SD-in-the-Head rust implementation and optimization

Hugh Benjamin Zachariae, 201508592
Magnus Jensen, 201708626

Master's Thesis, Computer Science
November 2024
Advisor: Diego F. Aranha

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

▶in English. . . ◀

Resumé

▶in Danish. . . ◀

Acknowledgments

►. . . ◄

Hugh Benjamin Zachariae and Magnus Jensen
Aarhus, November 2024.

Contents

Chapter 1

Introduction

few pages. Introduce what we have done and how the paper is structured

1.1 Post-Quantum Cryptography and the NIST call to action

►motivate and explain the problem to be addressed◄

In 1994, the discovery of Shor's algorithm [24], provided a polynomial time algorithm for factoring integers and computing discrete logarithms using quantum computing. Back then, quantum computers where theories on paper. However, two decades later, significant developments in quantum technology, has brought us dangerously close to the point where quantum computers can solve foundational problems, which would break the security of modern standards of public-key cryptosystems. For example the Diffie-Hellman key exchange standard X25519, used in TLSv1.3, is not post-quantum secure.

In 2016, the National Institute of Standards and Technology (NIST), known for standardizing cryptographic primitives like AES and SHA, initiated a call for quantum resistant public-key cryptosystems [20]. This led to the standardization of three signature schemes, namely Dilithium [10], Falcon [13] and SPHINCS+ [7]. All three of the schemes are based on the hardness of structured lattice problems. This has led to the most recent call in 2022 for additional quantum resistant signature schemes based on different assumptions to diversify the standards available for the future.

With the introduction of public-key cryptosystems by Diffie and Hellman in 1976, we saw the introduction to schemes like RSA, that is based on factoring of large primes and ElGamal which is based on the discrete logarithm problem. While these protocols have seen wide adoption, there exists schemes based on coding theory. These are their own discipline of information theory which predates the introduction of public-key cryptography. Although the scheme is less prominent, McEliece introduced a PK scheme based on coding theory already in 1978 [17]. The McEliece PK scheme is based on the hardness of decoding random linear codes [6]. While the scheme never saw widespread adoption due to its large key sizes, the interest in code-based crypto-schemes have seen a resurgence as it is not vulnerable to known quantum attacks like

Shor's algorithm. Recent movements in code-based cryptosystems have shown that the performance needed for code based cryptographic schemes practically is within reach. Introductions of linear based codes on finite fields and MPC (Multi Party Computation) threshold schemes [4] show promising results and with the SD-in-the-Head scheme, such techniques are combined to create a post-quantum secure signature scheme.

▶Why does it need to be implemented? and why in Rust?◀

1.2 Our contributions

▶describe how we progressed through the project in an interesting way. What did we set out to do, what were the hurdles.◀

▶contributions: rust, issues we found.◀

Chapter 2

Preliminaries

In this section, we outline the mathematical and cryptographic foundations necessary to understand the SD-in-the-Head scheme and its components, providing a step-by-step progression from the basics to the construction of the protocol.

We begin by introducing the essential subroutines that underpin the SD-in-the-Head protocol. This includes **Galois Finite Fields** $\mathbb{F}_q$ [16, 22, 8] along with cryptographic primitives; **collision-resistant hash functions** and **Merkle tree commitment scheme** [5]. In addition, we provide short introductions to the foundational protocols of **Secure Multi-Party Computation** (MPC) and **Zero-Knowledge** (ZK) proofs, which serve as the underlying cryptographic protocols.

Next, we delve into the Syndrome Decoding (SD) problem, which forms the backbone of the SD-in-the-Head protocol. This includes an overview of the problem definition [2, 17, 6, 3], and its polynomial representation. This will support an understanding of how the protocol operates on hard instances of the SD problem.

Building upon these foundations, we lay the groundwork for the construction of the SD-in-the-Head protocol. First, we discuss the **Multi-Party-Computation-in-the-Head** (MPCitH) paradigm, explaining how it is leveraged to construct efficient ZK proofs [14]. Additionally we detail the role of the MPC preprocessing [4] and linear secret-sharing scheme [11], which dramatically improves the efficiency of the MPCitH construction, and verification. Next, we introduce the **Fiat-Shamir heuristic**, a powerful tool that transforms an interactive ZK protocol into a non-interactive signature scheme [12].

2.1 Galois Finite Field

Finite fields lays the basis for many cryptographic protocols and primitives. In essence, finite field theory describes the research into the relation and properties between numbers, while disregarding the numbers themselves. In order to define finite fields, we first need to define a *group* and a *finite field* (Definition 2.1.1 and Definition 2.1.2, respectively).

Definition 2.1.1. A group $\mathbb{G}$ is a tuple $(\mathbb{G}, \times, 1)$ where, $\mathbb{G}$ is the set of elements in the group, $\times$ is a binary operator, and 1 is the multiplicative identity. The group $\mathbb{G}$ must satisfy the following properties:

1. $\mathbb{G}$ is **closed** under $\times$. For all $a, b \in \mathbb{G}$, $a \times b \in \mathbb{G}$.
2. $\mathbb{G}$ is **associative** under $\times$. For all $a, b, c \in \mathbb{G}$, $(a \times b) \times c = a \times (b \times c)$.
3. $\mathbb{G}$ has an **identity** element 1 under $\times$. For all $a \in \mathbb{G}$, $a \times 1 = a$.
4. $\mathbb{G}$ has an **inverse** element for each element under $\times$. For all $a \in \mathbb{G}$, there exists an element a^{-1} such that $a \times a^{-1} = 1$.

Definition 2.1.2. A finite field $\mathbb{F}$ is a tuple $(\mathbb{F}, +, \times, 0, 1)$ where, $\mathbb{F}$ is the set of elements in the field, $+$ and $\times$ are the addition and multiplication operators, and 0 and 1 are the additive and multiplicative identities. The field $\mathbb{F}$ must satisfy the following properties:

1. $+$ and $\times$ are **commutative**, **associative** and $\times$ is **distributive** over $+$.
   (a) $\forall a, b \in \mathbb{F} : a + b = b + a$ and $a \times b = b \times a$ (commutative)
   (b) $\forall a, b, c \in \mathbb{F} : (a + b) + c = a + (b + c)$ and $(a \times b) \times c = a \times (b \times c)$ (associative)
   (c) $\forall a, b, c \in \mathbb{F} : a \times (b + c) = a \times b + a \times c$ (distributive)
2. $(\mathbb{F}, +, 0)$ forms an additive group
3. $(\mathbb{F} \setminus \{0\}, \times, 1)$ forms a multiplicative group

The most commonly known finite fields are the prime fields $\mathbb{F}p$, widely used in protocols like RSA. These fields can be extended into $\mathbb{F}p^n$, containing p^n elements. However, prime fields are not efficient for iterating over large numbers of digits, a common requirement in protocols such as SD-in-the-Head. For better efficiency, we require a field that can be represented using bits or bytes. One method to achieve this is by constructing fields commonly known as Galois fields. Consider the simplest prime field $\mathbb{F}_2 = (0, 1, \texttt{XOR}, \texttt{AND}, 0, 1)$. It is straightforward to verify that this field satisfies the properties outlined in Definition 2.1.2.

