IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 2, pp. 1–22. DOI:10.46586/tches.v2025.i2.1-22

Improving MPCitH with Preprocessing: Mask Is All You Need

Guowei Liu[1,7], Guoxiao Liu[2], Kaijie Jiang[3], Qingyuan Yu[1,7], Keting Jia[2,5,6](✉), Puwen Wei[1,7,4] and Meiqin Wang[4,1,7](✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China
{guoweiliu,yuqy}@mail.sdu.edu.cn,pwei@sdu.edu.cn
[2] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
lgx22@mails.tsinghua.edu.cn,ktjia@tsinghua.edu.cn
[3] Institute for Advanced Study, Tsinghua University, Beijing, China
jkj21@mails.tsinghua.edu.cn
[4] Quan Cheng Shandong Laboratory, Jinan, China
mqwang@sdu.edu.cn
[5] Zhongguancun Laboratory, Beijing, China
[6] BNRist, Tsinghua University, Beijing, China
[7] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China

Abstract. The MPC-in-the-head with preprocessing (MPCitH-PP) paradigm presents a novel approach for constructing post-quantum digital signatures like Picnic3. This paper revisits the MPCitH-PP construction, analyzing both its offline and online phases and proposing a reformulation of the protocol. By identifying redundant computations in these phases, we optimize them into a single phase, thereby enhancing the efficiency of MPCitH-PP. Furthermore, we explore the independence of the mask, demonstrating that it can be calculated in parallel, which also enables the optimization of the masked witness calculation.
Our optimized implementation of Picnic3 shows significant improvements. At the L1 security level, the optimal software implementation reduces MPCitH-PP calculation time to about 30% of the previous implementation. The optimal signature implementation costs about 78% of the previous implementation time. At the L5 security level, MPCitH-PP with parallelism optimal is reduced to about 26% of the previous solution's time, and the optimal signature implementation runs at about 53% of the previous solution's time. For the hardware implementation, our optimizations reduce the clock cycles of MPCitH-PP from r sequential rounds to a single parallel round, where r denotes the number of rounds in the LowMC algorithm, with little change in hardware usage, and perform better in AT product, especially for parallel computing.

Keywords: MPCitH with preprocessing · Post-Quantum Digital Signature · Software Implementation · Hardware Implementation

1 Introduction

In 2016, NIST launched the post-quantum (PQ) standardization process, attracting significant interest from researchers. Picnic [ZCD+20] is a third-round alternate candidate among post-quantum signature submissions. Unlike traditional PQ signatures relying on mathematical assumptions like lattice-based signatures, multivariate signatures, or isogeny signatures, Picnic only relies on the security of symmetric primitives, which could provide more conservative security.

Received: 2024-10-15 Accepted: 2024-12-15 Published: 2025-03-04

The core technique used by Picnic is known as Multi-Party Computation in the Head (MPCitH), which was first proposed by Ishai et al. [IKOS07] for zero-knowledge (ZK) protocols, and was further improved by [GMO16] to a fast ZK protocol named ZKBoo.

The basic idea of MPCitH is built upon the N-party MPC protocol that can jointly compute a function $f_{\mathbf{x}}(\mathbf{w})$. By revealing all-but-one parties' transcripts, the prover can convince the verifier $f_{\mathbf{x}}(\mathbf{w}) = 1$ in a zero-knowledge manner. Chase et al. proposed an improved version of ZKBoo, named ZKB++, and the Picnic signature scheme based on their paradigm [CDG+17]. Katz et al. [KKW18] showed that a particular communication-efficient MPC protocol in the preprocessing model is well suited to MPCitH proofs. They introduced MPCitH with preprocessing (MPCitH-PP) to reduce proof sizes, leading to more compact signature Picnic2. The main idea of the MPCitH-PP model is to split the proof protocol into offline phase and online phase, enabling independent computation in the offline phase. By precomputing correlated randomness in offline phase, the communication cost in online phase can be reduced drastically. This idea has also been used in traditional MPC settings, such as [DPSZ12, DZ13], and influenced subsequent works, including [BN20, dSGMOS19, Beu20], as well as the improved version of Picnic, Picnic3, introduced at TCHES 2020 [KZ20].

However, Picnic has much higher costs when compared to lattice-based signatures like Dilithium [LLDL+20] and FALCON [PFH+20]. In contrast to SPHINCS+ [HBD+20], which relied on standard hash functions, the signing speed of Picnic is much faster, but the security of Picnic's underlying encryption scheme, LowMC, has not been as extensively studied as that of standard symmetric-key primitives. Despite not being chosen as a finalist, Picnic demonstrates advantages in both software and hardware performance. The MPCitH technique used in Picnic has inspired recent PQ signature submissions. In NIST's recent call for additional digital signature proposals, there are about 9 out of 40 submissions that are related to MPCitH [BKPV24, ABB+23, ABB+24, BFR23, BBD+24, BCF+23, CCJ23, BBdSG+23, KHS+23]. In particular, new "in the head" techniques such as VOLEitH [BBdSG+23] and TCitH [FR] demonstrate enhanced performance over original MPCitH in computation and communication.

So far, several optimized implementations of MPCitH-PP have been developed. Kales and Zaverucha optimized the calculation of the linear operations of MPCitH-PP, so that the linear operations of the original N parties only need to be calculated once [KZ20]. Liu et al. implemented MPCitH-PP in hardware, so that the LowMC-MPC calculation of N parties (even 16 parties) can also be arranged on a resource-constrained FPGA [LJWJ24]. Although many digital signatures based on MPCitH-PP have been proposed, there is a requirement to optimize both the signature footprint and the implementation efficiency for practical application. The primary focus of this paper is to identify and measure redundant calculations between MPCitH-PP for digital signatures, and to propose optimizations for MPCitH-PP.

Contributions. In this paper, we re-describe MPCitH-PP within the KKW protocol with three phases instead of two phases and identify redundant computations in the last two phases of MPCitH-PP. We observe that the inclusion of a random mask introduces a degree of independence between the round functions of the adopted symmetric primitive. Based on these observations, we propose optimization techniques to improve efficiency and perform experimental verification as follows:

1. **Reformulation of MPCitH-PP Protocol.** We highlight the gap between the MPCitH-PP protocol and MPCitH-PP-based digital signatures. In the MPCitH-PP protocol, there is no witness required in the offline phase (preprocessing), and the online phase uses a masked secret for computation. However, in digital signatures, the random tape sampled during the offline phase of MPCitH-PP is generated by taking the public key of the digital signature (statement x in MPCitH-PP), the

private key (witness w in MPCitH-PP), the message, etc. as input. Therefore, the offline phase of MPCitH-PP in digital signatures implicitly includes the witness, making it impossible to calculate the random tape "in advance" for the offline phase. Therefore, we reorganized the MPCitH-PP protocol into three phases: sample phase, the aux phase, and the msgs phase. We divided the offline phase into the sample phase, which includes only the random tape, and the aux phase, while the msgs phase remains the same as the original online phase.

2. **Efficient Mask Calculation Strategy: Mask Is All You Need.** Our core idea is to make full use of the calculation of the masks. In MPCitH-PP, the aux phase calculates the evaluation of the underlying circuit C, and the msgs phase recalculates the evaluation of the underlying circuit C. Because the prover has a witness, they can leverage this and calculate all the intermediate states of the aux phase with the intermediate states of the normal calculation of the underlying circuit C, thereby directly obtaining the intermediate states of the msgs phase. This is why we say that mask is all you need.

3. **Optimization of Mask Independence.** The mask in the sample phase is directly sampled from random tapes, so we explored the independence of the mask. For each mask, there is no need for data dependency like the calculation of the underlying circuit. The mask can be calculated directly from random tapes, so the calculation has no dependency and parallel calculation becomes possible. The calculation of the mask can be directly optimized, but the calculation of the masked witness is not only related to the mask, so we use the optimization in Section 4 to make the masked witness independent.

4. **Performance Improvements in Software and Hardware Implementations.** We present the techniques implemented in software and hardware, and compare the results with the protocol before optimization. Our new software-optimized implementation achieves significant improvements: at the security level L1, it runs in approximately 74% of the time of the previous solution implemented in [KZ20], and with parallelism implementation of MPCitH-PP, it costs only about 30% of that time. For the signature scheme, it costs about 88% of the previous solution's time, and with parallelism implementation, it costs about 78%. At the security level L5, the running time for MPCitH-PP is approximately 62% of that of the previous solution, and about 26% with parallelism. For the signature scheme, the times are about 73% and 53% with parallelism, respectively. At the hardware level, our optimizations reduce the clock cycles from r sequential rounds to a single parallel round, with little change in hardware usage. We provide an area-time(AT) product, and our hardware implementation AT performs better, especially for parallel computing.

2 Priliminaries

Notation. Let L denote an NP language. The NP relation is defined as $R(\mathbf{x}, \mathbf{w}) = 1$ if the statement $x \in L$ and $\mathbf{w}$ is the corresponding witness. Let $[x]$ denote an N-out-of-N (XOR-based) secret sharing scheme of a bit x, i.e., $x = [x]_1 \oplus \cdots \oplus [x]_i \oplus \cdots \oplus [x]_N$, where $[x]_i$ for $1 \leq i \leq N$ is the secret share. Let $[i, j]$ denote the range from integers i to j.

