High-Performance Hardware Implementation of MPCitH and Picnic3

Anonymous Submission

Abstract. Picnic is a post-quantum digital signature, the security of which relies solely on symmetric-key primitives such as block ciphers and hash functions instead of number theoretic assumptions. One of the main concerns of Picnic is the large signature size. Although Katz et al.'s protocol (MPCitH-PP) significantly reduces the size of Picnic, the involvement of more parties in MPCitH-PP leads to longer signing/verification times and more hardware resources. This poses new challenges for implementing high-performance Picnic on resource-constrained FPGAs. So far as we know, current works on the hardware implementation of MPCitH-based signatures are compatible with 3 parties only. In this work, we investigate the optimization of the implementation of MPCitH-PP and successfully deploying MPCitH-PP with more than three parties on resource-constrained FPGAs, e.g., Xilinx Artix-7 and Kintex-7, for the first time. In particular, we propose a series of optimizations, which include pipelining and parallel optimization for MPCitH-PP and the optimization of the underlying symmetric primitives. Besides, we make a slight modification to the computation of the offline commitment, which can further reduce the number of computations of Keccak. These optimizations significantly improve the hardware performance of Picnic3. Signing messages on our FPGA takes 0.047 ms for the L1 security level, outperforming Picnic1 with hardware by a factor of about 5.3, which is the fastest implementation of post-quantum signatures as far as we know. Our FPGA implementation for the L5 security level takes 0.146 ms beating Picnic1 by a factor of 8.5, and outperforming Sphincs by a factor of 17.3.

Keywords: FPGA · MPCitH · Picnic · LowMC

1 Introduction

The emergence of quantum computers presents a potential threat to the security of commonly used cryptographic schemes [Sho94, Gro96]. To address this concern, the US National Institute of Standards and Technology (NIST) is currently undertaking the Standardization Process for Post-Quantum Cryptography (PQC). Picnic [CDG+20], which has been selected as one of the digital signature candidates in the third round, has attracted significant attention among all the post-quantum candidates of NIST [Nat23]. Picnic's security does not rely on any number theoretic assumptions but instead solely on symmetric-key primitives, such as block cipher. This unique property is made possible by the MPC-in-the-head (MPCitH) paradigm, a novel method for constructing zero-knowledge proofs. MPCitH allows the prover to prove the correctness of a statement without revealing any additional information by simulating the executions of multi-party computation (MPC) protocols. As stated in [Nat20], "NIST also sees Picnic reliance on only assumptions about symmetric primitives as an advantage in case the need arises for an extremely conservative signature standard in the future".

The early version of Picnic, called Picnic1, suffers from a larger signature size compared to other post-quantum candidates since the proof size of MPCitH is linear with the number of non-linear operations of the underlying block cipher, i.e., LowMC. To overcome this problem, Katz et al. [KKW18] introduced MPCitH with preprocessing

(MPCitH-PP), which reduces the size of Picnic1 dramatically by involving more parties in the MPCitH protocol. The resulting scheme, called Picnic2, allows for a much smaller signature size, but the involvement of more parties leads to longer signing/verification times compared to Picnic1. Moreover, MPCitH-PP in Picnic2 poses new challenges for implementing MPCitH-based digital signatures on hardware, particularly considering the hardware implementation requirements of NIST [AASA+19]. So far as we know, previous works [KRR+20, Wal19] on the hardware implementation of MPCitH-based digital signatures have only implemented signatures with 3 parties. It is challenging to implement high-performance Picnic2 on resource-constrained FPGAs because more parties in Picnic2 increase the running time of signing and verification and significantly impact hardware resources.

Recently, several optimizations [MZ17, MR18, KZ20, MGH+23] have been developed for MPCitH-PP, particularly in the case of Picnic3 [KZ20]. These optimizations improve the linear calculation method in the underlying circuit of MPCitH-PP, resulting in an N-fold increase in linear calculation speed, where N represents the number of parties. While these optimizations provide a promising avenue for implementing digital signatures based on MPCitH-PP for more parties, there is currently a lack of hardware-level implementation of these techniques. It is, therefore, a natural question to ask *whether we can deploy MPCitH-PP with more than three parties on a resources-constrained FPGA board, and if so, what the resulting performance of Picnic would be, as well as any additional limitations that may arise.*

1.1 Contributions

In this work, we focus on optimizing the implementation of MPCitH-PP and successfully deploying MPCitH-PP with more than three parties on resource-constrained FPGAs (Xilinx Artix-7 and Kintex-7) for the first time, resulting in significantly improved performance of Picnic3.

We first define a general gate-level circuit model for MPCitH-PP and apply the linear swapping and optimized mask sampling proposed by Kales and Zaverucha [KZ20] to improve the implementation of MPCitH-PP. We have analyzed and demonstrated that in general boolean circuits, these optimizations reduce the computational complexity of the underlying circuit in MPCitH-PP from $\mathcal{O}(2N \cdot (C_L + C_N))$ to $\mathcal{O}(2N \cdot C_N + 2C_L)$, where C_N represents the number of nonlinear operations in the circuit and C_L represents the number of linear operations. This optimization is applicable to any circuit with invertible linear operation based on the KKW protocol. In particular, we implemented LowMC from 3 parties to 16 parties, and the experimental results show that the hardware usage no longer increases linearly with the parties.

Building on this, we propose several optimizations for the hardware implementation of Picnic3 so that Picnic3 can better utilize the resources of the FPGA development board to achieve high-performance implementation.

- Pipelining and parallel optimization. We performed a detailed study of the Picnic3 signature and divided it into three distinct steps. The construction of a Merkel tree is encompassed within the first and third steps, while the second step focuses on the repetitive instantiation of zero-knowledge proofs. To enhance efficiency, we fragmented the second stage into multiple parts and devised a multi-level pipeline structure, resulting in a substantial reduction in time consumption. Furthermore, leveraging the fundamental attributes of FPGA, we orchestrated widespread parallel computing across all three steps.

- Optimization of symmetric primitives. LowMC has a longer critical path, while Keccak has a shorter critical path. Since the clock cycle of Keccak is larger than that of LowMC, we investigate the clock cycle and critical path of symmetric primitives. To design an optimized version for a high-performance implementation of digital signatures, we reduce Keccak's clock cycles so that its critical path does not exceed (or slightly exceed) that of LowMC.

- Extending the pipeline construction covering the first and third steps. The Merkel Tree needs to be stored in the BRAM. We precompute the nodes from the root to the second-to-last layer and store them. For each instance, only the last leaf nodes are needed to compute, which has a similar time interval to that of the pipeline in step 2. For step 3, two independent Keccak components with optimized critical paths are incorporated into the pipeline in step 2 to save the running time of $\mathcal{O}(M)$ calls to Keccak, where M is the number of instances.

- To further reduce the number of computations of Keccak, we make a slight modification on the computation of the final offline commitment, which does not affect the security of the KKW protocol and Picnic3.

By combining all the above optimizations, our FPGA implementation of Picnic3 with 4 parties achieves significantly improved performance. Concretely, signing messages on our FPGA takes 0.047 ms for the L1 security level beating Picnic3 with software by a factor of about 110, and outperforming Picnic1 with hardware by a factor of about 5.3, which is the fastest implementation of post-quantum signatures seen Table 10. Our FPGA implementation for the L5 security level takes 0.146 ms beating Picnic1 by a factor of 8.5, and outperforming the NIST selected signature SPHINCS by a factor of 17.3.

1.2 Related Work

The MPCitH paradigm has seen significant advancements since Ishai et al.'s work [IKOS07]. Notably, ZKBoo [GMO16] and ZKB++ [CDG+17] have significantly advanced the practicality of MPCitH, culminating in the submission of Picnic1 to Round 1 of the NIST PQC Standardization Process. Katz et al. [KKW18] extended the paradigm to MPCitH-PP, leading to Picnic2. Furthermore, Kales and Zaverucha [KZ20] optimized the structure of LowMC [ARS+15] utilized in Picnic2 and reduced the time required for signature generation and verification, resulting in the development of Picnic3.

Efficient hardware implementations have become a focal point of extensive research, further accentuated by the NIST post-quantum standardization project [AASA+19]. On the hardware implementations of MPCitH or Picnic algorithms, only Picnic1 is currently available in a hardware-enable format [KRR+20]. In the case of Picnic1, the utilization of FPGA resources is exceedingly substantial. [KRR+20] argues that Picnic2 allows for shorter signatures, but performing the simulated MPC protocol with a larger number of parties results in longer signing and verification times compared to Picnic1. [Wal19] proposed an efficient VHDL implementation of Picnic2, therefore, requires at least 63 LowMC instances which would not fit on any FPGA. Implementing the MPCitH-PP digital signature, involving more parties and requiring pre-computation, becomes challenging to deploy on a resource-constrained FPGA. Picnic3 introduces optimizations to the LowMC computation of MPCitH-PP, thereby reducing the calculation of the linear layer. To the best of our knowledge, no prior work has investigated the hardware implementations of MPCitH-PP and Picnic3.

2 Preliminaries

Notation. Let L denote an NP language. The NP relation is defined as $R(\mathbf{x}, \mathbf{w}) = 1$ if $x \in L$ 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 Symmetric Primitives

The underlying symmetric primitives of Picnic3 are block cipher and hash function, which are instantiated by LowMC and SHAKE, respectively. The security assumption (one-way function) is LowMC. The hash function Keccak is used for random number generation and commitment.