The Galois field $\mathbb{F}2^n$ is an extension field of $\mathbb{F}2$, defined as $\mathbb{F}2^n$. This field is constructed by selecting an irreducible polynomial $p(x)$ of degree n with coefficients in $\mathbb{F}2$. The elements of $\mathbb{F}2^n$ are represented as polynomials of degree at most $n - 1$ with coefficients in $\mathbb{F}2$. Consequently, $\mathbb{F}2^n$ contains exactly 2^n elements. The polynomial $p(x)$ ensures that $\mathbb{F}_{2^n}$ forms a valid field as it defines the modular reduction in multiplication.

Each element in $\mathbb{F}_{2^n}$ is represented as a polynomial of degree at most $n - 1$ with coefficients in $\mathbb{F}_2$.

$$a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

where $a_i \in \mathbb{F}_2$ This corresponds directly into the 8-bit binary representation $a_{n-1}a_{n-2} \ldots a_1a_0$. The addition and multiplication operations within $\mathbb{F}2^n$ are defined as follows:

- **Addition**: The addition of two elements $a, b \in \mathbb{F}_{2^n}$ is defined as the bitwise XOR of the two elements.

- **Multiplication**: The multiplication of two elements $a, b \in \mathbb{F}_{2^n}$ is defined as the multiplication of the two polynomials modulo the irreducible polynomial $p(x)$.

A common choice for the irreducible polynomial is the Rijndael polynomial $p(x) = x^8 + x^4 + x^3 + x + 1$ used in the AES protocol [8]. This field is commonly known as $\mathbb{F}_{2^8}$. The size of the field allows us to represent elements as bytes, which is efficient for iterating over large numbers of digits and it contains all the elements of a byte.

Field extension

An advantageous property of Galois fields is that they can easily be extended, like $\mathbb{F}_2$ to $\mathbb{F}_{2^8}$. To ensure security in the SD-in-the-Head protocol, the field $\mathbb{F}_{2^8}$ is extended to encompass the field 32-bit unsigned integers. This can be done as a *tower extension* by first building a degree-2 extension to $\mathbb{F}_{2^{16}}$ using the irreducible polynomial $q(x) = x^2 + x + 32$ and finally extending this field to $\mathbb{F}_{2^{32}}$ using the irreducible polynomial $r(x) = x^2 + x + 32(x)$. While you could create the final field independently, the tower extension allows us to reuse the construction of the intermediate field. Addition and multiplication in the extension fields are represented as the following

$$
\begin{aligned}
a &= a_0 + a_1 x && a \in \mathbb{F}_{q^\eta}, a_0, a_1 \in \mathbb{F}_q \\
b &= b_0 + b_1 x && b \in \mathbb{F}_{q^\eta}, b_0, b_1 \in \mathbb{F}_q
\end{aligned}
$$

$$
\begin{aligned}
a + b &= (a_0 + a_1 x) + (b_0 + b_1 x) = (a_0 + b_0) + (a_1 + b_1)x \\
a * b &= (a_0 + a_1 x) * (b_0 + b_1 x) \\
&= a_0 b_0 + (a_0 b_1 + a_1 b_0)x + a_1 b_1 x^2 \\
&= a_0 b_0 + (a_0 b_1 + a_1 b_0)x + a_1 b_1 (x + 32) \\
&= a_0 b_0 + 32 a_1 b_1 + (a_0 b_1 + a_1 b_0 + a_1 b_1)x
\end{aligned}
$$

Polynomial evaluation

►Should this be moved to the specification section?◄ For polynomial evaluation in the multiparty computation algorithm, seen in Section 2.2.3, we need to evaluate a polynomial $P(x)$ at a point $r \in F_q^\eta$. This is done by evaluating the polynomial at each coefficient and summing the results. The polynomial $P(x)$ is defined as

$$
\begin{aligned}
&\textstyle\bigcup_{|Q|} (F_q)^{|Q|} \times F_q^\eta \to F_q^\eta \\
&\qquad Q(r) = \textstyle\sum_{i=0}^{|Q|} Q_i \cdot r^{i-1} \qquad Q_i \in F_q, r \in F_q^\eta \qquad (2.1)
\end{aligned}
$$

2.2 Cryptographic Primitives and protocols

2.2.1 Collision Resistant Hash Functions

We denote a hash function by a generator $\mathcal{H}$ which on input of a security parameter k outputs a function $h : \{0,1\}^* \to \{0,1\}^k$. We denote a hash function h to be cryptographically secure if it satisfies the following properties

Lemma 2.2.1 (Preimage Resistance). *Given a hash function h and a hashed message c, there exist no PPT algorithm that can find a message m such that $h(m) = c$ with non-negligible probability.*

Lemma 2.2.2 (Collision Resistance). *Given a hash function h, there exist no PPT algorithm that can find two distinct messages m, m' such that $h(m) = h(m')$ with non-negligible probability.*

The consensus among researchers indicates that for combinatorial problems, such as those underlying SHA2 and the more recent SHA3, quantum computing is expected to reduce the security level from k to $k/2$ for preimage resistance through approaches like Grover's algorithm [18]. In contrast, no known methods have been proposed to reduce the security level for collision resistance. Consequently, protocols that depend on collision-resistant hash functions, such as Merkle Signature or SD-in-the-Head, remain a viable choice for post-quantum secure protocols.

In proving the security of hash functions, researchers often resort to models like the Quantum Random Oracle Model (QROM), while there is significant work being done to further prove the future security of hash functions [1].