2.1 MPC-in-the-head with Preprocessing

MPC-in-the-head proposed by Ishai *et al.* [IKOS07] provides a novel method to construct zero-knowledge proof (ZKP) for any NP language L. In this paper, we consider the relation $R(\mathbf{x}, \mathbf{w})$ as $f_{\mathbf{x}}(\mathbf{w}) = 1$ for a function f. An MPCitH proof system (P, V) is built upon an

N-party MPC protocol that jointly computes the function f. Here, f takes $\mathbf{x}$ and $\mathbf{w}$ as the public and private inputs, respectively, and computes $f_{\mathbf{x}}(\mathbf{w}) = R(\mathbf{x}, \mathbf{w})$.

At a high level, the MPCitH prover P aims to convince the verifier V that they possess a valid witness $\mathbf{w}$ by demonstrating that the MPC protocol has been correctly executed "in the head" of P using input $\mathbf{w}$.

We now consider an MPC protocol Π_C for the corresponding circuit C defined over the field $\mathbb{F}_2$, where the statement information $\mathbf{x}$ (e.g., the plaintext-ciphertext pair) is hard-coded such that $C(\cdot) = f_{\mathbf{x}}(\cdot)$. We assume that the witness can be represented as an n-dimensional vector and C takes a set of n input wires denoted by IN. Let z_α denote the value of wire α of $C(w)$, then $\mathbf{w} = (z_\alpha)_{\alpha \in \mathrm{IN}} \in \mathbb{F}_2^n$ be the input of C. To initiate the protocol, the prover P first additively secret shares each input z_α as $z_\alpha = [z_\alpha]_1 \oplus \cdots \oplus [z_\alpha]_N$ in $\mathbb{F}_2$. Each share $[z_\alpha]_i$ is considered as a private input to party P_i. Then, prover P internally runs Π_C for parties $\mathrm{P}_1, \cdots, \mathrm{P}_N$ to obtain the views $\mathrm{V}_1, \cdots, \mathrm{V}_N$, where view V_i consists of P_i's private input $[z_\alpha]_i$, the random tape of P_i, and all incoming messages observed by P_i during the execution of Π_C. The proof system now follows the typical "commit-challenge-response" flow (Σ-protocol [FS86]). Using a secure commitment scheme, P sends $\mathrm{Commit}(\mathrm{V}_i)$ as the first message for all $i \in [1, N]$. Upon receiving distinct challenges $i_1, \cdots, i_t \in [1, N]$ from the verifier V, the prover P responds with the corresponding t views $\mathrm{V}_{i_1}, \cdots, \mathrm{V}_{i_t}$ and the commitment opening information. Finally, the verifier V accepts the proof if and only if the opened views are consistent with each other and they result in an output of 1 from the protocol Π_C. The (honest verifier) zero-knowledge property is guaranteed if the underlying MPC Π_C achieves t-privacy in the semi-honest model.

MPCitH with preprocessing (MPCitH-PP). Katz *et al.* [KKW18] improved the MPCitH paradigm by using the preprocessing mode. Further improvements can be found in subsequent works [ZWX+22]. Loosely speaking, Katz *et al.*'s protocol (KKW) has two phases, which are the *offline* phase (preprocessing phase) and the *online* phase. We denote Π_C^{off} and Π_C^{on} as the offline phase protocol and the online phase protocol, respectively. The offline phase protocol Π_C^{off}, which is executed independently of the witness, prepares the randomness for the online phase protocol Π_C^{on}. Considering the application in Picnic3, the following descriptions of the MPC protocol and KKW protocol are based on boolean circuits.

Suppose the underlying N-party MPC protocol is Π_C, which is executed by N parties $P_1, \cdots, P_N$. The value of each input wire z_α of each AND gate will be masked by a random bit λ_α, say, $\hat{z}_\alpha = z_\alpha \oplus \lambda_\alpha$. Each party P_i holds a share of λ_α, denoted by $[\lambda_\alpha]_i$. We use the notations clearly, as in [KKW18]. The details are in Figure 1.

KKW Protocol. We briefly recall the basic framework of KKW for one MPC instance, which is a three-round MPCitH-PP system.

We adopt the complete description of the KKW proof system as proposed in [KKW18], which utilizes multiple instances in parallel to achieve a negligible soundness error. The parameter M describes the number of repetitions of MPCitH-PP required to reduce the soundness error to the desired security level. The parameter τ is the opened execution in MPCitH with preprocessing, and N is the number of parties.

- **Commit.** The prover P begins by sampling a random seed for each P_i and executes protocol Π_C^{off} to obtain the states of all N parties. Then, using these states and the masked witness $(\hat{z}_\alpha)_{\alpha \in \mathrm{IN}}$ as input, P executes protocol Π_C^{on} to obtain all broadcast messages observed during the online phase. P computes commitments to the states and broadcast messages. Finally, P sends commitments to the verifier V.

- **Challenge.** V asks P to disclose either the offline or the online phase. In the case of the latter, V also randomly selects a party index $p^\star$, whose view should remain

- **Offline phase** $\Pi_C^{\mathbf{off}}$. In the offline phase, the prover generates the masks for each party P_i. More precisely, P_i is given the following values.
  - $[\lambda_\alpha]_i$ for each input wire α.
  - $[\lambda_\gamma]_i$ for the output wire γ of each AND gate.
  - $[\lambda_{\alpha,\beta}]_i$ for each AND gate with input wires α and β such that $\lambda_{\alpha,\beta} = \lambda_\alpha \cdot \lambda_\beta$.

  For $i = 1, \cdots, N-1$, $[\lambda_\alpha]_i$, $[\lambda_\gamma]_i$ and $[\lambda_{\alpha,\beta}]_i$ are generated using a pseudorandom generator (PRG) with a random seed $\mathbf{seed}_i$. Besides, $[\lambda_\alpha]_N$, $[\lambda_\gamma]_N$ are generated by PRG with a random seed $\mathbf{seed}_N$. Here $[\lambda_\alpha]_1 \oplus \cdots \oplus [\lambda_\alpha]_N = \lambda_\alpha$, $[\lambda_\gamma]_1 \oplus \cdots \oplus [\lambda_\gamma]_N = \lambda_\gamma$. Notice that $[\lambda_{\alpha,\beta}]_N$ cannot be generated using $\mathbf{seed}_N$ due to $\lambda_{\alpha,\beta} = \lambda_\alpha \cdot \lambda_\beta$. Actually, $[\lambda_{\alpha,\beta}]_N := \lambda_\alpha\lambda_\beta \oplus [\lambda_{\alpha,\beta}]_1 \oplus \cdots \oplus [\lambda_{\alpha,\beta}]_{N-1}$, which plays the role of "correction bits". In order to reduce the total proof size, it is possible that $\mathbf{seed}_i$ is given to P_i, and $\mathbf{seed}_N$ and $\mathbf{aux}_N = [\lambda_{\alpha,\beta}]_N$ are given to P_N.
- **Online phase** $\Pi_C^{\mathbf{on}}$. During the online phase, each party P_i evaluates the circuit C gate-by-gate in topological order. For each gate with input wires α and β and output wire γ,
  - For an XOR gate, P_i can locally compute $\hat{z}_\gamma = \hat{z}_\alpha \oplus \hat{z}_\beta$ and $[\lambda_\gamma]_i = [\lambda_\alpha]_i \oplus [\lambda_\beta]_i$, since P_i already holds $\hat{z}_\alpha, [\lambda_\alpha]_i, \hat{z}_\beta$ and $[\lambda_\beta]_i$.
  - For an AND gate, P_i locally computes $[s]_i = \hat{z}_\alpha [\lambda_\beta]_i \oplus \hat{z}_\beta [\lambda_\alpha]_i \oplus [\lambda_{\alpha,\beta}]_i \oplus [\lambda_\gamma]_i$, publicly reconstructs $s = [s]_1 \oplus \cdots \oplus [s]_N$, and computes $\hat{z}_\gamma = s \oplus \hat{z}_\alpha\hat{z}_\beta$ which satisfies $\hat{z}_\gamma = z_\gamma \oplus \lambda_\gamma = z_\alpha z_\beta \oplus \lambda_\gamma$. Note that party P_i holds $[\lambda_{\alpha,\beta}]_i$ and $[\lambda_\gamma]_i$ in addition to $\hat{z}_\alpha, [\lambda_\alpha]_i, \hat{z}_\beta$ and $[\lambda_\beta]_i$ for each AND gate.

Figure 1: The online and offline phase of KKW protocol in [KKW18].

hidden.

- **Response.** To disclose the offline phase, P sends all random seeds used during protocol Π_C^{off}. To disclose the online phase, P sends the broadcast messages from party $P_{p^\star}$ during protocol Π_C^{on}, as well as all the state information of the remaining $N-1$ parties.
- **Verification.** To verify the offline phase, V simply uses the random seeds to execute protocol Π_C^{off} as P would, resulting in the states of all N parties. Then, V checks if these states correctly match the commitments of the offline phase. To verify the online phase, V simulates protocol Π_C^{on} with the broadcast messages from $P_{p^\star}$ and the states of the other $N-1$ parties as input, obtaining the broadcast messages from the other $N-1$ parties. Finally, V checks if these broadcast messages correctly match the commitments of the online phase.

2.2 Seed Generation

Hash functions are employed in Cto generate random values and commitments. In Picnic2, hash functions are employed to expand a random "seed" into additional random values using a tree structure, and to create a Merkle Tree of the committed values. Picnic3 uses the extendable-output functions SHAKE of the hash function SHA-3[BDPA11] for all hashing, with specific parameters detailed in Table 1. For more information on SHAKE, we refer the reader to [BDPA11].