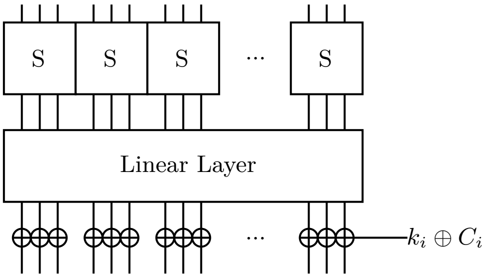

Figure 1: One round of encryption with LowMC in Picnic3.

Block cipher. LowMC [ARS$^+$15] is a family of lightweight SPN-based block ciphers. One of the key advantages of LowMC is its low multiplicative complexity, e.g., small AND gate/depth. This property makes LowMC well-suited for a range of cryptographic applications, including multi-party computation, fully homomorphic encryption, and zero-knowledge proofs. LowMC encryption starts with an initial whitening by XORing the first round key to the plaintext, followed by r rounds. As depicted in Figure 1, one round consists of four steps: (i) SboxLayer, (ii) LinearLayer, (iii) ConstantAddition and (iv) KeyAddition, i.e., LowMCRound(i) = KeyAddition ∘ ConstantAddition ∘ LinearLayer ∘ SboxLayer(i).

Algorithm 1: LowMC encryption.

```
Input: plaintext $p \in \mathbb{F}_2^n$ and key $mk \in \mathbb{F}_2^k$.
Output: ciphertext $c \in \mathbb{F}_2^n$.
$state \leftarrow K_0 \cdot mk + p$
foreach $i \in [1, r]$ do
    $state \leftarrow$ SboxLayer($state$)
    $state \leftarrow L_i \cdot state$          ▷ LinearLayer
    $state \leftarrow C_i + state$          ▷ ConstantAddition
    $state \leftarrow K_i \cdot mk + state$          ▷ KeyAddition
end
$c \leftarrow state$
```

In the SboxLayer, m 3-bit Sboxes are typically applied to the initial $3 \cdot m$ bits of the state. The SboxLayer does not modify the remaining bits of the state. Note that in Picnic3, the parameter n is set to $3 \cdot m$ in order to decrease the number of rounds r. The specific parameters are $(n, k, m, r) \in \{(129, 129, 43, 4), (192, 192, 64, 4), (255, 255, 85, 4)\}$ [Pic20], where k denotes the key size. Sbox is defined as $S(a, b, c) = (a \oplus b \cdot c, a \oplus b \oplus a \cdot c, a \oplus b \oplus c \oplus a \cdot b)$ with three bits inputs and outputs. In LinearLayer, the state is multiplied with a pseudorandomly generated matrix $L_i \in \mathbb{F}_2^{n \times n}$. The matrices are chosen pseudorandomly from the set of all invertible binary $n \times n$ matrices during the instantiation of LowMC. During ConstantAddition the vector $C_i \in \mathbb{F}_2^n$ is XORed to the state. During KeyAddition, the round key of the current round is XORed to the state. All round keys are generated as a result of the multiplication of the master key with the matrix $K_i \in \mathbb{F}_2^{n \times k}$. The matrices are chosen pseudorandomly from the set of all full-rank binary $n \times k$ matrices during the instantiation of LowMC. The full description of the encryption algorithm is given in Algorithm 1. Despite findings by [Din21a, Din21b, LMSI22, SCW23] questioning the security claims of LowMC, this symmetric cipher still finds valuable applications in the fields of fully homomorphic encryption and secure multi-party computation. It was demonstrated by [Din21a] that the parameter sets for L3 and L5 are insufficient to guarantee 192 (resp. 256) bits of security, as the attacks only require 2^{188} (resp. 2^{245}) operations with large memory requirements of 2^{164} (resp. 2^{219}) bits.

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

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

2.2 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 $f_{\mathbf{x}}(\mathbf{w}) = R(\mathbf{x}, \mathbf{w})$. In Picnic, $f_{\mathbf{x}}(\mathbf{w}) := \mathrm{LowMC}(sk, p) \overset{?}{=} c$, where $\mathbf{x} = (p, c)$ represents a plaintext-ciphertext pair and $\mathbf{w} = sk$ denotes the private key. In this case, the prover P proves knowledge of a private key that generates a specific public ciphertext from the corresponding public plaintext.

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 $\mathbf{w}$. To enhance compatibility with hardware implementation and Picnic3 signatures, we utilize boolean circuits instead of arithmetic circuits for LowMC. 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 party $\mathrm{P}_1, \cdots, \mathrm{P}_N$ 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 [FS87]). 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 [dSGDMOS20, BN20, BdSGK+21, KZ20, MGH+23, ZWX+22, KHS+22, AMGH+23, BBdSG+23, BFR23, ABB+23, BKPV23]. 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 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 $[\lambda_\alpha]_i$.

- **Offline phase Π_C^{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, $[\lambda_{\alpha,\beta}]_N := \lambda_\alpha \lambda_\beta \oplus [\lambda_{\alpha,\beta}]_1 \oplus \cdots [\lambda_{\alpha,\beta}]_{N-1}$, 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 Π_C^{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 $s = [s]_0 \oplus \cdots \oplus [s]_N$, 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.

KKW Protocol. We briefly recall the basic framework of KKW for one MPC instance, which is a three-round MPCitH-PP system. In Figure 2, we provide a complete description of the KKW proof system that 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 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.

2.3 Picnic and Its Parameters

Using the well-known Fiat-Shamir transform, KKW described above can be transformed into a non-interactive version or a digital signature, where the message m to be signed is hashed and incorporated into the challenge. The signature scheme Picnic is a concrete instantiation of the non-interactive KKW, where C is instantiated with the boolean circuit of LowMC. More precisely, the signer's secret key sk is the witness $\mathbf{w}$, and the public key corresponds to the statement $\mathbf{x} = (p, c)$, which is a pair of plaintext and ciphertext. A signature involves a proof of knowledge of sk that satisfies $\text{LowMC}_{sk}(p) = c$.

The parameter sets for the algorithms submitted to the NIST competition must meet one of five security levels. Picnic defines parameters for security levels L1, L3 and L5, corresponding to the security of AES 128, 192 and 256, respectively. This work implements the L1 and L5 versions of Picnic3. The corresponding parameters (n, k, m, r) for LowMC in L1 and L5 are $(129, 129, 43, 4)$ and $(255, 255, 85, 4)$, respectively. Table 2 shows the parameters of different Picnic versions. Note that Picnic3 (based on KKW) offers better efficiency in terms of signature size compared to Picnic1 (based on ZKB++), but it has lower runtime performance. Additionally, Table 2 shows the different key and signature sizes for the Picnic instances. More details of parameter sets for the ZKB++ proof system, KKW proof system, and specific parameters chosen for LowMC are shown in the Picnic specification and NIST submission [Pic20].

Table 2: Picnic parameters.

Scheme	**Parameters**			**Sizes (byte)**		
	M	τ	N	**pk**	**sk**	σ
Picnic1-L1	219	219	3	32	16	34032
Picnic3-L1	206	66	4	34	17	19573
Picnic3-L1	252	36	16	34	17	12590
Picnic1-L5	438	438	3	64	32	132856
Picnic3-L5	401	133	4	64	32	75721
Picnic3-L5	604	68	16	64	32	53274

2.4 Test Platform

To facilitate better comparisons with previous work [KRR+20], we utilized the Xilinx Kintex-7 and Artix-7 as our experimental platforms. The latter is one of the optimization

KKW 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 $(\mathbf{seed}_1^*, \cdots, \mathbf{seed}_M^*)$. 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 $\mathbf{aux}_j \in \{0,1\}^{|C|}$ by running the offline phase of MPC Π_C^{off}. For $i = 1, \cdots, N-1$, let $\mathbf{state}_{j,i} := \mathbf{seed}_{j,i}$. Let $\mathbf{state}_{j,N} := \mathbf{seed}_{j,N} \| \mathbf{aux}_j$.

   (b) Commit to the offline phase: For $i \in [1, N]$, compute $\mathbf{com}_{j,i} := \mathrm{H}\left(\mathbf{state}_{j,i}\right)$. Compute $\mathbf{com}\text{-off}_j := \mathrm{H}\left(\mathbf{com}_{j,1}, \cdots, \mathbf{com}_{j,N}\right)$.

   (c) Simulate the online phase of MPC Π_C^{on} using $\{\mathbf{state}_{j,i}\}$, beginning by computing the masked witness $\{\hat{z}_{j,\alpha}\}$, where $\alpha \in \mathrm{IN}$. Let $\mathbf{msgs}_{j,i}$ denote the messages broadcast by party P_i in this protocol execution.

   (d) Commit to the online phase: Compute $\mathbf{com}\text{-on}_j := \mathrm{H}\left(\{\hat{z}_{j,\alpha}\}, \mathbf{msgs}_{j,1}, \cdots, \mathbf{msgs}_{j,N}\right)$.

2. Compute $h_{\mathrm{off}} = \mathrm{H}(\mathbf{com}\text{-off}_1, \cdots, \mathbf{com}\text{-off}_M)$ and $h_{\mathrm{on}} = \mathrm{H}(\mathbf{com}\text{-on}_1, \cdots, \mathbf{com}\text{-on}_M)$. Send $h^* = \mathrm{H}(h_{\mathrm{off}}, h_{\mathrm{on}})$ 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 $\left\{p_j^\star\right\}_{j \in \mathcal{C}}$ with $p_j^\star \in [1, N]$.