2.2.2 Merkle Tree Commitment Scheme

A commitment scheme allows a *prover* to commit to a value v and later prove that the revealed value is identical to the one initially committed to. The scheme satisfies the following two properties:

Lemma 2.2.3 (Binding). *Given a commitment c to a value v, a dishonest prover can only change the committed value and have the verifier accept the new value with negligible probability.*

Lemma 2.2.4 (Hiding). *Given a commitment c to a value v, the verifier can only recover the value v with negligible probability.*

A variant of such a scheme which allows for partial opening, is the *Merkle Tree Commitment scheme.*

Let $v_1, \dots, v_n$ be the values to be committed, and let h denote a cryptographically secure hash function. The leaves of the Merkle tree, $a_{0,1}, \dots, a_{0,n}$, are initialized as the hash values of the input values:

$$a_{0,i} = h(v_i), \quad \text{for } i = 1, \dots, n.$$

Each parent node $a_{i,j}$, where i represents the level of the tree and j denotes the parent index at that level, is computed as the hash of its two child nodes:

$$a_{i,j} = h(a_{i-1,2j} \,|\, a_{i-1,2j+1}),$$

where $|$ indicates concatenation of the child nodes.

The root node, $a_{h,0}$, where h is the height of the tree, serves as the commitment value and is sent to the verifier.

To reveal a committed value v, the prover computes the authentication path for the value's leaf node $a_{0,p}$, where p is the index of the value. The authentication path A consists of the sequence of sibling nodes required to reconstruct the root. The first element in the authentication path, A_0, is the sibling of the leaf node:

$$A_0 = \begin{cases} a_{0,p-1}, & \text{if } p \text{ is odd,} \\ a_{0,p+1}, & \text{if } p \text{ is even.} \end{cases}$$

This element is recorded as $auth_0$.

For higher levels, each subsequent element in the path is determined as:

$$A_i = h(A_{i-1} \,|\, auth_{i-1}),$$

where $auth_i$ is the sibling node of A_{i-1} at the same level.

The prover transmits the authentication path A and the index p of the selected value to the verifier. The verifier checks the validity of the commitment by using the provided leaf value and authentication path to recompute the root of the Merkle tree.

Security properties

The hiding property of the protocol is ensured by the use of a secure hash function, as the prover only shares the root of the tree. If the adversary were able to recover any value v, this would break the preimage resistance Lemma 2.2.1 of the hash function.

Conversely, the binding property Lemma 2.2.3 is upheld by the fact that the adversary would have to find a collision for the hash function to change the commitment value.

A notable property of the Merkle tree is its ability to open a partial subset of the committed values by providing only the necessary authentication paths for the values in the subset. The verifier can still validate these values using a single global root value.

This property is particularly advantageous in protocols like the SD-in-the-Head threshold protocol, where only a subset of the committed values needs to be opened while keeping the rest concealed. By leveraging this feature, the prover can efficiently prove the validity of the subset without revealing the entire set of commitments.

2.2.3 Secure Multi-Party Computation

Secure Multi-Party Computation (MPC) refers to a cryptographic protocol enabling multiple parties to jointly compute a function f represented by a circuit C, ensuring that no information about the individual inputs is revealed beyond what can be inferred from the output of C. In the following, sections we will denote a MPC protocol by Π_f for an n-party functionality $f(x, w_1, \ldots, w_n)$, a public input x and secret inputs w_i of the party P_i. The goal is to compute the function f on the inputs of the n parties while upholding the following properties [9]

- **Correctness**: The output of the protocol is the correct evaluation of the function f on the inputs of the parties w_i and the public input x
- **Privacy**: The protocol ensures that no partylearns anything about the inputs of the other parties beyond what can be inferred from the output of the function.

In terms of security needed for the SD-in-the-Head protocol, we define the following security requirements for the MPC protocol Π_f

- **Semi-honest security**: The protocol is secure against semi-honest adversaries, where parties follow the protocol but may attempt to learn information from the messages they receive.
- **Low-threshold security**: The protocol is secure against a coalition of up to ℓ parties, where ℓ is the threshold. This is also known as a ℓ-private MPC protocol.

We denote the view V_i of a party P_i in the protocol as $(i, x, r_i, w_i, (m_1, \dots, m_j))$ where r_i is the randomness used by P_i, w_i is the secret share, and $(m_1, \dots, m_j)$ is the messages received by P_i in the first $j + 1$ rounds of the protocol. We say that two views V_i, V_j are *consistent* according to the public input x if the views are identical given the public input x.

2.2.4 Zero-Knowledge Proofs

This section will only give a brief introduction to *two-party interactive* Zero-Knowledge (ZK) schemes for a *prover* and a *verifier*. We denote a ZK $\Pi_{\mathcal{R}}$ for some NP relation $\mathcal{R}(x, w)$. Let x be a public statement in **NP** and w be a witness such that $(x, w) \in \mathcal{R}$ [11].

Definition 2.2.1. Let x be a statement of language L in **NP**, and $W(x)$ the set of witnesses for x such that the following relation holds:

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

2.3 Syndrome Decoding Problem

The SD-in-the-Head protocol is built on the computational hardness of the Syndrome Decoding (SD) problem for random linear codes over a finite field (see Section 2.1). This protocol uses a variant of the SD problem, referred to as the *coset weights* problem, first introduced by Berlekamp and McEliece in 1978 [6]. The problem is defined as follows:

Definition 2.3.1. Given $H \in \mathbb{F}_q^{(m-k)\times m}$ and $y \in \mathbb{F}_q^{m-k}$. The problem is to find $x \in \mathbb{F}_q^m$ s.t. $\mathrm{wt}(x) \leq w$ and $Hx = y$.

Generating such an instance is straightforward: one can construct a uniformly random parity-check matrix H and a codeword x (with $wt(x) \leq w$), and then compute the syndrome $y = Hx$. In the SD-in-the-Head protocol, the values of the matrix H and the syndrome y are elements of the finite field $\mathbb{F}_q$ which we will explain further in Section 2.1. The SD problem is well-known to be NP-complete for random instances [6] also referred to as the general decoding problem.To illustrate the computational difficulty, solving the problem using brute force would require $O(\binom{m}{w}q^w)$ operations, which is computationally infeasible for large m and k.

Standard form of the parity-check matrix

To improve the performance and reduce the key size of the protocol, its possible to utilize the fact that the matrix H can be in standard form. $H = (H'|I_{m-k})$ Where $H' \in \mathbb{F}_q^{(m-k)\times k}$. This allows for the following representation of the syndrome:

$$y = Hx = H'x_a + x_b \tag{2.2}$$

with $x = (x_a|x_b)$. This improves the performance of the algorithms used in the SD-in-the-Head protocol in the following ways

- At the MPC layer, we only need to reveal one share x_a. Due to the fact that the other share x_b can simply be recomputed by $x_b = y - H'x_a$.
- By linearity of the above relation one only needs to send x_a in order to recover the SD instance. So from a sharing of x_a one can check the correctness of the SD instance.