When signing and verifying of signatures, Picnic3 generates a short random value (128 to 512 bits), called the *seed*, and expands it into a longer one (about 1KB), both are done with SHAKE. This choice allows a single function family (SHA-3) for both hashing and

Table 1: Parameters of KECCAK. Block length denotes the bit number absorbed or squeezed. Round denotes the number of repeat permutation KECCAK-p.

Scheme	Sec. Level	Block Length	Digest Length	Round
SHAKE128	L1	1344	256	24
SHAKE256	L5	1088	512	24

key derivation, as SHAKE with a fixed output length is also a secure hash function. At security level 1 we use SHAKE128 and security levels 3 and 5 use SHAKE256.

In Picnic3, the list `seeds[0,··· ,T-1][0,··· , N-1]` stores NT random seeds, each of length l_s bits, and the salt value *salt* is set to a 256-bit random value. It is recommended that these be derived deterministically, by calling the key derivation function (KDF) with input

$$sk||M||pk||l_s.$$

where l_s is encoded as a 16-bit little-endian integer. The number of bytes requested is $(NT)(l_s/8) + 32$, where $NT(l_s/8)$ for `seeds`, and 32 bytes for *salt*.

The test vectors associated with this document use this method to simplify testing. However, the specific method of generating `seeds` and *salt* does not affect interoperability, and implementations may differ (e.g., by choosing the values uniformly at random, using an alternative derivation method, or including alternative inputs to derivation).

2.3 LowMC

The security of Picnic also relies on a block cipher for one-way computing. The primitives are instantiated by LowMC [ARS+15] block cipher in Picnic3.

LowMC [ARS+15] is a family of lightweight SPN block ciphers proposed by Albrecht *et al.* at EUROCRYPT 2015. It is proposed for MPC- and FHE-friendly, the most important advantage is its low multiplicative complexity, *i.e.* small AND gate/depth. This property makes LowMC well-suited for a range of cryptographic applications, including multi-party computation (MPC), fully homomorphic encryption (FHE), and zero-knowledge proofs.

The encryption phase of LowMC starts with XORing with a whitening key K_0 and then iterates the round function by r times. The round function of the i-th round, $1 \le i \le r$, is composed of four steps, as described below:

- SboxLayer: A 3-bit S-box $S(a, b, c) = (a \oplus b \cdot c, a \oplus b \oplus a \cdot c, a \oplus b \oplus c \oplus a \cdot b)$ is applied to the first $3m$ bits, while the remaining bits not be modified.
- LinearLayer: A matrix $L_i \in \mathbb{F}_2^{n\times n}$ is randomly generated, the n-bit state is multiplied with L_i.
- ConstantAddition: A randomly generated n-bit constant $C_i \in \mathbb{F}_2^n$ is XORed to the n-bit state.
- KeyAddition: The n-bit is updated by XORed with a n-bit round key K_i. K_i is generated by multiplying the k-bit master key K with a randomly selected full-rank $n \times k$ binary matrix M_i. The whitening key K_0 is also calculated by $K_0 = M_0 \cdot K$.

Each round of LowMC can be described as LowMCRound(i) = KeyAddition ∘ ConstantAddition ∘ LinearLayer ∘ SboxLayer(i). The entire encryption phase is given in Algorithm 1. The parameters instantiated in Picnic3 [Pic20] are $(n, k, m, r) \in \{(129, 129, 43, 4), (192, 192, 64, 4), (255, 255, 85, 4)\}$.

Algorithm 1: LowMC encryption.

Input: plaintext $p \in \mathbb{F}_2^n$ and master key $K \in \mathbb{F}_2^k$.
Output: ciphertext $c \in \mathbb{F}_2^n$.

```
state ← p + M_0 · K
foreach i ∈ [1, r] do
    state ← SboxLayer(state)
    state ← L_i · state              ▷ LinearLayer
    state ← C_i ⊕ state              ▷ ConstantAddition
    state ← state ⊕ (M_i · K)        ▷ KeyAddition
end
c ← state
```

3 Reformulate MPCitH-PP

The MPCitH-PP protocol optimizes the proof size of MPCitH. There is a part of the calculation in the protocol that does not require a witness, so this part is called the offline phase. The online phase is the calculation that requires a witness. In MPCitH-PP, the offline phase is to calculate **aux**, and the online phase is to calculate the view **msgs** ($[s]$) of each participant. It can be observed that the calculation of **aux** is only related to the sampled random tape. In Subsection 2.2, we can see that the random tape of Picnic3 is generated by the public key of the digital signature (statement x in MPCitH-PP), the private key (witness $\mathbf{w}$ in MPCitH-PP), the message, etc. as input. Therefore, the offline phase of MPCitH-PP in the digital signature actually implicitly includes the witness, so the random tape cannot be calculated "in advance" for the offline phase. That is, the offline phase is not "offline", and both phases of MPCitH-PP are related to the witness, where the offline phase is implicitly represented by the random tape, while the online phase is represented by the masked witness $\hat{z}_\alpha$ and the random tape. In the previous description of the protocol [KKW18, KZ20, ZWX+22], the random tape is included as part of the offline phase, which involves the witness. Despite being labeled as "offline" this process does not strictly adhere to the traditional offline phase. Consequently, we reformulate the MPCitH-PP protocol based on the description in [LJWJ24]) to accommodate these nuances.

In order to adapt to the most famous MPCitH-PP-based protocol Picnic3, the reformulated MPCitH-PP protocol is optimized by [KZ20]. We adapt and modify the general circuit model presented by [LJWJ24]. To facilitate the explanation of these optimizations, the underlying circuit is abstracted into a structure where linear layers X, nonlinear layers A, and XOR state O alternate. O_j denotes the j-th key addition or some other state addition in the block cipher. X_j denotes the j-th linear layer in the block cipher. In LowMC, A_j is the AND gate of the j-th S-box layer, X_j is the j-th linear layer, and O_j is the equivalent representation of the XOR gate of the S-box after passing through the linear layer XORing with the key, i.e., the linear combination of input of S-box XORing with the key. For the sake of simplicity, the linear layer and the nonlinear layer in Figure 2 are assumed to consist of multiple XOR and AND gates, respectively, with the linear layer being invertible.

The reformulation of the MPCitH-PP protocol is divided into three phases: *Sample* phase, *Aux* phase, and *Msgs* phase in Figure 3, represented by as Π_C^{s}, Π_C^{a} and Π_C^{m}, respectively. The *Sample* phase protocol Π_C^{s} generates a random tape for each party P_i. The *Aux* phase protocol Π_C^{a} prepares the error correction value and output mask of each AND gate used in the *Msgs* phase protocol Π_C^{m}. The *Msgs* phase Π_C^{m} is used to generate the broadcast message for each AND gate. For each gate, denote the input wires as α and β and the output wire as γ for convenience. The reformulated *Sample*, *Aux*, and *Msgs*

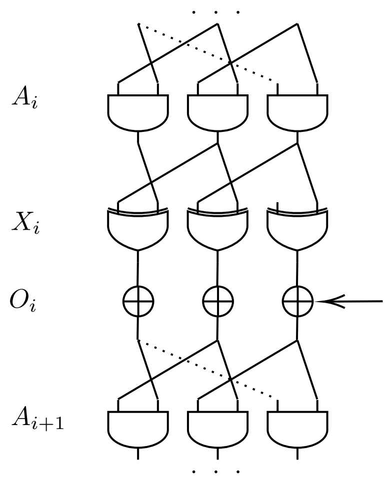

Figure 2: The circuit model in [LJWJ24].

phases of MPCitH-PP are shown in Figure 3.

- ***Sample*** **phase** $\Pi_C^{\mathbf{s}}$. The prover generates masks for all AND gates for all parties. Each party P_i has the following mask values.
  - $[\lambda_\alpha]_i$ (and $[\lambda_\beta]_i$) for the input wire α (and β) of each AND gate.
  - $[\lambda_{\alpha,\beta}]_{i \neq N}$ for each AND gate with input wires α and β.
- ***Aux*** **phase** $\Pi_C^{\mathbf{a}}$. The prover computes the error correction value **aux** for each AND gate round by round in the right of Figure 4.
  - For the linear layer X_j, the prover computes $(\lambda_\alpha^{j+1},\ \lambda_\beta^{j+1})$ by the input masks for A_{j+1} of all parts and computes λ_γ^j by the inverse of the linear operation X_j.
  - For each AND gate of A_j, the prover computes the error correction value $\left[\lambda_{\alpha,\beta}^j\right]_N := \lambda_\alpha^j \lambda_\beta^j \oplus \left[\lambda_{\alpha,\beta}^j\right]_1 \oplus \cdots \oplus \left[\lambda_{\alpha,\beta}^j\right]_{N-1} \oplus \lambda_\gamma^j$ as **aux** for the party P_N.
- ***Msgs*** **phase** $\Pi_C^{\mathbf{m}}$. Each party P_i evaluates the circuit C gate-by-gate in topological order.
  - For each AND gate of A_j, P_i locally computes $[s^j]_i = \hat{z}_\alpha^j \left[\lambda_\beta^j\right]_i \oplus \hat{z}_\beta^j \left[\lambda_\alpha\right]_i \oplus \left[\lambda_{\alpha,\beta}^j\right]_i$, publicly reconstructs $s^j = [s^j]_1 \oplus \cdots \oplus [s^j]_N$, and computes $\hat{z}_\gamma^j = s^j \oplus \hat{z}_\alpha^j \hat{z}_\beta^j$ which satisfies $\hat{z}_\gamma^j = z_\gamma^j \oplus \lambda_\gamma^j = z_\alpha^j z_\beta^j \oplus \lambda_\gamma^j$. Note that each party P_i holds $[\lambda_\alpha]_i, [\lambda_\beta]_i$ and $[\lambda_{\alpha,\beta}]_i$ for each AND gate.
  - For the linear operations X_j and O_j, each P_i can publicly compute linear operation with the output masked witness $\hat{z}_\gamma^j$ of A_j as input to get the masked witness input values $\hat{z}_\alpha^{j+1}$ and $\hat{z}_\beta^{j+1}$.