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

Verification The verifier accepts iff all the following checks succeed:

1. Check the offline phase:
   (a) For every $j \in \mathcal{C}$ and $i \neq p_j^\star$, the verifier uses $\mathbf{state}_{j,i}$ to compute $\mathbf{com}_{j,i}$ as the prover would. Then compute $\mathbf{com}\text{-off}_j = \mathrm{H}\left(\mathrm{com}_{j,1}, \cdots, \mathrm{com}_{j,N}\right)$ using the received value $\mathrm{com}_{j,p_j^\star}$.

   (b) For every $j \in [1, M] \backslash \mathcal{C}$ the verifier uses $\mathbf{seed}_j^*$ to compute $\mathbf{com}\text{-off}_j$ as the prover would.

   (c) The verifier computes $h_{\mathrm{off}} = \mathrm{H}(\mathbf{com}\text{-off}_1, \cdots, \mathbf{com}\text{-off}_M)$.

2. Check the online phase:
   (a) For $j \in \mathcal{C}$ the verifier simulates the online phase using $\{\mathbf{state}_{j,i}\}_{i \neq p_j^\star}$, masked witness $\{\hat{z}_{j,\alpha}\}$, where $\alpha \in \mathrm{IN}$ and $\mathbf{msgs}_{j,i}$ to compute $\{\mathbf{msgs}_{j,i}\}_{i \neq p_j^\star}$. Then compute $\mathbf{com}\text{-on}_j$ as if the prover would do.

   (b) The verifier computes $h_{\mathrm{on}} = \mathrm{H}(\mathbf{com}\text{-on}_1, \cdots, \mathbf{com}\text{-on}_M)$ using the received $\mathbf{com}\text{-on}_j$ for $j \in [1, M] \backslash \mathcal{C}$.

3. The verifier checks that $\mathrm{H}\left(h_{\mathrm{off}}, h_{\mathrm{on}}\right) \overset{?}{=} h^*$.

Figure 2: KKW proof system for a boolean circuit C.

platforms recommended by NIST. We use a Xilinx Kintex-7 FPGA development board[1] which has 298600 lookup-tables (LUTs), 597200 flip-flops (FFs), and 955 block RAMs (BRAMs) available. Moreover, we make all of our code and results publicly available at https://anonymous.4open.science/r/CHES2024-64F9/.

3 Optimizing the Implementation of MPCitH-PP

For typical MPC protocols, communication among parties is not required to compute the linear operations. Hence, there are lightweight nonlinear gates with expensive linear operations for the cryptography primitive in MPC. However, in MPCitH-PP, the linear operations may be too resource-intensive to implement on lightweight FPGA, and determining the length of the critical path can be time-consuming, as the prover needs to simulate computations for N parties. Hence, great attention must also be paid to linear operations in the MPCitH-PP protocol. There are already many techniques [MZ17, MR18, KZ20] to optimize linear operations, including the software implementation of Picnic3. We describe a circuit model for conveniently presenting the essential performance of these techniques at the hardware level.

Circuit model. To facilitate the explanation of the hardware level performance of these optimizations, the underlying circuit is abstracted into a structure where linear and nonlinear layers alternate. For the sake of simplicity, the linear layer and the nonlinear layer in Figure 3 are assumed to consist of multiple XOR and AND gates, respectively, with the linear layer being invertible. We propose a circuit model that not only provides a better explanation of the optimization techniques but also demonstrates that block ciphers with invertible linear operations (e.g., not just LowMC) can benefit from the following optimizations in the KKW protocol within MPCitH-PP.

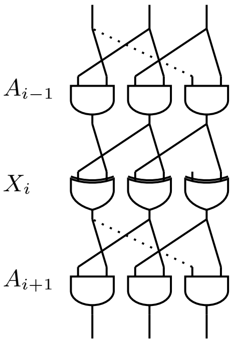

Figure 3: 3 layers of a general circuit.

Optimize the offline phase. Let λ_α and λ_β be two input masks of an AND gate in a non-linear layer, and λ_γ is the output mask. Each party obtains λ_γ by sampling for all AND gates. λ_α and λ_β of layer A_{i+1} are then calculated using the obtained λ_γ of layer A_{i-1} through the linear layer X_i. In the offline phase, the goal of Π_C^{off} is to compute the auxiliary information **aux** for each AND gate, which is computed as $\textbf{aux} = [\lambda_{\alpha\beta}]_N = \lambda_\alpha \cdot \lambda_\beta \oplus [\lambda_{\alpha\beta}]_1 \oplus \cdots \oplus [\lambda_{\alpha\beta}]_{N-1}$.

For instance, the input mask λ_α and λ_β of A_{i+1} are obtained from the output mask λ_γ of A_{i-1} through the linear calculation of X_r, which may be too costly, such as a linear layer of LowMC. If we compute it for each party and sum the results to obtain the input mask of A_{i+1}, there are N computations of linear operations X_i (see Figure 4).

[1] Our test platform is XC7K480T-3. In order to better compare with other post-quantum digital signature FPGA implementations, we also provide test results of Artix-7 and smaller Kintex-7 in Section 5.

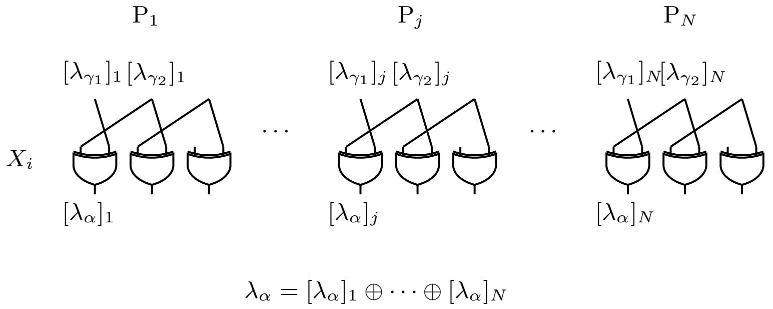
P$_1$
P$_j$
P$_N$
$[\lambda_{\gamma_1}]_1\,[\lambda_{\gamma_2}]_1$
$[\lambda_{\gamma_1}]_j\,[\lambda_{\gamma_2}]_j$
$[\lambda_{\gamma_1}]_N[\lambda_{\gamma_2}]_N$
$\cdots$
$\cdots$
X_i
$[\lambda_\alpha]_1$
$[\lambda_\alpha]_j$
$[\lambda_\alpha]_N$
$\lambda_\alpha = [\lambda_\alpha]_1 \oplus \cdots \oplus [\lambda_\alpha]_N$

Figure 4: Each party uses its own A_{i-1} output share $[\lambda_{\gamma_1}]$ and $[\lambda_{\gamma_2}]$ to calculate X_i to get the input of A_{i+1} share $[\lambda_\alpha]$, and then get the mask λ_α.

It is more efficient to swap the order of summing and computing linear layers. Each party sums the output mask share $[\lambda_\gamma]$ of each AND gate of A_{i-1}, and then computes the linear operation once to obtain the input mask of A_{i+1}. Hence, in Figure 5 there is only to deploy one linear layer for N parties. This optimization not only improves efficiency by avoiding per-party linear computation but also reduces hardware resource consumption.

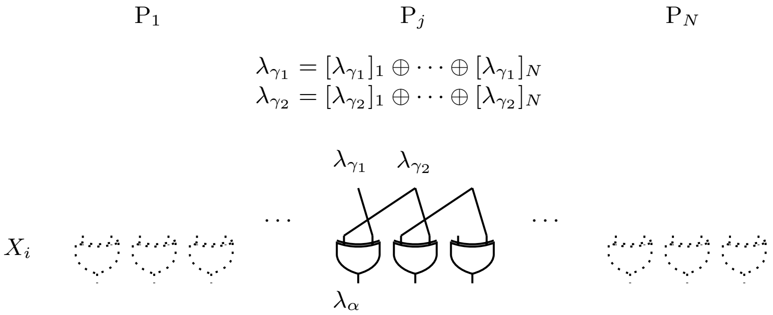
P$_1$
P$_j$
P$_N$
$\lambda_{\gamma_1} = [\lambda_{\gamma_1}]_1 \oplus \cdots \oplus [\lambda_{\gamma_1}]_N$
$\lambda_{\gamma_2} = [\lambda_{\gamma_2}]_1 \oplus \cdots \oplus [\lambda_{\gamma_2}]_N$
λ_{γ_1}
λ_{γ_2}
$\cdots$
$\cdots$
X_i
λ_α

Figure 5: Each party sum A_{i-1} output share $[\lambda_{\gamma_1}]$ and $[\lambda_{\gamma_2}]$ to get λ_{γ_1} and λ_{γ_2}. Then λ_{γ_1} and λ_{γ_2} calculate X_i to get the input of A_{i+1} mask λ_α.

Optimize the online phase. We use the optimized mask sampling method given by Kales and Zaverucha [KZ20] to avoid the linear operation computation for each party. In the online phase, for each AND gate, each party needs to compute the broadcast message **msgs**, which is $[s] = \hat{z}_\alpha\,[\lambda_\beta] \oplus \hat{z}_\beta\,[\lambda_\alpha] \oplus [\lambda_{\alpha,\beta}] \oplus [\lambda_\gamma]$. However, the input shares $[\lambda_\alpha]$ and $[\lambda_\beta]$ of A_{i+1} have to be calculated from the output share $[\lambda_\gamma]$ of A_{i-1}, so this cannot be directly optimized like the offline phase.