Polynomial representation of SD

In order to convert the SD problem into a form that allows for *Multi Party Computation in the Head* (MPCitH, which we will exlpain further in Section 2.4) the SD-in-the-Head protocol is based on three (witness-dependent) polynomials S, Q and P, and one public polynomial F. These are used for checking the correctness of the SD solution by verifying the following relation:

$$S \cdot Q = P \cdot F \tag{2.3}$$

Let $f_1, \dots, f_q$ denote elements of $\mathbb{F}_q$, then the polynomials are defined as:

- $S \in \mathbb{F}_q[X]$ is the Lagrange interpolation of the coordinates of x, such that it matches $S(f_i) = x_i$ for $i \in [1 : m]$ and has degree $\deg(S) \leq m - 1$
- $Q \in \mathbb{F}_q[X]$ is defined by $Q(X) = \prod_{i \in E}(X - f_i)$. $E \subset [1 : m]$ with order $|E| = w$, such that E contains the non-zero coordinates of x. Q has degree $\deg(Q) = w$.
- $P \in \mathbb{F}_q[X]$ is defined as $P = S \cdot Q/F$ and has degree $\deg(P) \leq w - 1$. By definition the polynomial F divides $S \cdot Q$.

- $F \in \mathbb{F}_q[X]$ is the *vanishing polynomial* of the set $f_1, \ldots, f_m$ also defined as $F(X) = \prod_{i \in [1:m]}(X - f_i)$ and has degree $\deg(F) = m$.

We can now look at the relation in Equation 2.3. If we look at the left-hand side, which has the following property by design $S \cdot Q(f_i) = 0 \;\forall\; f_i \in [1:m]$. This comes from the fact that the polynomial $S(f_i) = 0$ whenever $x_i = 0$, as it is the lagrange interpolation of x. Furthermore, the polynomial $Q(f_i)$ is zero whenever f_i is a non-zero coordinate of x, which follows from the definition of Q.

For the right-hand side, the polynomial F is the vanishing polynomial for the set $f_1, \ldots, f_m$, so $F(f_i) = 0 \;\forall\; f_i \in [1:m]$. The polynomial P is needed to match the degree of $S \cdot Q$. As the degree of F is $m \leq \deg(S \cdot Q) \leq m + w - 1$.

It is now apparent that if the prover can convince the verifier that they know of polynomials P, Q such that $S \cdot Q = F \cdot P = 0$ at all points $f_i \in [1:m]$. The following must hold, either $S(f_i) = x_i = 0$ or $Q(f_i) = 0$. However, the polynomial Q can be zero in at most w points based on the degree, this means that S is non-zero in at most w points, based on the construction, which in turn implies that x has weight as most w.

With this, we can define the soundness of the MPC protocol as follows:

$$wt(x) \leq w \Leftrightarrow \exists P, Q \text{ with } \deg(P) \leq w-1 \text{ and } \deg(Q) = w \text{ s.t. Equation 2.3 holds} \tag{2.4}$$

We can now share a witness (x_a, Q, P). Now based on the relation Equation 2.2, x can be locally computed along with the polynomial S, this can then be used to run an equality test for the relation $S \cdot Q = P \cdot F$.

False positive probability

The equality test from Equation 2.3 has a small probability of false positives, denoted p. To reduce this probability, the relation is evaluated at random points $\{r_k \in \mathbb{F}_q\}_{k \in [t]}$. By the Schwartz-Zippel lemma, the probability of a false positive is bounded by $p \leq \frac{t}{q}$, where q is the size of the finite field $\mathbb{F}_q$. In short, this makes it unlikely that the relation will hold for all points r_k if the relation is not sound according to Equation 2.4. Furthermore, we can tweak the parameters t and q to reduce p. We will explain the choice of field size q in Section 2.1.

2.4 MPC-in-the-Head

The SD-in-the-Head protocol construction is based on the *Multi-Party Computation in the Head* (MPCitH) framework. In this section we will give an introduction to the framework and how it can be used to construct ZK proofs, which in turn can be combined with the Fiat-Shamir heuristic to create a signature scheme.

The MPC-in-the-Head (MPCitH) framework, introduced by [14], builds upon these techniques to construct generic zero-knowledge protocols (ZK). A ZK protocol allows a *prover* to convince a *verifier* of the validity of a statement without revealing any information about the inputs to the statement. The framework provides a versatile method of constructing protocols that are quantum safe as its security relies on assumptions that are still believed to be quantum secure. Namely, commitment schemes and hash functions [11] which has no known quantum algorithms that break their security.

We will give insight into the basic construction suggested by Ishai et al [14].

Definition 2.4.1. Given a semi-honest ℓ-private MPC protocol Π_f with perfect correctness, a relation $\mathcal{R}$ for some public statement x, a witness w. Let w_i be an additive secret share of $[[w]]$ for the party P_i. Let f be a n-party functionality $f(x, w_1, \ldots, w_n) = \mathcal{R}(x, w)$, i.e. $f(x, w)$ accepts if $(x, w) \in \mathcal{R}$.

1. The prover builds a random sharing of $[[w]] = w_1, \ldots, w_n$. Then
   - Simulates the outputs of the MPC protocol Π_f on the inputs $(x, w_1, \ldots, w_n)$ and the randomness $r_1, \ldots, r_n$.
   - Prepares views $V_1, \ldots, V_n$ of the parties in the protocol Π_f.[1]
   - Commits to each view V_i using a secure commitment scheme, and sends $(\text{Commit}(V_1), \ldots, \text{Commit}(V_n))$ to the verifier.
2. The verifier picks ℓ random distinct indices $i \in [n]$ and sends them to the prover.
3. The prover opens the commitments into the views V_i and sends the openings to the verifier.
4. The verifier accepts if and only if:
   (a) The views are valid according to the commitment scheme.
   (b) The views are consistent according to the public input x.
   (c) The views output 1 according to $\mathcal{R}$ meaning that $\mathcal{R}(x, w) = 1$.