Figure 3: The reformulated *Sample*, *Aux*, and *Msgs* phase of KKW protocol.

We review the modification of the KKW protocol by [KZ20] to reduce the computational complexity of linear operations from $O(N)$ to $O(1)$ in both the offline and online phases. Let λ_α^j and λ_β^j be the mask for the input wire of j-th non-linear operation (A_j), λ_γ^j be the mask for the output wire of j-th non-linear operation (A_j), and λ_O^j be the mask for other state (O_j). In the calculation of **aux** in the KKW protocol, each party P_i samples the output mask of A_j to obtain the share $[\lambda_\gamma^j]_i$, uses the share to calculate X_j and O_j, and finally obtains the input masks $[\lambda_\alpha^{j+1}]_i, [\lambda_\beta^{j+1}]_i$ of A_{j+1}. In order to calculate $\mathbf{aux} = [\lambda_{\alpha,\beta}^j]_N = \lambda_\alpha^j \cdot \lambda_\beta^j \oplus \sum_{i \neq N} [\lambda_{\alpha,\beta}^j]_i$, each party broadcasts $[\lambda_\alpha^{j+1}]_i, [\lambda_\beta^{j+1}]_i$. Since λ_α^{j+1} and λ_β^{j+1} are masks instead of shares, λ_γ^j can be calculated first, and then the linear layer is calculated. Therefore, the linear layer only needs to be calculated once instead of once for each of the N parties. However, this optimization cannot be directly applied to the calculation of **msgs**, because the calculation for P_i of A_j's $[s^j]_i = \hat{z}_\alpha^j [\lambda_\beta^j]_i \oplus \hat{z}_\beta^j [\lambda_\alpha^j]_i \oplus [\lambda_{\alpha,\beta}^j]_i \oplus [\lambda_\gamma^j]_i$ requires the shares $[\lambda_\alpha^j]_i, [\lambda_\beta^j]_i$, so [KZ20] modifies the sampling position, which is no longer the output of the AND gate $[\lambda_\gamma]_i$, but the input of the AND gate $[\lambda_\alpha]_i, [\lambda_\beta]_i$. However, at this time, λ_γ^j is calculated by λ_α^{j+1}, λ_β^{j+1} and the λ_O^j (which is computed from the mask of key $M_0^{-1}(\lambda_\alpha^1, \lambda_\beta^1, \cdots)$ and the input of $L_j \cdot L_*(\lambda_\alpha^j, \lambda_\beta^j, \cdots)$ in LowMC, where L_* denotes the linear operation for the Sbox).

Therefore, for A_j, [KZ20] modifies $\mathbf{aux} = \lambda_\alpha^j \cdot \lambda_\beta^j \oplus \sum_{i \neq N} [\lambda_{\alpha,\beta}^j]_i \oplus \lambda_\gamma^j$, and $[s^j]_i = \hat{z}_\alpha^j [\lambda_\beta^j]_i \oplus \hat{z}_\beta^j [\lambda_\alpha^j]_i \oplus [\lambda_{\alpha,\beta}^j]_i$. Therefore, as shown in Figure 4, for a block cipher of r rounds, in MPCitH-PP, the circuit is no longer calculated from A_1 to O_r, but from O_r to A_1.

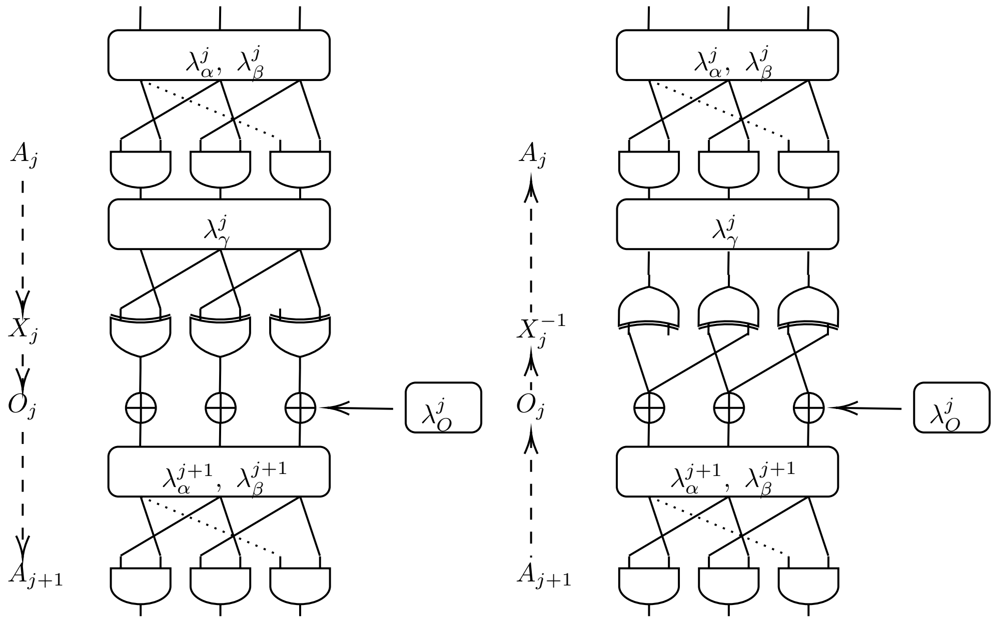

Figure 4: The circuit changes for the sampling of random masks (The left shows the Π_C^{a} circuit of [KKW18] and the right shows the Π_C^{a} circuit of [KZ20]).

4 Mask Is All You Need

In Section 3, the three phases of MPCitH-PP in KKW protocol shown in Figure 3 require (implicitly) witnesses to calculate, and both Π_C^{a} and Π_C^{m} need to calculate the underlying circuit C, and the calculation order of the two is different. C_A, C_X and C_O denote the A_i operation, the X_i operation and O_i operation of a round of the circuit C, respectively. The computational complexity of Π_C^{a} in an N-party r-round circuit is $O(r \cdot C_A + r \cdot C_X + r \cdot C_O)$, and the complexity of Π_C^{m} is $O(r \cdot N \cdot C_A + r \cdot C_X + r \cdot C_O)$. Then the complexity of calculating $\Pi_C^{\mathrm{a+m}}$ is $O(r \cdot (N+1) \cdot C_A + 2 \cdot r \cdot C_X + 2 \cdot r \cdot C_O)$. Π_C^{m} needs to calculate N-party C_A and once C_X, C_O, because $[s]_i$ needs to be calculated separately for each participant and compute linear operation X_i and O_i to get the masked witness input values $\hat{z}_\alpha^{j+1}$(

$\hat{z}_\beta^{j+1}$). while Π_C^{a} only needs to calculate one $[\lambda_{\alpha,\beta}]_N$ with λ_γ obtained by the computation of the inverse of the linear operation X_i and O_i.

According to the above discussion and the reformalized MPCitH-PP protocol given in Figure 3, the computation of C_A for Π_C^{a} and Π_C^{m} is completely different, but the computation of C_X for both is similar because $z_\alpha = \hat{z}_\alpha \oplus \lambda_\alpha$. Therefore, the computation of C_X for Π_C^{a} and Π_C^{m} differs only in the witness $z_\alpha, z_\beta, z_\gamma$.

The prover (signer) of the MPCitH-PP based digital signature has the secret witness (private key), so for the prover, there is no need to calculate the *Aux* phase and *Msgs* phase separately. As mentioned earlier, the purpose of the prover is to calculate **aux** and **msgs** to ensure that the verifier can verify that it has the secret, so the prover only needs to calculate **msgs** when calculating **aux**. However, in Figure 3, **aux** is calculated in the aux phase, so we need to calculate the secret in advance (the prover has the secret witness). First, run the circuit C and store the input wire z_α of all AND gates of the circuit, then calculate **aux**, and at the same time calculate the stored state z_α and XOR it with the mask λ_α to get $\hat{z}_\alpha$, so it can be guaranteed to be used to calculate **msgs** ($[s]$). We propose a new protocol that merges the calculations of the *Aux* and *Msgs* phases, eliminating redundant computations in Figure 5.

- ***Sample* phase** Π_C^{s}. The prover generates masks for all AND gates for all parties. Each party P_i has the following mask values.
  - $[\lambda_\alpha]_i$ (and $[\lambda_\beta]_i$) for the input wire α (and β) of each AND gate.
  - $[\lambda_{\alpha,\beta}]_{i \neq N}$ for each AND gate with input wires α and β.
- ***Compute* phase** Π_C^{c}**.** Prover precomputes the underlying circuit with the witness, stores the secret input values z_α, z_β of all AND gates, and sends them to the corresponding party P_i. Each party P_i evaluates the circuit C gate-by-gate by the order in the right of Figure 4.
  - For the linear operations X_j and O_j, the prover calculates using the input masks $(\lambda_\alpha^{j+1},\ \lambda_\beta^{j+1})$ for A_{j+1} of all parties and determines λ_γ^j by the inverse of the linear operation X_j.
  - For each AND gate of A_j, the prover computes λ_α^j and λ_β^j, and the error correction value $\mathbf{aux} = \left[\lambda_{\alpha,\beta}^j\right]_N := \lambda_\alpha^j \lambda_\beta^j \oplus \left[\lambda_{\alpha,\beta}^j\right]_1 \oplus \cdots \oplus \left[\lambda_{\alpha,\beta}^j\right]_{N-1} \oplus \lambda_\gamma^j$ for the party P_N. Then the prover computes $\hat{z}_\alpha^j = z_\alpha^j \oplus \lambda_\alpha^j$ and $\hat{z}_\beta^j = z_\beta^j \oplus \lambda_\beta^j$, calculates $[s^j]_i = \hat{z}_\alpha^j \left[\lambda_\beta^j\right]_i \oplus \hat{z}_\beta^j \left[\lambda_\alpha^j\right]_i \oplus \left[\lambda_{\alpha,\beta}^j\right]_i$ and sends $[s^j]_i$ to each participant P_i.