Kales and Zaverucha changed the sampling position of shares. That is, sampling is performed before AND gate calculation, rather than after AND gate. The input shares $[\lambda_\alpha]$ and $[\lambda_\beta]$ are sampled from the random tape. This means that each party does not need to calculate the linear calculation of $[\lambda_\alpha]$ and $[\lambda_\beta]$, but only needs to sample directly from the random tape when using it. Therefore, only $\hat{z}_\gamma$ is required to calculate the $\hat{z}_\alpha$ and $\hat{z}_\beta$ of the latter AND gate, and at the hardware level, only 1 hardware usage is required instead of N. After modifying the position, λ_γ needs to be obtained by an invertible linear calculation of the input mask before the calculation of the latter AND gate. In this way, however, each party still needs to calculate $[\lambda_\gamma]$, so in the case of using optimizations in the offline phase, there are some modifications in the protocol, that is, **aux** $= [\lambda_{\alpha,\beta}]_N = \lambda_\alpha \cdot \lambda_\beta \oplus [\lambda_{\alpha\beta}]_1 \oplus \cdots \oplus [\lambda_{\alpha\beta}]_{N-1} \oplus \lambda_\gamma$, and $[s] = \hat{z}_\alpha\,[\lambda_\beta] \oplus \hat{z}_\beta\,[\lambda_\alpha] \oplus [\lambda_{\alpha,\beta}]$. Because the calculation of **aux** has changed, the offline phase is optimized as shown in Figure 6.

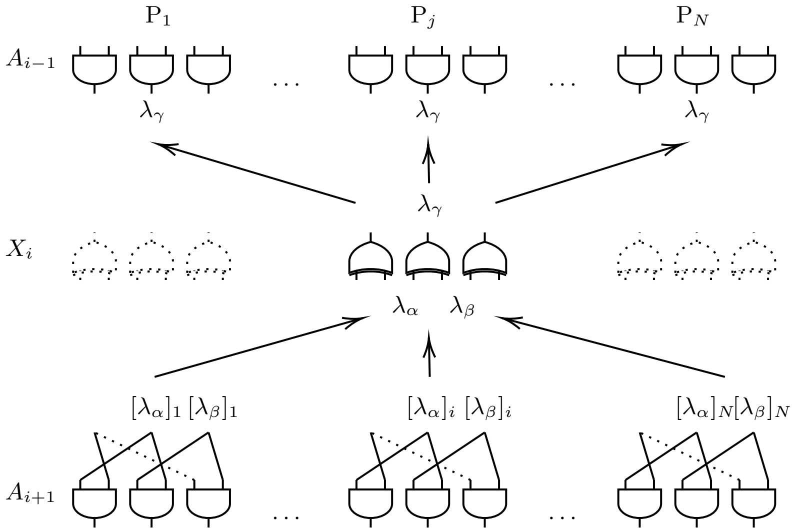
P_1
P_j
P_N
A_{i-1}
λ_γ
λ_γ
λ_γ
λ_γ
X_i
λ_α λ_β
$[\lambda_\alpha]_1$ $[\lambda_\beta]_1$
$[\lambda_\alpha]_i$ $[\lambda_\beta]_i$
$[\lambda_\alpha]_N[\lambda_\beta]_N$
A_{i+1}

Figure 6: Each party samples $[\lambda_\alpha]$ and $[\lambda_\beta]$ of A_{i+1}, get λ_α and λ_β, then compute λ_γ of A_{i-1}.

As discussed above, we review software optimizations and reanalyze the hardware performance of these optimizations. These optimizations reduce the computational complexity of the underlying circuit in MPCitH-PP from $\mathcal{O}(2N \cdot (C_L + C_N))$ to $\mathcal{O}(2N \cdot C_N + 2C_L)$, where C_N represents the number of nonlinear operations in the circuit, C_L represents the number of linear operations and the factor of 2 arises from preprocessing. As a result, nonlinear operations need to be computed N times, while linear operations only need to be computed once. In terms of hardware implementation, the AND gate of the circuit requires an extension to accommodate N parties, whereas the XOR gate only requires "one part".These optimizations have significant implications for hardware resources, particularly in scenarios where the underlying circuit is a symmetric primitive like LowMC, which heavily relies on linear calculations and minimally increases hardware resources with the number of parties. To validate our observation, we implemented the MPCitH-PP protocol with different parties using LowMC as the underlying circuit. Table 3 demonstrates consistent results between the experimental and theoretical observations.

A complete MPCitH-PP protocol based on LowMC (referred to as LowMC-MPC) involves two separate calculations: the offline phase and the online phase, both requiring a normal implementation of LowMC. To optimize this, we merged the two phases by sharing the constant matrix between them. It is important to note that based on the previous optimization, the linear calculation in the offline phase is an inverse operation. Thus, we can only reuse the constant K_i mentioned in Algorithm 1, where $i \in [1, r]$. We analyze LowMC-MPC: to complete the offline phase in r clock cycles, the state first XOR with $K_i \cdot mk$, and then multiplies with L_i^{-1}. This calculation leads to the critical path of LowMC being too long, so we have to divide the calculation into 2 clock cycles to complete one round, and as a result, the offline phase requires $2r$ clock cycles. To address this, we propose an equivalent representation: $L_i^{-1} \cdot (K_i \cdot mk \oplus state) = (L_i^{-1} \cdot K_i) \cdot mk \oplus L_i^{-1} \cdot state$. Essentially, we give a key scheduling matrix $K_i' = L_i^{-1} \cdot K_i$ specifically for the offline phase. This serves as a trade-off between runtime and hardware usage. Notably, this problem does not arise in the online phase.

In Table 3, as the number of parties increases, the growth of hardware usage is slow, which is consistent with our analysis, which the hardware growth comes from AND gates, not XOR gates. Therefore, we implemented LowMC-MPC with 16 parties, answered the

Table 3: Utilization of LowMC-MPC for different parties. N denotes the number of parties. Sec. denotes the security level. Without opt. denotes the LowMC-MPC with $2r + r$ clock cycles, and With opt. denotes the LowMC-MPC with $2r$ clock cycles

Picnic1		**Utilization**							
Sec.	N	**LUTs**	**% LUTs**	**FFs**	**% FFs**				
L1	3	32224	15.81%	3061	0.75%				
L5	3	98319	48.24%	5958	1.46%				
Picnic3		**Without opt.**				**With opt.**			
Sec.	N	**LUTs**	**% LUTs**	**FFs**	**% FFs**	**LUTs**	**% LUTs**	**FFs**	**% FFs**
L1	3	21450	7.18%	2530	0.42%	27186	9.10%	2484	0.42%
	4	22580	7.56%	3076	0.52%	28240	9.46%	3000	0.50%
	8	25924	8.68%	5044	0.84%	32353	10.83%	5035	0.84%
	16	36264	12.14%	9167	1.53%	41378	13.86%	9198	1.54%
L5	3	81388	27.26%	4859	0.81%	104240	34.91%	4867	0.81%
	4	84712	28.37%	5912	0.99%	104859	35.12%	5877	0.98%
	8	97018	32.49%	9991	1.67%	116062	38.87%	9963	1.67%
	16	128668	43.09%	18149	3.04%	148211	49.64%	18116	3.03%

questions we raised in Section 1, and proved that MPCitH-PP with more than 3 parties can be arranged on the FPGA development board. We give the FPGA implementation of Picnic1 [KRR+20] in Table 3. In [KRR+20], an optimization was introduced that reduced the hardware usage of LowMC-MPC from $\mathcal{O}(N \cdot (C_L + C_N))$ to $\mathcal{O}((N-1) \cdot (C_L + C_N))$, where our LowMC-MPC is from $\mathcal{O}(2N \cdot (C_L + C_N))$ to $\mathcal{O}(2C_L + 2N \cdot C_N)$. Therefore, the hardware resources required by [KRR+20]'s LowMC-MPC with 3 parties are larger than those of our 8 parties. It is precisely because of this optimization that we can use pipeline and other optimizations.

4 Optimizing the Implementation of Picnic3

In Section 3, we give a hardware-oriented optimization of MPCitH-PP with more than 3 parties, which makes it possible to implement a digital signature based on MPCitH-PP with high performance. Although we have implemented LowMC-MPC on the FPGA development board to support many parties from 3 to 16, the hardware resources usage of the hash function Keccak in turn begins to restrict the number of parties. It is difficult to implement efficiently for too many parties (e.g., 16 parties). We built a pipeline architecture to implement the Picnic3 with 4 parties to maximize performance as much as possible. Figure 7 illustrates the overall architecture, with streamlined input and output modules. This simplification arises from the utilization of the standardized API design [FFD+18] provided by NIST. Notably, we have chosen to adopt the design presented in reference [Wal19, KRR+20], enhancing the versatility of our implementation. In the following sections, we describe several optimizations proposed for Picnic3, which enable it to fully utilize the resources of Kintex-7 for efficient implementation.