We see the following properties of the protocol:

Lemma 2.4.1 (Completeness [14]). *A boolean function is said to be complete if it depends on all inputs of the function. Given an honest prover, $\mathcal{R}(x, w) = 1$ and the correctness of Π_f, all outputs of P_i are one and all views are consistent.*

Lemma 2.4.2 (Soundness [14]). *If the statement x is false, i.e. $x \notin L$, then $\mathcal{R}(x, w) = 0$ for all w. By the correctness of Π_f, the output of all parties must be 0. If the verifier accepts, then the prover must have created ℓ inconsistent views. This happens with probability at most $p = 1/\binom{n}{\ell}$. The error probability can be reduced to 2^{-k} by repeating the protocol $O(kn^2)$ times.*

Lemma 2.4.3 (Zero-Knowledge [14]). *The verifier only sees ℓ views, and therefore from the definition of Π_f learns nothing of the secret witness w.*

[1]Remember that the views include the inputs, randomness and messages received by the parties.

The basic MPC-in-the-Head protocol provides a foundation for constructing ZK proofs for any NP relation and has been shown to produce relatively efficient ZK protocols [11, 4, 15]. Variants of the protocol have demonstrated reductions in communication complexity, though at the cost of decreased computational efficiency on the prover's side, as these constructions require a substantial increase in the number of parties N [14, 11], which subsequently leads to heightened computational complexity.

Recent advancements, however, offer promising improvements on both fronts. First, the integration of *MPC preprocessing*[15] enables the instantiation of an MPC-in-the-Head instance where communication complexity is independent of N, thereby facilitating increased security while maintaining low communication costs. Additionally, employing *Linear Secret Sharing Schemes* (LSSS)[11] in place of additive sharing significantly enhances computational efficiency, as the computations for both prover and verifier become bounded by the threshold ℓ. The incorporation of these techniques ultimately paves the way for the development of the SD-in-the-Head protocol.

2.4.1 Proving the SD relation using Beaver triples

In the following section we introduce the specific MPC protocol employed to verify the SD polynomial relation $P \cdot F = S \cdot Q$ (Section 2.3).

Consider the multiplication MPC protocol by [4] which verifies the relation $z = x \cdot y$ for $x, y, z \in \mathbb{F}_q$.

Definition 2.4.2. Given an input triple $(x, y, z) \in \mathbb{F}$ random shared triple $([[a]], [[b]], [[c]]) \in \mathbb{F}$, it is possible to verify the correctness of the statement $z = x \cdot y$ without revealing any information on either of the input.

1. The parties generate a random $\epsilon \in \mathbb{F}$.
2. The parties locally set $[[\alpha]] = \epsilon[[x]] + [[a]], [[\beta]] = [[y]] + [[b]]$.
3. The parties run $\texttt{open}([[\alpha]])$ and $\texttt{open}([[\beta]])$ to obtain α and β.
4. The parties locally set $[[v]] = \epsilon[[z]] - [[c]] + \alpha \cdot [[b]] + \beta \cdot [[a]] - \alpha \cdot \beta$.
5. The parties run $\texttt{open}([[v]])$ to obtain v and accept iff $v = 0$.

Observe that if both triples are correct multiplication triples (i.e., $z = xy$ and $c = ab$) then the parties will always accept since

$$\begin{aligned}
v &= \epsilon \cdot z - c + \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta && (2.5)\\
&= \epsilon \cdot xy - ab + (\epsilon \cdot x + a)b + (y + b)a - (\epsilon \cdot x + a)(y + b) && (2.6)\\
&= \epsilon \cdot xy - ab + \epsilon \cdot xb + ab + ya + ba - \epsilon \cdot xy - \epsilon \cdot xb - ay - ab && (2.7)\\
&= (\epsilon \cdot xy - \epsilon \cdot xy) + (ab - ab) + (\epsilon \cdot xb - \epsilon \cdot xb) + (ya - ay) + (ba - ab) && (2.8)\\
&= 0 && (2.9)
\end{aligned}$$

Lemma 2.4.4. *If $([[a]], [[b]], [[c]])$ or $([[x]], [[y]], [[z]])$ is an incorrect multiplication triple then the parties output* $\texttt{Accept}$ *in the sub-protocol above with probability* $\frac{1}{|\mathbb{F}|}$.[2]

[2]A full proof of this can be found in [4].

2.4.2 MPC preprocessing model

▶This part can quickly become tedious, so we need to make sure that we are not going too deep into the details.◀

We need to read into [15] and how they use preprocessing to improve the efficiency of MPCitH from [14]. Se page 3 in [15] for preprocessing phase.

> As a consequence of being able to rely on preprocessing, the space of possible protocols Î we can use is greatly expanded. In particular, we find that we obtain much shorter proofs by using an n-party protocol (secure against semi-honest corruption of allbut-one of the parties) with n as high as 64. The ability to rely on preprocessing is critical here: the communication complexity of traditional MPC protocols (that do not rely on preprocessing) with security against all-but-one corruption is quadratic in the number of parties, but by relying on preprocessing we can obtain communication complexity independent of n. Further optimizations and specific parameter choices for the above proof are discussed in the remainder of the paper.

2.5 Linear Secret Sharing Schemes (LSSS)

Secret sharing schemes (SSS) are a type of cryptographic protocol that allows for the distribution of a secret amongst a group of participants. The secret can only be reconstructed when a sufficient number of shares are combined together. The threshold variant of the SD-in-the-Head protocol relies on a low-threshold linear secret sharing scheme (*LSSS*). Threshold secret sharing schemes allow for the reconstruction of a secret from a subset of shares of length ℓ, where ℓ is the threshold. The threshold allows for the SD-in-the-Head protocol to be more communication efficient, as the amount of shares needed to reconstruct the secret is low.

Definition 2.5.1. *S is a (ℓ, n) threshold SSS if it satisfies the following properties*:

- **Share generation**: Given a secret s, the scheme generates n shares $\texttt{share}(s) = [[s]] = [[s_1, s_2, \ldots, s_n]]$.
- **Reconstruction**: Given a subset of shares $[[s']]$ of size ℓ, the scheme can reconstruct the secret $s = \texttt{open}([[s']])$.

Linear secret charing schemes, or $(+,+)$-homomorphic schemes, refers to secret sharing schemes that are linearly homomorphic over some field $\mathbb{F}$ (say the galois field `GF256` described in Section 2.1). This means that given shares $[[a]]$ and $[[b]]$ we have that

Definition 2.5.2. A (ℓ, n) threshold SSS is $(+,+)$-homomorphic if for any two secrets s_1 and s_2 and their shares $[[s_1]]$ and $[[s_2]]$, the sum of the shares $[[s_1]] + [[s_2]] = [[s_{11} + s_{21}, s_{12} + s_{22}, \ldots, s_{1n} + s_{2n}]]$ is equal to the share of the sum of the secrets $[[s_1 + s_2]]$ for the same subset of shares.