Figure 5: The *Sample* phase and *Compute* phase of our MPCitH protocol.

Each party just computes once the circuit C, thus the time required to calculate the MPC protocol Π has been reduced by half of the original time. The optimization we give requires secrets, so it can only be used for signing. For verification, it is still executed according to the original protocol. So we give a new protocol in Figure 6. The computational complexity of proving in the new protocol is $O(r \cdot (N+1) \cdot C_A + r \cdot C_X + r \cdot C_O)$. This optimization can only be applied to the prover who has the witness, while the verifier still needs to calculate Π_C^{a} and Π_C^{m}, which is the same as that in [KZ20]. Hence, the new protocol in Figure 6 optimizes the Commit computation while preserving the Challenge, Response and Verification computation. The computation of signature verifying remains unchanged.

Security analysis. The optimization in Figure 5 precomputes the underlying circuit with the witness, and caches the secret input values for all AND gates, which is used to update $\hat{z}_\alpha$ and $\hat{z}_\beta$ by XORing the $[\lambda_\alpha]_i$ and $[\lambda_\beta]_i$ respectively. The computation of $[s]_i$ only reduces redundant calculations. The precomputing of the underlying circuit does not result in any changes to the security of the MPCitH protocol (and the signature). For the same input, the output of the KKW protocol is the same as that of our optimized computation protocol.

5 Independence of the Mask

The KKW protocol computes the underlying symmetric primitive round-by-round. We optimize the proof of the KKW protocol by applying the circuit computation in the MPCitH protocol Figure 5, as shown in Figure 6.

Referring to Figure 5, the underlying MPCitH protocol operates as follows: The prover precomputes the underlying circuit using the witness, and stores the plain input wire values z_α, z_β of all AND gates in a look-up table T. These values are then used to update the masked circuit values $\hat{z}_\alpha, \hat{z}_\beta$ for all parties, which is essential for generating the global message tape.

For the masks used in the MPCitH-PP, we present the following lemma.

Lemma 1. *In the circuit computation of the round function of the underlying symmetric primitive in the MPCitH-PP protocol, the input masks λ_α and λ_β and the output mask λ_γ for each AND gate of nonlinear operation A are independent.*

For each AND gate in the non-linear operation A_j, the input mask values λ^j_α and λ^j_β are derived from the random tapes. The output mask λ^j_γ is computed using the inverse of X_j, incorporating the fresh random input mask of the layer A_{j+1} and the mask λ^j_O, as shown in Figure 7. Clearly, the mask λ^j_γ is independent of λ_α and λ_β. In order to make the AND operation hold, an addition correction mask $\lambda_{\alpha,\beta}$ is introduced to satisfy the equation $\lambda_\alpha \cdot \lambda_\beta = \lambda_\gamma \oplus \lambda_{\alpha,\beta}$.

Lemma 2. *Given all random tapes and precomputed plain input values for all the AND gates of the underlying symmetric primitive, the circuit computation of the round functions in the MPCitH-PP protocol can be processed in parallel with only one-round computation time cost.*

Proof. The purpose of the circuit computation Π^c_C for each round is to generate the error correction values **aux** and the broadcast message $[s]_i (i = 1, \dots, N)$ for each AND gate.

With all random tapes and the plain input wire values z_α, z_β of all AND gates stored in a look-up table T, the circuit computation of the round function is as follows.

In the j-th round, where $0 < j \leq r$, all random masks in the ***Sample*** **phase** are directly sampled from random tapes. For each nonlinear layer A_j, the mask share values $[\lambda^j_\alpha]_i$, $[\lambda^j_\beta]_i$ and $[\lambda^j_{\alpha,\beta}]_{i \neq N}$ from the random tape are used to mask the input of an AND gate for party P_i, satisfying:

$$\left[\lambda^j_\alpha\right]_1 \oplus \cdots \oplus \left[\lambda^j_\alpha\right]_N = \lambda^j_\alpha,\ \left[\lambda^j_\beta\right]_1 \oplus \cdots \oplus \left[\lambda^j_\beta\right]_N = \lambda^j_\beta. \quad (1)$$

In the ***Compute*** **phase**, input mask values λ^j_α and λ^j_β for an AND gate are deduced from all the corresponding mask shares with Equation 1. The output mask λ^j_γ is computed by the inverse of X_j with the new random input mask of the layer A_{j+1} and the mask λ^j_O, seen Figure 7. The prover then calculates the correction values **aux** using all masking shares as follows:

$$\mathbf{aux} = \left[\lambda^j_{\alpha,\beta}\right]_N = \lambda^j_\alpha \cdot \lambda^j_\beta \oplus \left[\lambda^j_{\alpha,\beta}\right]_1 \oplus \cdots \oplus \left[\lambda^j_{\alpha,\beta}\right]_{N-1} \oplus \lambda^j_\gamma. \quad (2)$$

New protocol

The prover and verifier receive circuit C as a statement, and the prover holds a witness $\mathrm{w} = (z_\alpha)_{\alpha \in \mathrm{IN}}$ such that $C(\mathrm{w}) = 1$. Values (M, N, τ) are parameters of the protocol. Let H denote a hash function, which can be modeled as the random oracle.

Commit

1. The prover chooses uniform random values $(\mathbf{seed}^*_1, \cdots, \mathbf{seed}^*_M)$, computes the circuit C, and stores the secret input wires of all AND gates. For each $j \in [1, M]$, the prover:
   (a) Use $\mathbf{seed}^*_j$ to generate $\mathbf{seed}_{j,1}, \cdots, \mathbf{seed}_{j,N}$. Compute the random tapes by running the *Sample* phase of MPC Π^{s}_C. Compute the masked witness $\hat{z}_{j,\alpha}$, $\mathbf{aux}_j \in \{0,1\}^{|C|}$, and $\mathbf{msgs}_j$ by running the *Compute* phase of MPC Π^{c}_C with the stored secrets of all AND gates. For $i = 1, \cdots, N-1$, let $\mathbf{state}_{j,i} := \mathbf{seed}_{j,i}$. Let $\mathbf{state}_{j,N} := \mathbf{seed}_{j,N} \| \mathbf{aux}_j$. Let $\mathbf{msgs}_{j,i}$ denote the messages broadcast by party P_i in this protocol execution, and $\mathbf{msgs}_j := \mathbf{msgs}_{j,1}, \cdots, \mathbf{msgs}_{j,N}$.
   (b) Commit to the *Compute* phase: For $i \in [1, N]$, compute $\mathbf{com}_{j,i} := \mathrm{H}(\mathbf{state}_{j,i})$. Compute $\mathbf{com}\text{-}\mathrm{a}_j := \mathrm{H}(\mathbf{com}_{j,1}, \cdots, \mathbf{com}_{j,N})$.
   (c) Compute $\mathbf{com}\text{-}\mathrm{m}_j := \mathrm{H}\left(\{\hat{z}_{j,\alpha}\}, \mathbf{msgs}_{j,1}, \cdots, \mathbf{msgs}_{j,N}\right)$.
2. Compute $h_{\mathrm{a}} = \mathrm{H}(\mathbf{com}\text{-}\mathrm{a}_1, \cdots, \mathbf{com}\text{-}\mathrm{a}_M)$ and $h_{\mathrm{m}} = \mathrm{H}(\mathbf{com}\text{-}\mathrm{m}_1, \cdots, \mathbf{com}\text{-}\mathrm{m}_M)$. Send $h^* = \mathrm{H}(h_{\mathrm{a}}, h_{\mathrm{m}})$ to the verifier.

Challenge The verifier sends the challenge: $(\mathcal{C}, \mathcal{P})$, where $\mathcal{C} \subset [1, M]$ is a set of size τ, and $\mathcal{P}$ is a list $\{p^\star_j\}_{j \in \mathcal{C}}$ with $p^\star_j \in [1, N]$.

Response For each $j \in [1, M] \backslash \mathcal{C}$, the prover sends $\mathbf{seed}^*_j, \mathbf{com}\text{-}\mathrm{m}_j$. Also, for each $j \in \mathcal{C}$, the prover seeds $\{\mathbf{state}_{j,i}\}_{i \neq p^\star_j}, \mathbf{com}_{j,p^\star_j}, \{\hat{z}_{j,\alpha}\}$, and $\mathbf{msgs}_{j,p^\star_j}$.