4.1 Multi-stage Pipeline and Parallel Implementation of Picnic3

We give a simple analysis of the implementation performance and resource usage of some key parts of the Picnic3. Picnic3 signature algorithm applied the KKW protocol, seen the Figure 2. In order to reduce the size of the public key, the Merkle tree construction is used to generate the random seeds (the leaf nodes of the Merkel tree), and it is also used in the computation of h_{on} to reduce the signature size. A basic hardware implementation process for $N = 4$ parties with M instances is given in the following:

Step 1. Picnic3 applies Keccak to generate random seeds $\mathbf{seed}_j^*$ for M instances in

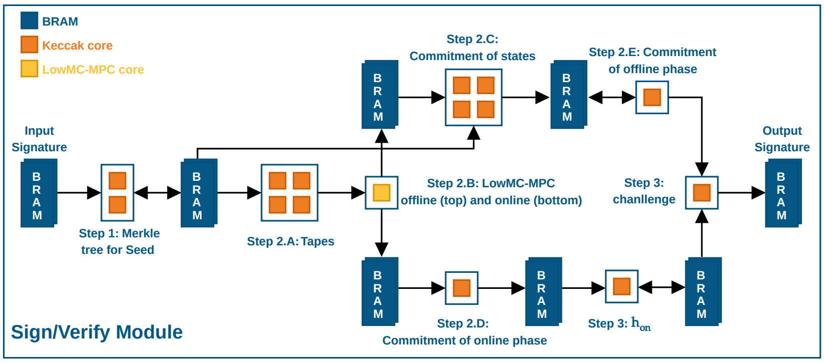
BRAM
Keccak core
LowMC-MPC core
Input Signature
B R A M
Step 1: Merkle tree for Seed
Step 2.A: Tapes
Step 2.B: LowMC-MPC offline (top) and online (bottom)
Step 2.C: Commitment of states
Step 2.E: Commitment of offline phase
Step 3: chanllenge
Step 2.D: Commitment of online phase
Step 3: h_on
Output Signature
Sign/Verify Module

Figure 7: Overall architecture of Picnic3.

Merkle tree mode and then drive N parties' seeds $\mathbf{seed}_{j,i}$ with seed $\mathbf{seed}_j^*$ as the root of Merkle tree. Since there are only 4 parties, jth instance generates $\mathbf{seed}_{j,i}$ simultaneously, where $i \in [1, 4]$. Therefore, two Keccak components are required to parallelize the computation.

Step 2. We give a pipeline implementation of instances description of the offline phase and online phase of MPC. We divide this step into 5 parts, which are independent for easy assembly lines. We describe each part and explain the symmetric primitive components required for each part.

A **Tapes.** It executes the offline phase of MPC Π_C^{off} with Keccak to generate the random tapes required for 4 parties in parallel mode. Here, 4 Keccak components are required, which are also used to drive the random taps in the online phase of MPC Π_C^{off}.

B **LowMC-MPC.** According to the MPCitH-PP protocol specified in Picnic, the block cipher LowMC is used in the offline phase Π_C^{off} to generate the auxiliary information $\mathbf{aux}_j$, and it is also used in the online phase Π_C^{on} to simulate N parties to compute the view $\mathbf{msgs}$ $[s]$. Since online and offline phases are performed separately, we give a circuit calculation (LowMC) to support both phases.

C **Commitment of states.** It adopts Keccak to compute the commitments $\mathbf{com}_{j,i}$= $H(\mathbf{state}_{j,i})$, where $\mathbf{state}_{j,i} = \mathbf{seed}_{j,i}$ $(i = 1, \cdots, N-1)$, $\mathbf{state}_{j,N} = \mathbf{seed}_{j,N} \| \mathbf{aux}_j$. In this part, we need 4 Keccak components to commit to the offline phase for $N = 4$ parties.

D **Commitment of online phase.** After finishing the calculation of the view in step B, a Keccak component is required to commit to the online phase. This is the $\mathbf{com}\text{-on}_j$ in Figure 2.

E **Commitment of offline phase.** Compute $\mathbf{com}\text{-off}_j$ = H($\mathbf{com}_{j,1}$, $\cdots$, $\mathbf{com}_{j,N}$), which needs a components of Keccak.

Step 3. Utilize Keccak to generate h_{on} and h_{off} separately. Subsequently, a Keccak component is needed to hash data, including commitment, public key, salt, and message, and compute the challenge value.

In Step 2, we analyze the executing time of 5 parts, the computation time of LowMC in Step B is less than other parts. Hence, we carry out the LowMC in offline phase and online phase in the serial processing. Figure 8 illustrates the 5-stage pipeline structure of the initial 3 instances. $\mathbf{com}_{i,j}$ (C_1) starts after the calculation of auxiliary information (the first B_1), $\mathbf{com}$-on (D_1) starts after the calculation of the view (the second B_1). $\mathbf{com}$-off

(E_1) starts after the calculation of $\mathbf{com}_{i,j}$ (C_1). The time t refers to the time required for the longest-running part.

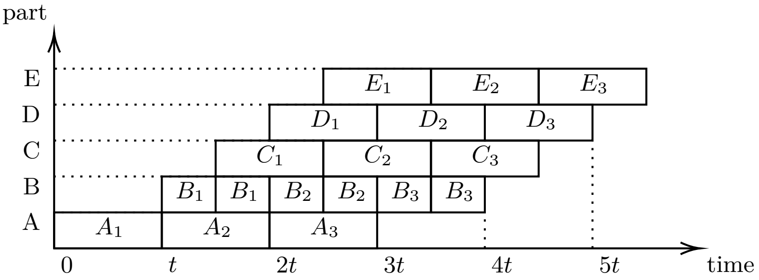

Figure 8: Pipeline of the Step 2 of Picnic3.

Table 4 provides the count of symmetric primitive components utilized in each part, offering a clear representation of the level of parallelism in each step. To maximize speed, the quantity of Keccak instances is directly related to the number of parties. Despite the low hardware utilization of an individual Keccak instance, the substantial number makes it challenging to accommodate them in L5 level Picnic3 with over 4 parties. For the basic pipeline construction, we give more optimation to speed up the implementation of Picnic3, including the optimation of symmetry primitive, extending the pipeline construction with Step 1 and Step 3, and the construction of signature as well.

Table 4: symmetric components used by different parts.

Step	Design Part	Symmetric Primitive	Number
Step 1	Merkel Tree for seeds	Keccak	2
Step 2	Tapes	Keccak	4
	LowMC-MPC	LowMC-MPC	1
	Commitment of states	Keccak	4
	Commitment of online phase	Keccak	1
	Commitment of offline phase	Keccak	1
Step 3	Merkel Tree for h_{on}	Keccak	1
	h_{off}	Keccak	1
	Challenge	Keccak	1

4.2 Optimization of Symmetric Primitives

The value of t in Figure 8 depends on the longest-running design part. In our initial implementation, the offline phase of LowMC-MPC consists of r or $2r$ cycles, while the online phase consists of r clock cycles. Here, r represents the number of rounds in LowMC-MPC, which is 4. Additionally, Keccak has 24 rounds, each requiring a cycle. This is significantly more than LowMC's cycles. According to Table 1, Keccak's absorption or squeezing capacity is limited. Hence, certain parts can be computed with one execution of Keccak, such as the generation of random seed. Conversely, some parts require multiple executions of Keccak. For instance, Tapes in the L5 security level require two executions to generate the $255 \times 2 \times r = 2040$-bit random number, Commitment of online phase $\mathbf{com}\text{-on}_j$ in L5 security level requires four executions to hash the views with $255 + 255 \times r \times N = 4335$-bit, and in L1 security level, two executions are necessary. Therefore, we conduct an analysis of symmetric primitives as our initial step. Compared to the round function of Keccak, LowMC requires a significant amount of computation and has a longer critical path. Modifying the clock cycles of Keccak allows for consistency across all pipeline modules. For example, we reduce the clock cycles of Keccak to

12 clock cycles during the Commitment of the online phase and perform the execution twice, resulting in total clock cycles equivalent to that of Commitment of state. Although hardware utilization has increased, pipeline performance has been effectively optimized.

Table 5 present the critical paths and hardware utilization of the KECCAK and LowMC on the Kintex-7 platform. Moreover, as the critical path decreases, the hardware utilization gradually increases, which puts a limit on the efficient implementation of PICNIC3 on the FPGA platform.

Table 5: Hardware utilization and critical path of symmetric primitives. LowMC-MPC-o denotes optimization of LowMC-MPC in Subsection 4.1.

Design Part	LUTs	%	FFs	%	Clock Cycles	Critical Path
KECCAK-1	3467	1.16%	1606	0.27%	24	2.392 ns
KECCAK-2	6964	2.33%	1624	0.27%	12	3.956 ns
KECCAK-3	11041	3.70%	1619	0.27%	8	4.897 ns
KECCAK-4	13638	4.57%	1617	0.27%	6	6.475 ns
KECCAK-6	20525	6.87%	1618	0.27%	4	9.018 ns
LowMC-MPC-L1	22580	7.56%	3076	0.52%	12	5.021 ns
LowMC-MPC-o-L1	28240	9.46%	3000	0.50%	8	5.422 ns
LowMC-MPC-L5	84712	28.37%	5912	0.99%	12	5.889 ns

In Section 3, we discuss the trade-off between hardware utilization and runtime for LowMC-MPC, which is also observed in the implementation of KECCAK. Furthermore, besides the mentioned pipeline optimization in Figure 8, Step 1 and Step 3 can also be optimized in a similar manner. Additionally, as the number of cycles in LowMC-MPC is fewer than that of KECCAK, the pipeline's implementation of KECCAK can also be reduced to match the cycle count of LowMC-MPC. However, achieving this requires a carefully chosen trade-off, and we provide optimized versions of PICNIC-L1 in Section 5.