2.5.1 Shamir's Secret Sharing

Shamir's Secret Sharing scheme [23]. Shamir's Secret Sharing is a method for distributing a secret amongst a group of participants, each of which is given a share of the secret. The secret can only be reconstructed when a sufficient number of shares are combined together.

For a secret s and a given security threshold ℓ, the share $[[s]]$ is generated by sampling a random polynomial of degree $t-1$ with s as the free coefficient. Each participant is given a share of the polynomial evaluated at a point. The secret can be reconstructed by interpolating the polynomial from a sufficient number of shares.

The (t, n) Shamir's polynomial based secret sharing scheme is $(+, +)$-homomorphic in which the addition of two polynomials secrets equals the Lagrange's interpolation of the sum-of-shares for the same subset of shares.

►Proof of lemma and solution◄

2.6 Fiat-Shamir Heuristic

To transform any zero-knowledge protocol into a signature scheme, one can use the approach described in [12]. Here, we outline the general concept.

Zero-knowledge protocols typically rely on the verifier to issue a challenge to the prover. This challenge serves as a source of randomness that the prover is not supposed to control. In the Fiat-Shamir framework, the original protocol is based on the problem of factoring integers – a problem that is now vulnerable to quantum attacks, specifically Shor's algorithm. However, the fundamental method of converting a zero-knowledge protocol into a signature scheme remains unaffected.

To adapt a zero-knowledge protocol into a signature scheme, the prover eliminates the need for a verifier to provide randomness. Instead, the prover generates the randomness using a pseudo-random function. This function takes as input the message to be signed along with some random values generated by the prover. The output of this function serves as the challenge. With this, the prover can compute the required values to prove authenticity without external interaction.

In the context of the SD-in-the-Head protocol, the prover uses the pseudo-randomly generated challenge to locally compute the witness values required for the proof. These witness values are derived through the MPC protocol, where the prover simulates the necessary computations internally, generating and processing the shares "in their head." This eliminates the need for the verifier's active participation in the MPC protocol, streamlining the signature generation process.

Chapter 3

Specification

more detailed description of the algorithm e.g. how we sampled I[e] witness challenge table of spec params (with our code naming and different categories)

3.1 Field implementation

3.1.1 Field extension

First extend F_q to $F_{q^4} = F_q[Z]/(Z^2 + Z + 32(X))$. This is done by representing elements in the field as polynomials of degree at most 1 with coefficients in F_q.
▶Define addition and multiplication◀

▶See gf256_ext.rs◀

3.2 Merkle Tree Commitment

▶Can we use Haraka v2◀

3.3 Hashing and XOF

▶Write about Shake, Keccak◀ and how we initiate the hash function and XOF

3.4 MPC computation

The computation is based on HVZKAoK Protocol using imperfect preprocessing and sacrificing. Section 3.3 [4].

A toy example of the computation protocol computations, can be seen in Figure 3.1. We have the two computation methods. Here for 1 split and 1 evaluation point. This means that we have only value for each challenge and beaver triple. Note that any arithmetic is run in GF256, so addition and subtraction are both `XOR` and multiplication is modulus $x^8 + x^4 + x^3 + x + 1$. Furthermore, for negation we have that $-a = a$.

Note that the

```
PartyComputation

Input:
$(s_a, Q', P, a, b, c), (\overline{\alpha}, \overline{\beta}), (H', y)$
$(\epsilon, r)$, with_offset
Output:
$(\alpha, \beta, v)$

$Q = Q'_1$ if with_offset else $Q'_0$
$S = (s_a | y + H's_a)$ if with_offset else $(s_a | H's_a)$
$v = -c$
$\alpha = \epsilon \cdot Q(r) + a$
$\beta = S(r) + b$
$v \mathrel{+}= \epsilon \cdot F(r) \cdot P(r)$
$v \mathrel{+}= \overline{\alpha} \cdot b + \overline{\beta} \cdot a$
$v \mathrel{+}= -\alpha \cdot \beta$    if with_offset
```

```
InverseComputation

Input:
$(s_a, Q', P), (\alpha, \beta, v), (\overline{\alpha}, \overline{\beta}), (H', y)$
$(\epsilon, r)$, with_offset
Output:
$(a, b, c)$

$Q = Q_1$ if with_offset else $Q_0$
$S = (s_a | y + H's_a)$ if with_offset else $(s_a | H's_a)$
$c = -v$
$a = \alpha - \epsilon \cdot Q(r)$
$b = \beta - S(r)$
$c \mathrel{+}= \epsilon \cdot F(r) \cdot P(r)$
$c \mathrel{+}= \overline{\alpha} \cdot b + \overline{\beta} \cdot a$
$c \mathrel{+}= -\alpha \cdot \beta$    if with_offset
```

Figure 3.1: Simplified version of the MPC party computation and inverse computation. Q_0 means that Q is completed with a 0 for leading coefficient. Furthermore, F is precomputed. Note that all arithmetic is done in $\mathbb{F}_q = GF256$. All elements are in $\mathbb{F}_q^\eta$ except for the coefficients of Q, S and P which are in $\mathbb{F}_q$.

If we first instantiate an input i and one random input i^* (like the `input_coef`).Then the input share is generated by adding the two. Similar, but simpler, to the input share generation of Algorithm 12, line 13 of the specification.

$$\begin{aligned}
i &= (s_a, Q, P, a, b, c) \\
i^* &= ({s_a}^*, Q^*, P^*, a^*, b^*, c^*) \\
[i] = i + i^* &= (s_a + {s_a}^*, Q + Q^*, P + P^*, a + a^*, b + b^*, c + c^*) \\
&= ([s_a], [Q], [P], [a], [b], [c]) \\
\texttt{chal} &= (\epsilon, r) \\
\texttt{pk} &= (H', y)
\end{aligned}$$

We also compute the plain broadcast share of the input as per Algorithm 12, line 18. Note that $\overline{v}$ is computed to zero and therefore removed from the computation in the implementation.