Verification The verifier accepts iff all the following checks succeed:

1. Check the *Aux* phase:
   (a) For every $j \in \mathcal{C}$ and $i \neq p^\star_j$, the verifier uses $\mathbf{state}_{j,i}$ to compute $\mathbf{com}_{j,i}$ by running the *Aux* phase Π^{a}_C. Then compute $\mathbf{com}\text{-}\mathrm{a}_j = \mathrm{H}(\mathrm{com}_{j,1}, \cdots, \mathrm{com}_{j,N})$ using the received value $\mathrm{com}_{j,p^\star_j}$.
   (b) For every $j \in [1, M] \backslash \mathcal{C}$ the verifier uses $\mathbf{seed}^*_j$ to compute $\mathbf{com}\text{-}\mathrm{a}_j$ as the prover would.
   (c) The verifier computes $h_{\mathrm{a}} = \mathrm{H}(\mathbf{com}\text{-}\mathrm{a}_1, \cdots, \mathbf{com}\text{-}\mathrm{a}_M)$.
2. Check the *Msgs* phase:
   (a) For $j \in \mathcal{C}$ the verifier simulates the *Msgs* phase Π^{m}_C using $\{\mathbf{state}_{j,i}\}_{i \neq p^\star_j}$, masked witness $\{\hat{z}_{j,\alpha}\}$, where $\alpha \in \mathrm{IN}$ and $\mathbf{msgs}_{j,i}$ to compute $\{\mathbf{msgs}_{j,i}\}_{i \neq p^\star_j}$. Then compute $\mathbf{com}\text{-}\mathrm{m}_j$ as if the prover would do.
   (b) The verifier computes $h_{\mathrm{m}} = \mathrm{H}(\mathbf{com}\text{-}\mathrm{m}_1, \cdots, \mathbf{com}\text{-}\mathrm{m}_M)$ using the received $\mathbf{com}\text{-}\mathrm{m}_j$ for $j \in [1, M] \backslash \mathcal{C}$.
3. The verifier checks that $\mathrm{H}(h_{\mathrm{a}}, h_{\mathrm{m}}) \stackrel{?}{=} h^*$.

Figure 6: The new proof system for a boolean circuit C.

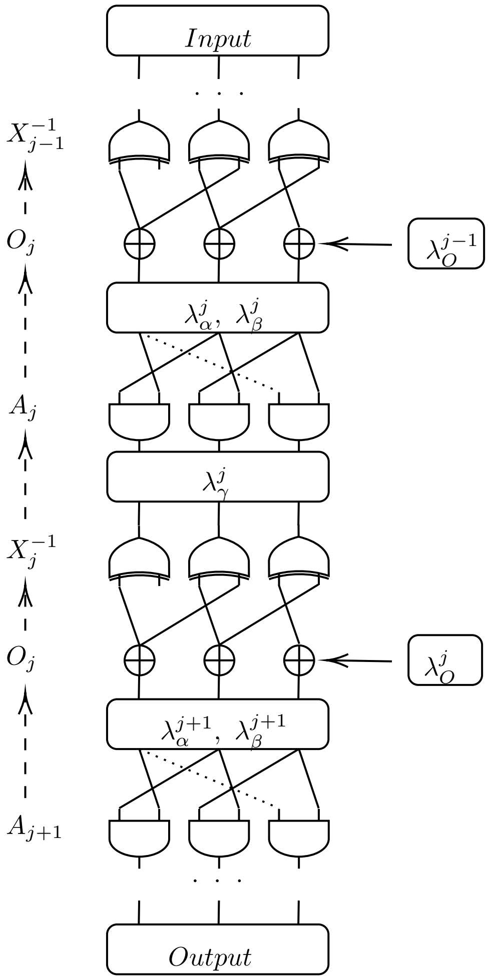

Figure 7: Serial computation for the general circuit for Π_C^c.

Subsequently, the prover updates the mask input wire values $\hat{z}_\alpha^j = z_\alpha^j \oplus \lambda_\alpha^j$ and $\hat{z}_\beta^j = z_\beta^j \oplus \lambda_\beta^j$ using the precomputed values z_α^j and z_β^j from table T. Finally, the global message tape is computed:

$$[s^j]_i = \hat{z}_\alpha^j \left[\lambda_\beta^j\right]_i \oplus \hat{z}_\beta^j \left[\lambda_\alpha^j\right]_i \oplus \left[\lambda_{\alpha,\beta}^j\right]_i, i = 1, \ldots, N. \tag{3}$$

Since random taps and precomputed values in table T suffice to compute **aux** and $[s]_i$, all r rounds can be processed in parallel. □

Lemma 2 is applicable not only to Picnic3 but to all circuits. The method precomputes and stores the plain values of the inputs for each round's AND gates. Then, based on these values and the corresponding masks from each participant, XOR calculations are performed to generate the necessary response values in the signature. This approach avoids the need for each participant to perform a separate symmetric encryption operation to generate the witness circuit.

We use Π_C^c to explain in detail the impact of Lemma 2 on computation. In Figure 7, recalling the original protocol, the prover's circuit for a round r is to calculate from O_r to A_1. However, according to Lemma 2, we can transform Figure 7 into Figure 8, and the prover can directly calculate each round in parallel. Lemma 2, shows that for

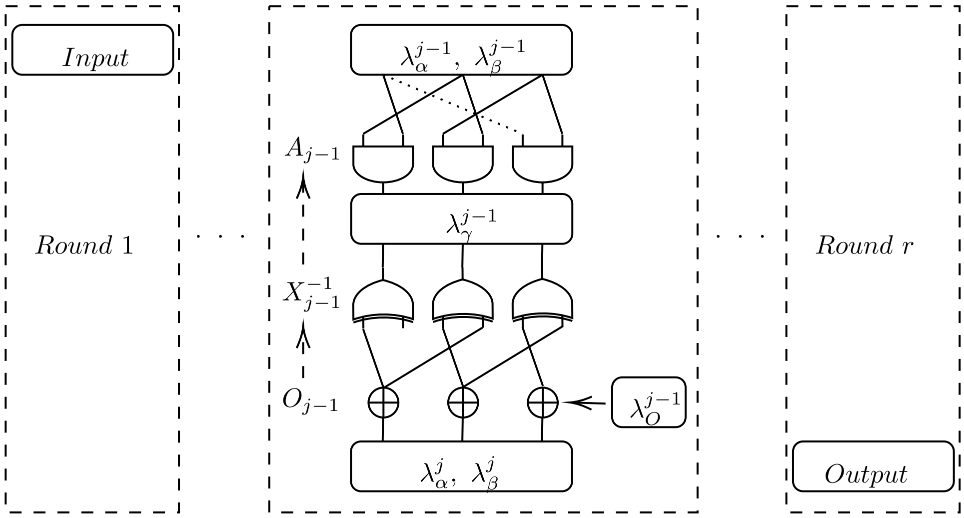

Figure 8: Parallel computation for the general circuit for Π_C^{c}.

any j-th round, its computation of Π_C^{c} is only related to the sampling information and precompuation states values with secret input, so **aux** and **msgs** can be directly calculated.

For Figure 5, this optimizes the prover's Π_C^{c}, and verifier's Π_C^{a} in Figure 3. However the verifier's Π_C^{m} cannot be applied with such optimization both in Figure 3 and Figure 5, because the verifier cannot get the secret witness. For the software implementation, we utilized more computational resources to speed up the implementation. This is a trade-off to get the signature faster.

This optimization reduces the time required for the r-round block cipher from r sequential rounds to a single parallel round. In hardware implementations, it reduces the original clock cycles required for MPC calculation by r-fold, achieving this improvement with minimal or no additional computing resources.

6 Implementation

In this section, we first describe the optimization implementation techniques for Section 4 and Section 5, and present the results of both software and hardware implementations. The software implementation results are based on the reference version as described in [KZ20] on the Ubuntu 22.04 system, while the hardware implementation results are based on the FPGA hardware version as described in [LJWJ24] on Kintex-7. The CPU of the experimental device is AMD Ryzen 9 5900HS, with 8 cores and 16 threads. It should be noted that the test methods for the original version of software implementation are from Picnic3-Software, and the test methods for the original version of hardware implementation are from Picnic3-Hardware. Our implementation is publicly available in Mask Is All You Need.

Table 2 shows the time cost of the original $\Pi_C^{\mathrm{a+m}}$ phase and the optimized Π_C^{c} phase of the software implementation at security levels L1, L3, and L5. Table 3 shows the time cost of the Π_C phase, other phases in the signing, and the sum time of signing in the original and optimized software implementations at L1, L3, and L5 security levels. Additionally, the values represent the average Π_C runtime cost over 100 iterations, with time given in milliseconds on the reference platform. Table 4 shows the hardware utilization, clock cycles, critical path and AT product of LowMC-MPC for 16 parties on the Kintex-7 after the hardware implementation is optimized at security levels L1 and L3.

6.1 Techniques of Optimal Implementation

First, we outline the technique of implementation in Section 4. We merge Π_C^{a} and Π_C^{m} into a single computation phase, denoted as Π_C^{c}. This integration reduces overall computation time by eliminating redundancies. Although the cost of Π_C^{c} is higher than that of either Π_C^{a} or Π_C^{m} alone, it is significantly cheaper than their combined cost. Specifically, both Π_C^{a} and Π_C^{m} require M computations of the LowMC circuit, resulting in a total of $2M$ computations. In contrast, the Π_C^{c} phase requires only $M+1$ computations of the LowMC circuit with the extra computation needed to obtain secret state information for each round before executing Π_C^{c}.

While Π_C^{c} consolidates the computations of Π_C^{a} and Π_C^{m} and eliminates redundant calculations, it still performs the essential computations of both phases. Consequently, its cost is slightly higher than the more expensive phase, but much lower than the total cost of both phases combined.

