position by considering all queries to the random oracle that use the domain separator "t commit". He then uses the map γ_{Field} on this value modulo q^k to find the matching $h_{t_0}^i$. By searching though the oracle queries again, he can find the value of r_i that was used to obtain $h_{t_0}^i$.
There are three cases that can occur after using this strategy to reconstruct the tree. The extractor obtains the full GGM tree, one leaf is missing or more than one leaf is missing. We consider each of these cases separately:

- If more than one leaf is missing, the extractor can use the response from $\mathcal{P}^*$ from the third step to obtain a PuncturedKey. Using this key, the extractor can find at least one missing leaf in the tree. If this PuncturedKey did not come from an honest protocol execution, the probability that the partial tree obtained from this key matches the tree of the extractor is negligible.
- If no leaf is missing, the extractor can compute every share of every party in the MPC protocol. Using the auxiliary values, the extractor can compute the values of $\vec{x}$ and $\vec{y}$ used by $\mathcal{P}^*$. If $(\vec{x}, \vec{y}) \notin \text{NSBC}[\vec{u}, \vec{v}]$, the verification algorithm in step 4 outputs accept with probability $2/q^k$, which occurs if t_0 happens to be a root of the polynomial $F^{\vec{x},\vec{y}}_{X_1,X_2,Y_1,Y_2}(t)$.
- If one leaf is missing, then $\mathcal{P}^*$ might still be able to provide a punctured key which is consistent with the tree obtained by the extractor. However, this only accounts for a $1/N$ success probability. In this case, the extractor queries at another position to obtain the missing path from the root to the leaf if he gets a consistent response from $\mathcal{P}^*$ in step 3. This occurs with probability at least ε. If the extractor learns the full tree, he can again compute the secret key $(\vec{x}, \vec{y})$.

Therefore, the combination of the extractor and $\mathcal{P}^*$ can break the SBC problem in time $1/\varepsilon$.

Acknowledgements

This work has been supported by the European Union's H2020 Programme under grant agreement number ERC-669891.

References

1. Adj, G., Rivera-Zamarripa, L., Verbel, J.: MinRank in the head: Short signatures from zero-knowledge proofs. Cryptology ePrint Archive, Report 2022/1501 (2022), https://eprint.iacr.org/2022/1501
2. 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, Heidelberg (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_20

3. Aragon, N., Bidoux, L., Chi-Domínguez, J.J., Feneuil, T., Gaborit, P., Neveu, R., Rivain, M.: Mira: a digital signature scheme based on the minrank problem and the mpc-in-the-head paradigm (2023). https://doi.org/10.48550/ARXIV.2307.08575, https://arxiv.org/abs/2307.08575
4. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. Cryptology ePrint Archive, Report 2013/400 (2013), https://eprint.iacr.org/2013/400
5. Bettale, L., Perret, L., Kahrobaei, D., Verbel, J.: Biscuit: Shorter MPC-based Signature from PoSSo (Jun 2023), https://hal.sorbonne-universite.fr/hal-04156470, specification of NIST post-quantum signature
6. Bidoux, L., Chi-Domínguez, J.J., Feneuil, T., Gaborit, P., Joux, A., Rivain, M., Vinçotte, A.: Ryde: A digital signature scheme based on rank-syndrome-decoding problem with mpcith paradigm (2023). https://doi.org/10.48550/ARXIV.2307.08726, https://arxiv.org/abs/2307.08726
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, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0_15
8. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
9. Faugère, J.C.: A new efficient algorithm for computing gröbner bases without reduction to zero (F5). In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation. p. 75–83. ISSAC '02, ACM, New York, NY, USA (2002). https://doi.org/10.1145/780506.780516, https://doi.org/10.1145/780506.780516
10. Faugère, J.C., Safey El Din, M., Spaenlehauer, P.J.: Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1,1): Algorithms and complexity. Journal of Symbolic Computation **46**(4), 406–437 (2011). https://doi.org/https://doi.org/10.1016/j.jsc.2010.10.014, https://doi.org/10.1016/j.jsc.2010.10.014
11. 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, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_19
12. 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, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
13. Goldreich, O.: The Foundations of Cryptography - Volume 1: Basic Techniques. Cambridge University Press (2001)
14. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986)
15. Göloğlu, F., Joux, A.: A simplified approach to rigorous degree 2 elimination in discrete logarithm algorithms. Cryptology ePrint Archive, Report 2018/430 (2018), https://eprint.iacr.org/2018/430
16. 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
17. Johnson, C.R., Šmigoc, H., Yang, D.: Solution theory for systems of bilinear equations. Linear and Multilinear Algebra **62**(12), 1553–1566 (Oct

2013). https://doi.org/10.1080/03081087.2013.839670, https://doi.org/10.1080/03081087.2013.839670
18. Joux, A., , Pierrot, C.: Algorithmic aspects of elliptic bases in finite field discrete logarithm algorithms. Advances in Mathematics of Communications **0**(0), 0–0 (2022). https://doi.org/10.3934/amc.2022085, https://doi.org/10.3934/amc.2022085
19. Joux, A.: A new index calculus algorithm with complexity $L(1/4 + o(1))$ in small characteristic. In: Lange, T., Lauter, K., Lisonek, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 355–379. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-43414-7_18
20. Joux, A., Pierrot, C.: Improving the polynomial time precomputation of frobenius representation discrete logarithm algorithms - simplified setting for small characteristic finite fields. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 378–397. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45611-8_20
21. Joux, A., Pierrot, C.: Technical history of discrete logarithms in small characteristic finite fields. Designs, Codes and Cryptography **78**(1), 73–85 (Nov 2015). https://doi.org/10.1007/s10623-015-0147-6, https://doi.org/10.1007/s10623-015-0147-6
22. 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
23. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. Chapman & Hall/CRC, 2nd edn. (2014)
24. Lang, S.: Algebra. Springer New York (2002). https://doi.org/10.1007/978-1-4613-0041-0, https://doi.org/10.1007/978-1-4613-0041-0
25. Spaenlehauer, P.J.: Solving multi-homogeneous and determinantal systems: algorithms, complexity, applications. Phd thesis, Université Pierre et Marie Curie (Univ. Paris 6) (Oct 2012)