$$(\overline{\alpha},\ \overline{\beta}) = \texttt{PartyComputation}(i,\ (\overline{\alpha}, \overline{\beta}),\ \texttt{chal},\ \texttt{pk},\ \texttt{true})$$

$$\begin{aligned}
\overline{\alpha} =& \epsilon \cdot Q_1(r) + a \\
\overline{\beta} =& S_y(r) + b \\
\overline{v} =& - c + \epsilon \cdot F(r) \cdot P(r) + \overline{\alpha} \cdot b + \overline{\beta} \cdot a - \overline{\alpha} \cdot \overline{\beta} \\
\overline{v} =& - c + \epsilon \cdot F(r) \cdot P(r) + (\epsilon \cdot Q_1(r) + a) \cdot b + (S_y(r) + b) \cdot a - (\epsilon \cdot Q_1(r) + a) \cdot (S_y(r) + b) \\
\overline{v} =& - c + \epsilon \cdot F(r) \cdot P(r) \\
&\quad + \epsilon \cdot Q_1(r) \cdot b + c + S_y(r) \cdot a + c \\
&\quad - \epsilon \cdot Q_1(r) \cdot S_y(r) - \epsilon \cdot Q_1(r) \cdot b - a \cdot S_y(r) - c \\
\overline{v} =& 0
\end{aligned}$$

We then compute a broadcast share from the randomness and the broadcast, as per Algorithm 12, line 21.

$$(\alpha^*, \beta^*, v^*) = \texttt{PartyComputation}(i^*, (\overline{\alpha}, \overline{\beta}), \texttt{chal}, \texttt{pk}, \texttt{false})$$

$$\begin{aligned}
\alpha^* &= \epsilon \cdot {Q^*}_0(r) + a^* \\
\beta^* &= {S^*}_0(r) + b^*
\end{aligned}$$

This broadcast share is sent to the verifier along with the truncated input share (removing the beaver triples). The verifier then needs to recompute the input share beaver triples using the `InverseComputation` function. First we add the input share to the broadcast share as per Algorithm 13, line 8.

$$(\alpha', \beta', v') = (\alpha^*, \beta^*, v^*) + (\overline{\alpha},\ \overline{\beta}, 0) = (\alpha^* + \overline{\alpha}, \beta^* + \overline{\beta}, v^* + 0)$$

$$\begin{aligned}
\alpha' &= \epsilon \cdot {Q^*}_0(r) + a^* + \overline{\alpha} \\
&= \epsilon \cdot {Q^*}_0(r) + a^* + \epsilon \cdot Q_1(r) + a \\
\beta' &= {S^*}_0(r) + b^* + \overline{\beta} \\
&= {S^*}_0(r) + b^* + S_y(r) + b
\end{aligned}$$

Next, the verifier computes the inverse of the broadcast share to recompute $([a], [b], [c])$ using the `InverseComputation` function. This is done as per Algorithm 13, line 10.

$$(a', b', c') = \texttt{InverseComputation}([i], (\alpha', \beta', v'), (\overline{\alpha}, \overline{\beta}), \texttt{chal}, \texttt{pk}, \texttt{true})$$

$$\begin{aligned}
a' &= \alpha' - \epsilon \cdot [Q]_1(r) \\
&= \epsilon \cdot {Q^*}_0(r) + a^* + \epsilon \cdot Q_1(r) + a - \epsilon \cdot [Q]_1(r) \\
&= \epsilon \cdot {Q^*}_0(r) - \epsilon \cdot [Q]_1(r) + \epsilon \cdot Q_1(r) + [a] \\
&= \epsilon \cdot ({Q^*}_0(r) - [Q]_1(r) + Q_1(r)) + [a] \\
&= \epsilon \cdot (Q_0^*(r) + Q_0^*(r)) + [a] && \textit{Equation } 2.1 \\
&= [a] && \textit{Equation } 2.1 \\
b' &= \beta' - [S]_y(r) \\
&= {S^*}_0(r) + b^* + S_y(r) + b - [S]_y(r) \\
&= {S^*}_0(r) - [S]_y(r) + S_y(r) + [b] \\
&= {S^*}_0(r) - S_0^*(r) + [b] && \textit{Equation } 2.1 \\
&= [b] && \textit{Equation } 2.1
\end{aligned}$$

▶Want to explain the above in a more detailed manner? Specifically ${Q^*}_0(r) - [Q]_1(r) = Q_1(r)$◀

3.5 Security

The security analysis of the SD-in-the-Head signature scheme is based on the proposed standardization requirements from the 2022 NIST call for proposals of non-lattice based signature schemes [20]. Therefore, as a preliminary, we will describe the reasoning and requirements of the NIST standardization process.

With the development of new quantum algorithms and unknowns in the capacities of the future quantum computers, there remain large uncertainties in estimating the security of the algorithms. To combat these uncertainties, NIST proposed for the 2022 stadardization effort, to define the security of submissions in a range of five categories. Each, with an easy-to-analyze cryptographic primitive providing the lower bound for a variety of metrics deemed relevant to practical security. The SD-in-the-Head specification provides security parameters adhering to categories one, three and five.

Definition 3.5.1. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit (e.g. AES-128), 192-bit (e.g. AES-192) and 256-bit (e.g. AES-256) for categories one, three and five respectively.

In terms of quantum security, the complexity and capability of quantum algorithms are measured in terms of quantum circuit size, i.e. the number of quantum gates in the quantum circuit. In order to estimate the quantum security of the signature protocols, circuit size can be compared to the resources required to break the security of Definition 3.5.1. Therefore, according to the proposal by NIST, the SD-in-the-Head specification provides security metrics in terms of quantum circuit depth to optimal key recovery for AES-128, AES-192 and AES-256 for categories one, three and five respectively. These are estimated to be 2^{143}, 2^{207} and 2^{272} classical gates [20].

3.5.1 Security Definition

The SD-in-the-Head signature scheme upholds the **E**xistential **U**n**f**orgeability under **C**hosen **M**essage **A**ttack (EUF-CMA) security property for digital signature schemes as this is the type of attack that NIST will evaluate signature proposals [20, 2]. EUF-CMA works as the following game:

1. The challenger generates a key pair (pk, sk) and sends pk to the adversary.
2. The adversary is then allowed to query a signing oracle for signatures of chosen messages $(m_1, ..., m_r)$ and receives valid signatures $(\sigma_1, ..., \sigma_r)$. For the NIST evaluation it is assumed that the adversary can query the signing oracle for up to 2^{64} chosen messages according to the adversary's running time. However, there is no requirement on the timing of the queries.
3. The adversary then outputs a pair (m^*, σ^*). The adversary wins if the following holds
   (a) m^* has not been queried to the signing oracle.
   (b) The pair (m^*, σ^*) is a valid signature for m^* under pk.