4.3 Extension of Multi-stage Pipeline

In Subsection 4.1, the pipeline construction only includes the operations in Step 2. We can extend the pipeline construction to be compatible with the computation of Step 1 and Step 3.

In Step 1 of Subsection 4.1, two independent KECCAK components are used to drive all the seeds, which are stored in BRAM. We first generate M instance seeds in Merkle tree mode and then generate N parties' seeds for all instances in the Merkel tree as well, which is regarded as a big Merkel tree. We first generate all non-leaf nodes in the Merkle tree and store them in BRAM, which is named Step $1'$. For every instance, the second-to-last level of the Merkle tree is obtained and used to produce the 4 parties' seeds with one execution of KECCAK by 2 KECCAK components. The parties' seed generation of an instance is added to the beginning of the pipeline of Step 2 in Figure 8.

In Step 3, the generation of the challenge requires the calculation of h_{off} and h_{on}. Because the computation of h_{off} and h_{on} use two independent KECCAK components. We propose incorporating the calculation of h_{off} and h_{on} from Step 3 into the pipeline of Step 2 to reduce the running time. For h_{off}, we start the calculation of h_{off} after the calculation of **com**-off$_j$ in pipeline, so there is no extra h_{off} computation time in Step 3. As for h_{on}, since we have already computed the leaf nodes of the Merkel tree in Step 2, we can directly calculate the parent nodes of the Merkel tree leaf nodes during the pipeline. Based on the structure of the binary tree, this optimization can approximately halve the computation time required for h_{on}. This optimization significantly reduces the time required for generating challenges. It can be seen that the time complexity of h_{off} is $\mathcal{O}(M)$ in Step 3 of the pipeline (see Subsection 4.1), where M represents the number of instances. Once this step is incorporated into the pipeline of Step 2, the extra $\mathcal{O}(M)$ computation time will be saved. We use Step $3'$ to represent these optimized parts.

4.4 Optimization of Constructing Signature

We present an optimization for challenge generation. Reviewing the KKW protocol Figure 2, we analyze the computation of h_{off} in more detail. By the computation in step (b) in **Commit**, we know a commitment $\mathbf{com}\text{-off}_j = H(\mathbf{com}_{j,0}, \ldots, \mathbf{com}_{j,i}, \ldots, \mathbf{com}_{j,N})$. Since the commitment h_{off} is generated for all instances of $\mathbf{com}\text{-off}_j$, i.e.,

$$h_{\text{off}} = H(\mathbf{com}\text{-off}_0, \ldots, \mathbf{com}\text{-off}_j, \ldots, \mathbf{com}\text{-off}_N).$$

In the meantime, it has been observed that $\mathbf{com}\text{-off}_j$ are not sent to the verifier and are instead calculated by the verifier. Hence, we propose an improved computation to compute the h_{off} directly using $\mathbf{com}_{j,i}$ as following,

$$h_{\text{off}} = H(\mathbf{com}_{0,0}, \ldots, \mathbf{com}_{0,N}, \ldots, \mathbf{com}_{j,0}, \ldots, \mathbf{com}_{j,N}, \ldots, \mathbf{com}_{M,0}, \ldots, \mathbf{com}_{M,N}). \quad (1)$$

Lemma 1. *For the KKW protocol, the computation of commitment h_{off} is instead with Equation 1, which has the same security as the original computation in the KKW protocol in Figure 2.*

Here, we use the optimization implementation that can reduce $\mathcal{O}(M)$ executions of the hash function Keccak, which is applicable to reduce the running time for software implementation and hardware implementation.

5 Implementation

In this section, we present an FPGA implementation of Picnic3 on the Kintex-7 by combining the optimizations discussed in Section 3 and Section 4. We first list the specific components used by each module of Picnic3, and then we give the analysis of their hardware usage and clock cycles.

5.1 Implementation of Picnic3

Initially, we develop a basic implementation of Picnic3 for security levels L1 and L5, and subsequently design and realize two optimized versions. The primary focus of these optimized versions lies in accelerating the operational speed of Keccak, utilizing additional hardware resources to significantly enhance the computational performance of Picnic3. Within this process, a delicate balance must be struck between boosting hardware operation speed and optimizing the utilization of limited hardware resources, thereby guiding the selection of the most appropriate optimization strategy.

For the L1 security level, we devise and implement three different hardware versions of Picnic3. Among these, version 1 employs a Keccak operational cycle of 24 clock cycles; version 2 reduces this to a Keccak operational cycle of 12 clock cycles; and for version 3, we further reduce the Keccak operation cycle to a mere 8 clock cycles. Due to the Keccak used in version 3 having a cycle count of 8, we use the 8-cycle LowMC-MPC-o to fully maximize the performance of the pipeline. Additionally, it is important to note that both version 2 and version 3 utilize two Keccak in the commitment of the online phase. This is because the use of K-4 and K-6 results in longer critical paths (see Table 5), consequently reducing the performance of the entire hardware implementation.

For the L5 security level, we first implemented one hardware version of Picnic3, namely version 1, which also incorporates a Keccak operational cycle of 24 clock cycles.

In Table 6, Tapes we use 3 K-2 and one K-1, because the N-th participant only needs to generate λ_{α} instead of $\lambda_{\alpha,\beta}$. Commitment of state requires 3 K-1 and one K-2, because the Nth party needs to make a commitment to the additional $\lambda_{\alpha,\beta}$. Commitment of online phase should use K-4, but since the critical path of K-4 is longer than LowMC-MPC (see Table 5), thus we use two K-2. The L5 security level is difficult to design versions 2 and 3 like L1, because the critical path of Keccak with a clock cycle number less than 8 is longer than that of LowMC-MPC. In Table 8, Step $1'$ has more clock cycles than Step 2, so we set version 2, using K-2 in Merkel Tree for seeds & Seeds (see Table 6).

Table 6: Symmetric primitive components in Picnic3. K-1 denotes Keccak-1, K-2 denotes Keccak-2, and K-3 denotes Keccak-3. LowMC-MPC-o denotes optimization of LowMC-MPC.

	Design Part	**Symmetric Primitive**		
		Version 1	**Version 2**	**Version 3**
L1	Merkel Tree for seeds & Seeds	2 × K-1	2 × K-2	2 × K-3
	Tapes	4 × K-1	4 × K-2	4 × K-3
	LowMC-MPC	LowMC-MPC	LowMC-MPC	LowMC-MPC-o
	Commitment of state	4 × K-1	4 × K-2	4 × K-3
	Commitment of online phase	K-2	2 × K-2	2 × K-3
	Commitment of offline phase	K-1	K-2	K-3
	Merkel Tree for h_{on}	K-1	K-2	K-3
	h_{off}	K-1	K-2	K-3
	Challenge	K-1	K-1	K-1
L5	Merkel Tree for seeds & Seeds	2 × K-1	2 × K-2	-
	Tapes	3 × K-2 + K-1	3 × K-2 + K-1	-
	LowMC-MPC	LowMC-MPC	LowMC-MPC	-
	Commitment of state	3 × K-1 + K-2	3 × K-1 + K-2	-
	Commitment of online phase	2 × K-2	2 × K-2	-
	Commitment of offline phase	K-2	K-2	-
	Merkel Tree for h_{on}	K-2	K-2	-
	h_{off}	K-2	K-2	-
	Challenge	K-1	K-1	-

Hardware Utilization. Table 7 displays the resource utilization of our implemented Picnic3. Version 2 and version 3 of Picnic3-L1 require 1.67 times and 2.41 times more resources compared to version 1, respectively. The resource occupancy rate of version 1 of Picnic3-L5 is already approaching 70%. Consequently, it is challenging to optimize the L5 security level by reducing the Keccak cycle. We emphasize that Picnic3-L1 of version 1 can be deployed on Artix. Artix's hardware volume is 1344600 LUTs and 269200 FFs.

Clock Cycle. Table 8 presents the number of clock cycles necessary for each part within our Picnic3 implementation. Except for LowMC-MPC, the clock cycles of the other parts depend on the Keccak function it invokes. Our implementation of Keccak requires 24 clock cycles, while the optimized L1 version takes 8 cycles. The Pipeline in the table refers to the time required for the pipeline to calculate an instance. In Table 4, Step $1'$ is the construction of the Merkel Tree of the seed, Step 2 is the time for the pipeline of all instances, and Step $3'$ is the total cycle of calculating h_{off}, h_{on} and challenge generation. It is evident that Step 2 in Table 8 exceeds $24 \times M$. This is due to the additional control cycles required by Keccak, as well as the need for additional cycles to complete the data storage. It is noted that our cycle analysis does not account for data transmission time. For signing batches of messages, we could build a pipeline with Step $1'$ to $3'$, in which the clock cycle of the throughput is the clock cycle of Step 2. However, as the Keccaks of Step $1'$ and Step $3'$ also run during Step 2, the addition of extra Keccaks becomes necessary to fulfill this objective. Additionally, in order to sign batch messages, extra

Table 7: Implementation of each version of Picnic3 under different security levels.