Next, we discuss optimal implementation techniques in Section 5. We demonstrate that the masks in the Π_C^{c} phase are independent and can therefore be computed directly. It is important to note that during the execution of the Π_C^{c} phase, the mask information of the witness must first be calculated serially. This involves reading the sampled mask information and multiplying it by the inverse matrix of M_0. The calculated mask information is then used in all subsequent parallel threads. Consequently, when the number of rounds r is small, the parallel phase Π_C^{c} does not reduce the time cost to the theoretical $1/r$ of the serial implementation for r rounds.

Another factor affecting the software implementation is highlighted in Section 4. Here, the input mask λ_α of the A_j layer is obtained by reading the sampled data. This data can also be used by the calculation of the X_{j-1}^{-1} layer and the O_{j-1} layer to determine the output mask of the A_{j-1} layer, as illustrated in Figure 7. Consequently, during the serial execution of the $(j-1)$-th and j-th rounds, the information of λ_α^j only needs to be read once to fulfill the calculation requirements of both the A_{j-1} and A_j layers.

However, in the optimization proposed in Section 5, the circuits and information between each round are independent. This means that the input mask λ_α^j of the A_j layer cannot be used directly by the circuit of the $(j-1)$ layer. Therefore, the method of reading the sampled data once is not feasible in this optimization. Figure 8 clearly shows that the sampled data needs to be read twice in an independent circuit: once for the input mask of the A_{j-1} layer in each round, and once for the input mask of the A_j layer (used for the output mask of the A_{j-1} layer obtained by the calculation of the X_{j-1}^{-1} layer and O_{j-1} layer).

As a result, compared to the optimization mentioned in Section 4, the parallel optimization proposed in Section 5 inevitably increases redundant data read operations. This also contributes to the factor that the Π_C^{c} phase, where each circuit is independent, does not achieve $1/r$ of the theoretical time reduction, particularly impacting the software implementation.

In general, assuming we ignore the impact of operations such as reading information on the overall time cost and only consider the impact of the vector-matrix multiplication algorithm and the S-box algorithm, we can summarize the time complexity of optimization Section 4 and optimization proposed in Section 5 as $O(r \cdot (N+1) \cdot C_A + r \cdot C_X + r \cdot C_O)$ and $O((N+1) \cdot C_A + C_X + C_O)$, respectively. It can be seen that when r is sufficiently large, the time cost of the optimization proposed in Section 5 is $1/r$ of the time cost of the optimization proposed in Section 4.

6.2 Results of Software Implementation

We first tested the time cost of the Π_C phase using the Picnic3 parameter set with the optimization proposed in Section 4, and the results are shown in Table 2. For each security

Table 2: The time cost of the original and the optimized Π_C phase under different security levels of the software implementation.

Scheme	Implementation Ref.	Time(ms)
LowMC-L1-$\Pi_C^{\mathrm{a+m}}$	Original	24.32
LowMC-L1-Π_C^{a}	Original	11.01
LowMC-L1-Π_C^{m}	Original	13.31
LowMC-L1-Π_C^{c}	Section 4	**19.55**
LowMC-L1-Π_C^{c}	Section 5	**8.32**
LowMC-L3-$\Pi_C^{\mathrm{a+m}}$	Original	56.63
LowMC-L3-Π_C^{a}	Original	24.56
LowMC-L3-Π_C^{m}	Original	32.07
LowMC-L3-Π_C^{c}	Section 4	**45.81**
LowMC-L3-Π_C^{c}	Section 5	**18.07**
LowMC-L5-$\Pi_C^{\mathrm{a+m}}$	Original	208.84
LowMC-L5-Π_C^{a}	Original	102.22
LowMC-L5-Π_C^{m}	Original	106.62
LowMC-L5-Π_C^{c}	Section 4	**136.28**
LowMC-L5-Π_C^{c}	Section 5	**53.41**

Table 3: The time cost of the Π_C phase, other phases in the signing, and the sum time of signing under different security levels of the software implementation.

Scheme	Implementation Ref.	Part	Time(ms)
Picnic3-L1-sign	Original	$\Pi_C^{\mathrm{a+m}}$	24.32
		Others	37.36
		Sum	**61.68**
Picnic3-L1-sign	Section 4	Π_C^{c}	19.55
		Others	34.94
		Sum	**54.49**
Picnic3-L1-sign	Section 5	Π_C^{c}	8.32
		Others	40.20
		Sum	**48.52**
Picnic3-L3-sign	Original	$\Pi_C^{\mathrm{a+m}}$	56.63
		Others	82.23
		Sum	**138.86**
Picnic3-L3-sign	Section 4	Π_C^{c}	45.81
		Others	74.80
		Sum	**120.61**
Picnic3-L3-sign	Section 5	Π_C^{c}	18.07
		Others	86.33
		Sum	**104.40**
Picnic3-L5-sign	Original	$\Pi_C^{\mathrm{a+m}}$	208.84
		Others	128.74
		Sum	**337.58**
Picnic3-L5-sign	Section 4	Π_C^{c}	136.28
		Others	109.44
		Sum	**245.72**
Picnic3-L5-sign	Section 5	Π_C^{c}	53.41
		Others	124.76
		Sum	**178.17**

level, we tested the cost time of the original Π_C^{a} phase, the original Π_C^{m} phase, and the optimized Π_C^{c} phase in Section 4, where the original version is given by [KZ20]. At the L1 and L3 security levels, the cost of the Π_C^{c} phase is approximately 74% of the $\Pi_C^{\mathrm{a+m}}$

phase. At the L5 security level, the cost of the Π_C^{c} phase is approximately 62% of the $\Pi_C^{\mathrm{a+m}}$ phase.

Additionally, we tested the time cost of signing with the optimization proposed in Section 4 at different security levels compared to the original signing time, with the results shown in Table 3. At the L1 and L3 security levels, the cost of signing is about 88% of the original signing. At the L5 security level, the cost of signing is approximately 73% of the original signing. These results validate our theoretical analysis as mentioned in Section 4.

Next, we tested the time cost of the optimized Π_C^{c} phase and the signing process with a round number $r = 4$ at different security levels by using the optimization proposed in Section 5, as shown in Table 2 and Table 3. Our software implementation uses four threads here, as the number of LowMC encryption rounds in the algorithm we test is four. From Table 2, at security levels L1 and L3, the time cost of the optimized Π_C^{c} phase is about 30% of the original $\Pi_C^{\mathrm{a+m}}$ phase, and at the L5 security level, it is about 26%. At security levels L1, L3, and L5, the time cost of the parallel optimized Π_C^{c} phase is approximately 40% of the optimized Π_C^{c} phase in Section 4, which is consistent with our theoretical analysis.

From Table 3, at security levels L1 and L3, the signing time cost (in Section 5) is optimized to about 78% of the original signing time cost. At the L5 security level, it is reduced to about 53% of the original signing time cost. The reason for the more significant improvement at the L5 security level compared to L1 and L3 is that the time cost of the $\Pi_C^{\mathrm{a+m}}$ phase at the L5 security level constitutes a larger proportion of the overall signing time cost than other operations. Therefore, the improvement is more pronounced.

It is worth noting that although the time performance has improved, more computing resources are required to achieve simultaneous computing using four threads across four CPU cores.

6.3 Results of Hardware Implementation

In [LJWJ24], it takes $3r$ clock cycles to calculate the block cipher LowMC, where $2r$ clock cycles are used to compute Π_C^{a} phase and r clock cycles are used to compute Π_C^{m} phase. To prevent the critical path from becoming too long, the calculation of Π_C^{a} phase first requires computing the key scheduling matrix M_i, followed by the inverse of the linear layer L_i^{-1}. To reduce the clock cycle, the position of the XOR key can be modified so that the equivalent key is directly XORed after the S-box layer, which requires additional computing resources for $L_i^{-1} \cdot M_i$. The computation of the key scheduling matrix M_i and the linear layer L_i for Π_C^{m} phase can be performed simultaneously, thus requiring only r clock cycles.

$\Pi_C^{\mathrm{a+m}}$ phase uses only the inverse of the linear layer L_i^{-1} and the key scheduling matrix M_i. Therefore, $\Pi_C^{\mathrm{a+m}}$ phase seems capable of reducing hardware usage, and reducing the clock cycle count from r rounds to a single round. However, since the optimization in Section 4 requires pre-computation of LowMC, the actual reduction in hardware resources is not realized. This suggests a new approach to reduce hardware usage: if the secret of the AND gate input wire of the circuit can be generated during key generation and transmitted to the FPGA, hardware resource usage can be genuinely reduced, albeit at the cost of increased transmission.

As shown in Table 4, at the L1 security level, the hardware usage of Π_C^{c} phase with 2 clock cycles is 69.3% of that for $\Pi_C^{\mathrm{a+m}}$ phase with 12 clock cycles, and 60.8% of that for $\Pi_C^{\mathrm{a+m}}$ phase with 8 clock cycles. At the L5 security level, the hardware usage of Π_C^{c} phase with 2 clock cycles is 68.4% of that for $\Pi_C^{\mathrm{a+m}}$ with 12 clock cycles, and 50.7% of that for $\Pi_C^{\mathrm{a+m}}$ phase with 8 clock cycles. Therefore, increasing the communication size of the FPGA is a meaningful way to reduce hardware resource usage. The critical path of our hardware implementation is better than [LJWJ24]'s implementation, especially for the parallel version, because the logic control of the parallel version is simpler, so the critical path is shorter. We provide an area-time (AT) product, and the new hardware implementation AT performs better, especially for parallel computing. Note that parallel computing does not require a lot of hardware resources. This is because for the hardware implementation of LowMC, each linear matrix is a different matrix of size $n \times n$, so the resources occupied are large, and the resources added by parallelism are small compared to the matrix. Understandably, the latency of the new implementation will not increase, because the calculation was previously done round by round, but now all rounds are calculated simultaneously, and each round is independent, so no additional critical path is added.