3.5.2 Assumptions

The SD-in-the-Head protocol is secure under the following assumptions:

Syndrome Decoding instances cannot be solved in complexity lower than 2^κ corresponding to the complexity of breaking AES by exhaustive search (see Definition 3.5.1) in terms of quantum circuit size. For this, κ is defined as 143, 207 and 272 for the categories [20]. Furthermore, the XOF primitive used is secure with 128-bit, 192-bit, 256-bit security levels for each of the categories respectively.

NIST security		**SD parameters**					**MPCitH Parameters**				
Category	Bits	q	m	k	w	d	N	ℓ	τ	η	t
I	143	256	242	126	87	1	q	3	17	4	7
III	207	256	376	220	114	2	q	3	26	4	10
V	272	256	494	282	156	2	q	3	34	4	13

Table 3.1: Security parameters for the SDitH protocol for categories one, three and five [2].

	I	**III**	**V**
Hash	SHA3-256	SHA3-384	SHA3-512
XOF	SHAKE-128	SHAKE-256	SHAKE-256

Table 3.2: Hash and XOF functions used in the SDitH protocol for categories one, three and five [2].

Finally, the Hash function used *behaves as random oracle*. Specifically, security holds in Random Oracle Model (ROM) and Quantum Random Oracle Model (QROM).

3.5.3 Security of the Syndrome Detection Problem

Recall the definition of the syndrome detection problem from Definition 2.3.1. The hardness of the SD problem is well established and the coset variant used in the SD-in-the-Head signature scheme, has been shown to be *NP-complete* [6, 2]. Furthermore, a brute force attack of guessing x would require finding a unique correct solution in $\binom{m}{w}q^w$ which is infeasible for the parameters outlined in the specification. As an example, for category one, the number of possible solutions are $\approx 7.7 \times 10^{276}$. There exists more sophisticated algorithms for solving the SD problem, such as the Generalized Birthday Algorithms (GBA) and Information Set Decoding (ISD) [21].

Chapter 4

Implementation

We set out to manually develop as many of the subroutines and primitives ourselves, including GF256 and merkle trees. However, hashing and XOF where made using pre-existing Rust libraries.

Tooling and language feaures (rust, criterion, rayon) code sections code re-usability with traits for categories.

const generics vs inline mutability (benchmarking?, nightly?) compiling constants for categories

can we use other hashes that still provide the same security assumption from Section 3.5.2 as post-quantum security (Xoodyak, KangarooTwelve, Haraka v2)

4.1 Rust

[19]

4.2 Subroutines

Here we implemented hashing as the first part, along with commitment. (cd1acdb0b6e9)

4.2.1 Addition of parameters

Then we introduced the parameters from the spec. f4739166d684

4.2.2 Hashing, XOF, and Commitments

4.2.3 Galois field arithmetic

Implemented the galois field. 94c3d9889709

4.3 Key generation

4.4 MPC

We started on implementing the mpc computation algo along with the inverse

4.5 Comparison with spec impl

In order to verify and debug our own code we tried to setup our implementation to fit with the specification implementation in C, meaning we had to run their code in order to get the intermediate data in order to compare the internal states. Here we found a bug in the way the view-opening challenges was calculated, they had forgotten to finalize the Shake before squeezing, which was a problem because in the tiny keccak rust library this was always done when initializing a hash or xof. Furthermore we found that in order to generate the same output from the xof we needed to first rotate the permutation once by supplying an empty vector before actually using the function. We also found that the merkle tree was not using the salt.

4.6 Benchmarking

We setup Criterion in order to do initial benchmarking

4.7 Categories

We tried reworking to use dynamically sized vectors instead of compile time sized arrays. Big rework and refactor. Benching showed slower running times. Instead we went with compile time script to set constants.

4.8 Optimizations

4.8.1 Feature flags

4.8.2 Parallelisation

4.8.3 SIMD

4.8.4 Hash functions

Blake3, Haraka v2, KangarooTwelve, Xoodyak

4.9 Kat for NIST

Chapter 5

Benchmarks

diaries of benchmarks. discussion of results

Test on both mac and linux. nightly vs stable rust different hashes benching at different tags parallelisation, test for amount of cores, 2, 4, 8, 16 no turbo boost (max 2.6 GHz) cycles per bytes compare ours to the optimised reference

Chapter 6

Conclusion

wrap up and pose future work what should people continue with point to round 2 NIST work in context of timeline

▶conclude on the problem statement from the introduction◀

future work: Verkle trees optimisation

Bibliography

[1] The post-quantum security of the SHA-3 hash function standard — orbit.dtu.dk. `https://orbit.dtu.dk/en/projects/the-post-quantum-security-of-the-sha-3-hash-function-standard`. [Accessed 21-11-2024].

[2] C Aguilar-Melchor et al. The syndrome decoding in the head (sd-in-the-head) signature scheme. submission to the nist call for additional post-quantum signatures (2023).

[3] Marco Baldi, Marco Bianchi, and Franco Chiaraluce. Optimization of the parity-check matrix density in qc-ldpc code-based mceliece cryptosystems. In *2013 IEEE International Conference on Communications Workshops (ICC)*, pages 707–711. IEEE, 2013.

[4] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *IACR International Conference on Public-Key Cryptography*, pages 495–526. Springer, 2020.

[5] Georg Becker. Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep*, 12:19, 2008.

[6] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[7] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The sphincs+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2129–2146, 2019.

[8] Karen H Brown. Advanced encryption standard (aes).

[9] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[10] Léo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals–dilithium: Digital signatures from module lattices. 2018.

[11] Thibauld Feneuil and Matthieu Rivain. Threshold linear secret sharing to the rescue of mpc-in-the-head. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 441–473. Springer, 2023.

[12] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[13] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, et al. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 36(5):1–75, 2018.

[14] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

[15] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 525–537, 2018.

[16] Consuelo Martínez and Fabián Molina. The syndromes decoding algorithm in group codes. *Finite Fields and Their Applications*, 89:102206, 2023.

[17] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[18] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information.* Cambridge university press, 2010.

[19] NIST. Safer languages. `https://www.nist.gov/itl/ssd/software-quality-group/safer-languages`, 2017.

[20] NIST. Call for additional digital signature schemes for the post-quantum cryptography standardization process. `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf`, 2022.

[21] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[22] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[23] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[24] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

Appendix A

The Technical Details

►. . .◄