Scheme	**Version**	**LUTs**	**%**	**FFs**	**%**	**BRAMs**	**%**
			XC7K480T				
Picnic3-L1-Sign	1	82789	27.74%	38554	6.46%	65.5	6.86%
Picnic3-L1-Sign	2	134964	45.18%	42556	7.13%	65.5	6.86%
Picnic3-L1-Sign	3	183012	61.29%	42783	7.17%	65.5	6.86%
Picnic3-L1-Verify	1	97199	32.53%	40369	6.76%	65.5	6.86%
Picnic3-L5-Sign	1	190469	63.79%	65551	11.32%	159.5	16.70%
Picnic3-L5-Sign	2	198186	66.37%	65594	10.98%	159.5	16.70%
Picnic3-L5-Verify	1	201441	67.48%	59348	9.93%	159.5	16.70%
			XC7K325T				
Picnic3-L1-Sign	3	183012	89.80%	42783	10.50%	65.5	14.72%
Picnic3-L5-Sign	2	198186	97.25%	65594	16.09%	159.5	35.84%
			XC7A200T				
Picnic3-L1-Sign	1	82826	61.53%	38554	14.82%	65.5	17.95%
Picnic3-L1-Sign	2	135001	100.30%	42556	16.36%	65.5	17.95%

BRAM is required, as the data stored in BRAM, such as the seed, needs to be signed after the challenge selection is completed.

Table 8: Clock cycles analysis of Picnic3.

Design part	Picnic3-L1 **Version 1**	Picnic3-L1 **Version 2**	Picnic3-L1 **Version 3**	Picnic3-L5 **Version 1**	Picnic3-L5 **Version 2**
LowMC	12	12	8	12	12
Pipeline	24	12	8	24	24
Step $1'$	6293	3497	2798	12418	6898
Step 2	5664	3370	2836	10943	10557
Step $3'$	3872	2556	2649	7363	7337

Picnic3 utilizes two Merkle Trees. In the verification process, generating the Seeds from the root node or constructing h_{off} from the leaf nodes is unnecessary. Consequently, the verification requires less time compared to the signing process. Given the non-deterministic tree construction and associated challenges, only the signature timing is provided here. Table 9 illustrates the running time of Picnic3 for signing on software and hardware. The table showcases the running time comparison between software and FPGA implementations for different versions of Picnic3 at various security levels.

Higher versions of Picnic3 generally involve fewer clock cycles. The AVX2 software implementation has a running time of 5.17 ms for Picnic3-L1. On the other hand, the FPGA implementation achieves a remarkable speedup, completing the task in only 0.079 ms. This corresponds to a significant performance increase, with version 1 being nearly 65 times faster, version 2 being nearly 110 times faster and version 3 being 112 times faster. The time difference between version 2 and version 3 is very small, but the hardware of version 2 has obvious advantages. Version 3 has less optimization because the storage data and the control logic of LowMC-MPC cannot reach 8 cycles at this time. We present three scheme versions with varying speed and hardware requirements. This trade-off in implementation offers flexibility for different purposes. As another example, for the Picnic3-L5 scheme, the software implementation takes 18.15 ms, while the FPGA implementation completes the task in 0.181 and 0.146 ms, making it approximately 100 and 124 times faster. We compare the performance of the software implementation in Table 9, which utilizes AVX2 instruction set acceleration. We observed a significant improvement in the performance of our implemented Picnic3 with 4 parties compared to Picnic2. However, it is important to note that implementing 4 parties in Picnic3 results in an increase in the signature size (see Table 2).

These results underscore the advantages of FPGA over software implementation, particularly the significantly improved execution time and performance. FPGA provides a highly parallelized hardware architecture, enabling efficient execution of complex algorithms with reduced latency.

Table 9: Running time of software (used AVX2 from [KZ20]) and FPGA. This table shows the best performing implementation, the performance of other FPGAs is in Table 10.

Scheme	N	**Version**	**Clock Cycle**	**Frequency** **ns**	**Platform**	**Sign time** **ms**
Picnic3-L1	16	-	-	-	AVX2	5.17
Picnic3-L1	4	1	15829	5.030	XC7K480T-3	0.079
Picnic3-L1	4	2	9423	5.019	XC7K480T-3	0.047
Picnic3-L1	4	3	8283	5.580	XC7K480T-3	0.046
Picnic3-L5	16	-	-	-	AVX2	18.15
Picnic3-L5	4	1	30724	5.893	XC7K480T-3	0.181
Picnic3-L5	4	2	24792	5.893	XC7K480T-3	0.146

5.2 Comparison to FPGA Implementations of Other Schemes

This section compares our work with other existing FPGA implementations of signature schemes, specifically focusing on the Sphincs and Picnic1 schemes. It is important to note that Sphincs is based on hash functions, while Picnic1 is based on block ciphers. Both of these schemes rely on symmetric primitives. In contrast, our work aligns with the security assumptions of symmetric primitives and is considerably faster than other schemes.

Table 10 provides a comparison of our work with these existing implementations. The table includes various parameters such as maximum frequency, number of LUTs, FFs, DSPs, BRAMs, cycles, and execution time. For a more complete comparison, we provide area-time (AT) product, and we also provide results of Artix-7 and different Kintex-7. Our implementation excels, especially at the L1 security level, demonstrating superior performance. It not only achieves fast signing speed but also boasts the smallest AT. Furthermore, our implementation surpasses Picnic1 and Sphincs in terms of speed at both security levels, with smaller AT values. Specifically, at the L1 level, our implementation is 21 times faster than Sphincs and 5 times faster than Picnic1. At the L5 level, it is 17 times faster than Sphincs and 8 times faster than Picnic1.

6 Conclusion and Future Work

In this paper, we study the FPGA implementation of the MPCitH-PP protocol, along with its related signature schemes. Our findings indicate that it is indeed viable to deploy the MPCitH-PP protocol, involving more than 3 parties (e.g., LowMC-MPC for 16 parties), on resource-constrained FPGAs. Expanding on this discovery, we have developed a 4-party implementation of the Picnic3 digital signature scheme, utilizing various hardware optimizations. As a result, the FPGA implementation of Picnic3 offers significantly improved performance compared to previous works.

In Section 3, we discussed the performance of MPCitH-PP on Boolean circuits with some optimization to reduce the hardware area of the block cipher by $\mathcal{O}(2N \cdot C_N + 2C_L)$ unit. The description of Section 3 is based on a general circuit, not LowMC. This optimization is suitable for the block cipher with invertible linear operation in the KKW protocol. However, the optimization for the hardware resources of the linear layer, which

Table 10: Comparison to FPGA implementations of other signature schemes (modified from [BNG22]). AT denotes area-time product, where AT = (DSP ×100 + BRAM ×196 + LUT /4) × Sign(s)[NKeSK+22, Xil16].

Design	**Algorithm**	**Max Freq.**	**LUTs**	**FFs**	**DSPs**	**BRAMs**	**Sign**		**AT**	**Family**
		(MHz)					**cycles**	μs		
Security Level 1										
[BNG22]	FALCON-512	142	14500	7287	4	2	-	-	-	A7
[ALCZ20]	SPHINCS$^+$-128s-simple	250 & 500	48231	72514	0	11.5	-	12400	177.5	A7
[ALCZ20]	SPHINCS$^+$-128s-robust	250 & 500	49146	73069	0	15.5	-	21100	323.3	A7
[ALCZ20]	SPHINCS$^+$-128f-simple	250 & 500	47991	72505	1	11.5	-	1010	14.5	A7
[ALCZ20]	SPHINCS$^+$-128f-robust	250 & 500	48930	72505	0	15.5	-	1640	25.0	A7
[KRR+20]	PICNIC1-L1-FS	125	90535	23516	0	52.5	31300	250	8.2	K7
[BNG22]	FALCON-512	314	14327	7314	4	2	-	-	-	VUS+
this paper	PICNIC3-L1	199	82789	38554	0	65.5	15829	**80**	2.7	XC7K480T-3
this paper	PICNIC3-L1	199	134964	42556	0	65.5	9423	**47**	2.2	XC7K480T-3
this paper	PICNIC3-L1	179	183012	42783	0	65.5	8283	**46**	2.7	XC7K480T-3
this paper	PICNIC3-L1	178/168	82789	38554	0	65.5	15829	**88/93**	3.0/3.1	XC7K325T-3/2
this paper	PICNIC3-L1	178/165	134964	42556	0	65.5	9423	**52/57**	2.4/2.7	XC7K325T-3/2
this paper	PICNIC3-L1	153/144	183012	42783	0	65.5	8283	**53/57**	3.2/3.3	XC7K325T-3/2
this paper	PICNIC3-L1	136	82826	37741	0	65.5	30724	**117**	3.9	XC7A200T-3
Security Level 5										
[BNG22]	FALCON-1024	142	13956	6737	1	2	-	-	-	A7
[BNG22]	Dilithium-V	116	53187	28318	16	29	24358/55070	210/475	4.3/9.8	A7
[ALCZ20]	SPHINCS$^+$-256s-simple	250 & 500	51130	74576	1	22.5	-	19,300	333.7	A7
[ALCZ20]	SPHINCS$^+$-256s-robust	250 & 500	5000	75738	1	30	-	36,100	261.0	A7
[ALCZ20]	SPHINCS$^+$-256f-simple	250 & 500	51009	74539	1	22.5	-	2,520	43.5	A7
[ALCZ20]	SPHINCS$^+$-256f-robust	250 & 500	50341	75664	1	30	-	4,680	86.9	A7
[LSG21]	Dilithium-V	140	44653	13814	45	31	70376/145912	503/1042	10.9/22.7	A7
[BNG22]	Dilithium-V	173	54468	28639	16	29	24358/55070	141/318	2.9/6.6	K7
[KRR+20]	PICNIC1-L5-FS	125	167530	33164	0	99	154500	1236	75.8	K7
[BNG22]	FALCON-1024	314	13729	6771	4	2	-	-	-	VUS+
[BNG22]	Dilithium-V	256	53907	28435	16	29	24358/55070	95/215	2.0/4.5	VUS+
[AMJ+21]	Dilithium-V	200	19100	9300	4	24	68500/-	342/-	3.4/-	ZUS+
this paper	PICNIC3-L5	170	190469	65551	0	159.5	30724	**181**	14.3	XC7K480T-3
this paper	PICNIC3-L5	170	198186	65594	0	159.5	24792	**146**	11.8	XC7K480T-3
this paper	PICNIC3-L5	157/147	190469	65551	0	159.5	30724	**196/209**	15.5/16.5	XC7K325T-3/2
this paper	PICNIC3-L5	157/147	198186	65594	0	159.5	24792	**158/168**	12.8/13.6	XC7K325T-3/2

depends on specific proof strategies[2], is more subtle and cannot be applied directly to other MPCitH proofs.