When it comes to optimization proposed in Section 5, the parallel computing implemented in software requires multiple cores to be realized, but in hardware implementation, especially in ASIC and FPGA, parallelism is very natural. For Π_C^{com}, due to a large number of matrices, parallel computing only adds a few registers compared to serial computing. It can also be found in Table 4 that the additional resources consumed by parallel computing are very small, which is equivalent to the original implementation. Parallel optimization can reduce both hardware usage and computing clock cycles.

The mask independence optimization performs very well for hardware implementation. In addition to reducing the clock cycle, it also ensures that the hardware implementation only increases a little. In [LJWJ24], they designed a pipeline for digital signatures based on MPCitH-PP. The maximum clock cycle of each module of the pipeline limits the performance of the pipeline. The clock cycle of the pipeline module mainly depends on the number of rounds of the block cipher and hash function. Since the hardware usage of LowMC is very large, they can only implement the Picnic3 algorithm with 4 parties. Using this optimization, more parties can be implemented in the same clock cycle with increasing few hardware usage. For example, in [LJWJ24], they use 8/12 clock cycles to complete the calculation of LowMC, and this optimization can complete 4/6 calculations of 4 parties in 8/12 clock cycles.

Table 4: Hardware utilization and critical path of LowMC-MPC for 16 parties on the Kintex-7 (modified from [LJWJ24]). To simplify the results, we mainly use LUT as the area measurement standard, and AT product is calculated by LUT × ClockCycle × CriticalPath.

Scheme	Optimization	Utilization				Clock	Critical	AT Product
		LUTs	% LUTs	FFs	% FFs	Cycles	Path	(#LUTs · ns)[1]
LowMC-L1-$\Pi_C^{\mathrm{a+m}}$	Original	36264	12.14%	9167	1.53%	12	6.583 ns	2864711
LowMC-L1-$\Pi_C^{\mathrm{a+m}}$	Original	41378	13.86%	9198	1.54%	8	7.032 ns	2327761
LowMC-L1-Π_C^{c}	Section 4	25146	8.42%	9038	1.51%	8	5.622 ns	1130967
LowMC-L1-Π_C^{c}	Section 5	28100	9.41%	9941	1.66%	2	4.154 ns	233455
LowMC-L5-$\Pi_C^{\mathrm{a+m}}$	Original	128668	43.09%	18149	3.04%	12	7.716 ns	11913627
LowMC-L5-$\Pi_C^{\mathrm{a+m}}$	Original	148211	49.64%	18116	3.03%	8	8.098 ns	9601701
LowMC-L5-Π_C^{c}	Section 4	75198	25.18%	18878	3.15%	8	5.932 ns	3568596
LowMC-L5-Π_C^{c}	Section 5	84698	28.36%	19638	3.28%	2	4.436 ns	751441

[1] #LUT represents the number of LUT.

7 Conclusion

In this paper, we revisited the MPCitH-PP construction within the KKW protocol, restructuring it into three phases and proposing significant optimizations to enhance its efficiency. By analyzing both the offline and online phases, we identified redundant computations and merged them into a single phase, leveraging the independence of random masks to enable parallel calculations. These optimizations led to a more efficient protocol for MPCitH-PP, suitable for both software and hardware implementations.

Through experimental verification using Picnic3, we achieved substantial performance improvements. At the L1 security level, our optimized software implementation reduces the calculation time of MPCitH-PP to approximately 74% of the previous solution, with further reductions to around 30% when parallelism is employed. The signature scheme also shows improved efficiency, operating at about 88% of the previous time, and 73% with parallelism. At the L5 security level, our optimizations reduce MPCitH-PP calculations to approximately 62% of the previous solution, and 26% with parallelism; the signature scheme runs at about 78% and 53% with parallelism, respectively. At the hardware level, our enhancements reduce the required clock cycles from 12 or 8 rounds to just 2 rounds, with negligible impact on hardware usage.

These results highlight the effectiveness of our proposed optimizations and demonstrate their potential to make post-quantum digital signature schemes more practical and efficient for real-world applications. Future work could investigate extending these optimizations to other "in the head" techniques, such as VOLEitH, to further improve the efficiency of PQ signatures.

Acknowledgements

The authors sincerely thank the anonymous reviewers of TCHES 2025 for providing valuable comments to help us improve the overall quality of the paper.

This work is supported by the National Key R&D Program of China (No. 2018YFA0704700), the National Natural Science Foundation of China (Grant Nos. 62072270, 62032014, U2336207 and 62302250), Department of Science & Technology of Shandong Province (No.SYS202201), Quan Cheng Laboratory (Grant Nos. QCLZD202301, QCLZD202306), the Young Elite Scientists Sponsorship Program by CAST (2023QNRC001), and the National Key R&D Program of China (No. 2024YFA1013003).

References

[ABB+23] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, Matthieu Rivain, and Jean-Pierre Tillich. Mira specifications. hal-04315820f, 2023.

[ABB+24] Gora Adj, Stefano Barbero, Emanuele Bellini, Andre Esser, Luis Rivera-Zamarripa, Carlo Sanna, Javier A. Verbel, and Floyd Zweydinger. Mirith: Efficient post-quantum signatures from minrank in the head. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):304–328, 2024.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

[BBD+24] Slim Bettaieb, Loïc Bidoux, Victor Dyseryn, Andre Esser, Philippe Gaborit, Mukul Kulkarni, and Marco Palumbi. PERK: compact signature scheme based on a new variant of the permuted kernel problem. *Des. Codes Cryptogr.*, 92(8):2131–2157, 2024.

[BBdSG+23] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. In

Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 581–615. Springer, 2023.

[BCF+23] Loïc Bidoux, Jesús-Javier Chi-Domínguez, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Matthieu Rivain, and Adrien Vinçotte. RYDE: A digital signature scheme based on rank-syndrome-decoding problem with mpcith paradigm. *CoRR*, abs/2307.08726, 2023.

[BDPA11] Guido Bertoni, Joan Daemen, Micha´el Peeters, and Gilles Van Assche. Cryptographic sponge functions. *Submission to NIST (Round 3)*, 2011. https://sponge.noekeon.org/CSF-0.1.pdf.

[Beu20] Ward Beullens. Sigma protocols for mq, PKP and sis, and fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 183–211. Springer, 2020.

[BFR23] Ryad Benadjila, Thibauld Feneuil, and Matthieu Rivain. MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. *IACR Cryptol. ePrint Arch.*, page 1719, 2023.

[BKPV24] Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier A. Verbel. Biscuit: New mpcith signature scheme from structured multivariate polynomials. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part I*, volume 14583 of *Lecture Notes in Computer Science*, pages 457–486. Springer, 2024.

[BN20] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, 2020.

[CCJ23] Eliana Carozza, Geoffroy Couteau, and Antoine Joux. Short signatures from regular syndrome decoding in the head. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 532–563. Springer, 2023.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842. ACM, 2017.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[dSGMOS19] Cyprien Delpech de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: using AES in picnic signatures. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers*, volume 11959 of *Lecture Notes in Computer Science*, pages 669–692. Springer, 2019.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 621–641. Springer, 2013.

[FR] Thibauld Feneuil and Matthieu Rivain. Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. *IACR Cryptol. ePrint Arch.*, page 1573.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 1069–1083. USENIX Association, 2016.

[HBD+20] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. *Technical report, National Institute of Standards and Technology*, 2020. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from Secure Multiparty Computation. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 21–30. ACM, 2007.

[KHS+23] Seongkwang Kim, Jincheol Ha, Mincheol Son, ByeongHak Lee, Dukjae Moon, Joohee Lee, Sangyub Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. AIM: symmetric primitive for shorter signatures with stronger security. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on*

Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023, pages 401–415. ACM, 2023.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 525–537. ACM, 2018.

[KZ20] Daniel Kales and Greg Zaverucha. Improving the performance of the picnic signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):154–188, 2020.

[LJWJ24] Guoxiao Liu, Keting Jia, Puwen Wei, and Lei Ju. High-Performance Hardware Implementation of MPCitH and Picnic3. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):190–214, 2024.

[LLDL+20] Vadim Lyubashevsky, Eike Kiltz Léo Ducas, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. *Technical report, National Institute of Standards and Technology*, 2020. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[PFH+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. *Technical report, National Institute of Standards and Technology*, 2020. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[Pic20] Picnic Design Team. An implementation of the LowMC block cipher family, 2020. https://github.com/microsoft/Picnic/blob/master/spec/spec-v3.0.pdf.

[ZCD+20] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. Picnic. *Technical report, National Institute of Standards and Technology*, 2020. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[ZWX+22] Handong Zhang, Puwen Wei, Haiyang Xue, Yi Deng, Jinsong Li, Wei Wang, and Guoxiao Liu. Resumable Zero-Knowledge for Circuits from Symmetric Key Primitives. In Khoa Nguyen, Guomin Yang, Fuchun Guo, and Willy Susilo, editors, *Information Security and Privacy - 27th Australasian Conference, ACISP 2022, Wollongong, NSW, Australia, November 28-30, 2022, Proceedings*, volume 13494 of *Lecture Notes in Computer Science*, pages 375–398. Springer, 2022.