The signature [KZ20, dSGDMOS20, BdSGK^{+}21, MGH^{+}23, AMGH^{+}23, KHS^{+}22] structures based on MPCitH are similar, where the corresponding seeds are first generated through Merkel tree, and then random tapes are generated using the seeds. Moreover, commitments need to be made for the views of MPCitH. The optimizations constructed in Section 4 and Section 5 are fully applicable to these structures, so our implementation has some inspiration for this type of algorithm.

Although Picnic did not eventually become a standard, our work provides the possibility for FPGA implementation of the more parties MPCitH protocol. In the campaign for the new additional round at NIST [Nat23], there are 40 submitted signatures, 9 of which are based on MPCitH (or its variants) [AMGH^{+}23, KHS^{+}22, BBdSG^{+}23, BFR23, ABB^{+}23, BKPV23]. In order to further promote this research, future work may focus on the implementation of the most advanced MPCitH protocols at present. This paper primarily focuses on the high-performance implementation of the MPCitH protocol. Therefore, it does not address the side-channel protection [SBWE20, ABE^{+}21] of the protocol. However, it is important to note that future work involves implementing a side-channel FPGA implementation for the MPCitH protocol.

References

[AASA^{+}19] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David A. Cooper, Quynh Dang, Yi-Kai Liu, Carl A. Miller, Dustin Moody, René Peralta, Ray A. Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the first round of the NIST post-quantum cryptography standardization process. 2019.

[ABB^{+}23] 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 Cryptol. ePrint Arch.*, page 1666, 2023.

[ABE^{+}21] Diego F. Aranha, Sebastian Berndt, Thomas Eisenbarth, Okan Seker, Akira Takahashi, Luca Wilke, and Greg Zaverucha. Side-Channel Protections for Picnic Signatures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):239–282, 2021.

[ALCZ20] Dorian Amiet, Lukas Leuenberger, Andreas Curiger, and Paul Zbinden. Fpga-based sphincs+ implementations: Mind the glitch. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 229–237, 2020.

[AMGH^{+}23] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The Return of the SDitH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 564–596, Cham, 2023. Springer Nature Switzerland.

[AMJ^{+}21] Aikata Aikata, Ahmet Can Mert, David Jacquemin, Amitabh Das, Donald Matthews, Santosh Ghosh, and Sujoy Sinha Roy. A unified cryptoprocessor for lattice-based signature and key-exchange. Cryptology ePrint Archive, Paper 2021/1461, 2021. https://eprint.iacr.org/2021/1461.

[2] Currently, some MPCitH-PP is no longer based on the KKW protocol, but on the BN++ protocol [MGH^{+}23, KHS^{+}22, AMGH^{+}23].

[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*, pages 430–454, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[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.

[BdSGK+21] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from aes. In Juan A. Garay, editor, *Public-Key Cryptography – PKC 2021*, pages 266–297, Cham, 2021. Springer International Publishing.

[BFR23] Ryad Benadjila, Thibauld Feneuil, and Matthieu Rivain. Mq on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. Cryptology ePrint Archive, Paper 2023/1719, 2023. https://eprint.iacr.org/2023/1719.

[BKPV23] Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier Verbel. Biscuit: New mpcith signature scheme from structured multivariate polynomials. Cryptology ePrint Archive, Paper 2023/1760, 2023. https://eprint.iacr.org/2023/1760.

[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*, pages 495–526, Cham, 2020. Springer International Publishing.

[BNG22] Luke Beckwith, Duc Tri Nguyen, and Kris Gaj. High-Performance Hardware Implementation of Lattice-Based Digital Signatures. *IACR Cryptol. ePrint Arch.*, page 217, 2022.

[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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 18251842, New York, NY, USA, 2017. Association for Computing Machinery.

[CDG+20] Melissa Chase, David Derler, Steven Goldfeder, Daniel Kales, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. Picnic: A Family of Post-Quantum Secure Digital Signature Algorithms, 2020. https://microsoft.github.io/Picnic/.

[Din21a] Itai Dinur. Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 374–403. Springer, 2021.

[Din21b] Itai Dinur. Improved Algorithms for Solving Polynomial Systems over GF(2) by Multiple Parity-Counting. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2550–2564. SIAM, 2021.

[dSGDMOS20] 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*, pages 669–692, Cham, 2020. Springer International Publishing.

[FFD+18] Ahmed Ferozpuri, Farnoud Farahmand, Viet Ba Dang, M. Sharif, Jens-Peter Kaps, and Kris Gaj. Hardware API for Post-Quantum Public Key Cryptosystems. 2018.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings on Advances in Cryptology—CRYPTO '86*, page 186194, Berlin, Heidelberg, 1987. Springer-Verlag.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster Zero-Knowledge for boolean circuits. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1069–1083, Austin, TX, August 2016. USENIX Association.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212219, New York, NY, USA, 1996. Association for Computing Machinery.

[IKOS07] 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*, STOC '07, page 2130, New York, NY, USA, 2007. Association for Computing Machinery.

[KHS+22] Seongkwang Kim, Jincheol Ha, Mincheol Son, ByeongHak Lee, Dukjae Moon, Joohee Lee, Sangyup Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. AIM: Symmetric Primitive for Shorter Signatures with Stronger Security. *IACR Cryptol. ePrint Arch.*, page 1387, 2022.

[KKW18] 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*, CCS '18, page 525537, New York, NY, USA, 2018. Association for Computing Machinery.

[KRR+20] Daniel Kales, Sebastian Ramacher, Christian Rechberger, Roman Walch, and Mario Werner. Efficient FPGA Implementations of LowMC and Picnic. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 417–441, Cham, 2020. Springer International Publishing.

[KZ20] Daniel Kales and Greg Zaverucha. Improving the Performance of the Picnic Signature Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):154188, Aug. 2020.

[LMSI22] Fukang Liu, Willi Meier, Santanu Sarkar, and Takanori Isobe. New low-memory algebraic attacks on lowmc in the picnic setting. *IACR Trans. Symmetric Cryptol.*, 2022(3):102–122, 2022.

[LSG21] Georg Land, Pascal Sasdrich, and Tim Güneysu. A hard crystal - implementing dilithium on reconfigurable hardware. Cryptology ePrint Archive, Paper 2021/355, 2021. https://eprint.iacr.org/2021/355.

[MGH$^+$23] Carlos Aguilar Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the sdith. 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 564–596. Springer, 2023.

[MR18] Payman Mohassel and Peter Rindal. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 3552, New York, NY, USA, 2018. Association for Computing Machinery.

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.

[Nat20] National Institute of Standards and Technology. Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process, 2020. https://doi.org/10.6028/NIST.IR.8309.

[Nat23] National Institute of Standards and Technology. Post-Quantum Cryptography: Digital Signature Schemes, 2023. https://csrc.nist.gov/projects/pqc-dig-sig.

[NKeSK$^+$22] Ziying Ni, Ayesha Khalid, Dur e Shahwar Kundi, Máire ONeill, and Weiqiang Liu. HPKA: A High-Performance CRYSTALS-Kyber Accelerator Exploring Efficient Pipelining. Cryptology ePrint Archive, Paper 2022/1093, 2022. https://eprint.iacr.org/2022/1093.

[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.

[SBWE20] Okan Seker, Sebastian Berndt, Luca Wilke, and Thomas Eisenbarth. Sni-in-the-head: Protecting mpc-in-the-head protocols against side-channel analysis. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1033–1049. ACM, 2020.

[SCW23] Yimeng Sun, Jiamin Cui, and Meiqin Wang. Improved attacks on lowmc with algebraic techniques. Cryptology ePrint Archive, Paper 2023/1718, 2023. https://eprint.iacr.org/2023/1718.

[Sho94] P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, page 124134, USA, 1994. IEEE Computer Society.

[Wal19] Roman Walch. Design and Implementation of a Picnic Coprocessor. Master's thesis, Graz University of Technology, 2019.

[Xil16] Xilinx. 7 series FPGAs configurable logic block: User guide, 2016.

[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.