A 28nm 8.3mm2 76.9μJ/Pairing Scheme-Adaptive Crypto-Processor for Hybrid ZKP-PBC Application with Software-Hardware Co-Design

Journal:	*IEEE Journal of Solid State Circuits*
Manuscript ID	JSSC-25-0638
Manuscript Type:	Regular Manuscript
Date Submitted by the Author:	27-Oct-2025
Complete List of Authors:	Chen, Xiangren; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University Yang, Bohan; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University Zhu, Wenping; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University Zhang, Jinfan; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University Jiang, Yifan; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University Wang, Hanning; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University Zhu, Min; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University Zhang, Aoyang; Wuxi Micro Innovation Integrated Circuit Design Co Ltd, Wuxi Micro Innovation Integrated Circuit Design Co., Ltd. Liu, Leibo; Beijing National Research Center for Information Science and Technology (BNRist) School of Integrated Circuits Tsinghua University
Keywords:	<b>ZKP</b>, <b>zkSNARK</b>, <b>PBC</b>, <b>Crypto-processor</b>, Software-Hardware Co-design

A 28nm 8.3mm^2 76.9μJ/Pairing Scheme-Adaptive Crypto-Processor for Hybrid ZKP-PBC Application with Software-Hardware Co-Design

***Abstract*—Zero-knowledge Proof (ZKP), one of the most popular privacy-preserving schemes, enables the prover to convince the verifier of a certain statement's correctness without leaking any private information. Among them, the Pairing-based zkSNARK (Succinct Non-interactive ARgument of Knowledge) like Groth16 [1] and Plonk [2], featuring the succinct proof size and constant-time verification, is deployed across promising ecosystem. However, the heavy operations of proof generation that grow rapidly with application sizes and the frequent verification that requires costly Pairing [3] severely hinder broader adoption. Moreover, there are many other cryptographic schemes also constructed based on Pairing, such as BLS signature, functional encryption (FE), identity-based encryption (IBE) and so on. Unfortunately, existing synthesis-based ZKP accelerators suffer from low area efficiency and incomplete operator coverage, causing poor overall speedup and expensive deployment cost.**

This work presents the first silicon-proven crypto-processor that supports complete proof generation/verification phases and can adapt to other Pairing-based cryptography (PBC). To achieve high flexibility while maintaining cost efficiency, we employ holistic optimizations across algorithm, architecture and compiler levels, including the hybrid-grained instruction set, dedicated memory system, reconfigurable datapath, custom Pairing compiler and utilization-oriented timing scheduling. Fabricated in 28nm CMOS, it covers all related operators and achieves 6.4× area reduction than prior synthesis-based work LegoZK [4], which currently makes it the most cost-effective candidate for the practical deployment of ZKP-PBC applications. As a case study, the processor performs 2^{16}-gate proof generation/verification with 0.15J/0.11mJ, offering two orders of magnitude energy savings over a 14-core 2.4GHz CPU. For the Pairing operator, it achieves development agility to enable fast design space exploration. It delivers 17×/1.8× improvements of area-time product (ATP)/energy compared to the prior ASIC work [5].

***Index Terms*—ZKP, zkSNARK, PBC, Crypto-processor**

I. Introduction

THE rapid growth of cloud computing, Internet of Things (IoT), and mobile devices has greatly enhanced convenience and intelligence in our daily life, but also expanded the attack surface, making systems more vulnerable to adversarial access and control. Consequently, data security issues have become increasingly severe. Traditional cryptographic schemes ensure confidentiality during storage and transmission but fail to protect data in use at low-trust environments. For example, users often need to disclose personal information to third-party providers when accessing services, exposing sensitive data to potential misuse. Relying solely on regulatory measures to safeguard privacy remains insufficient due to their inherent limitations. In this context, *privacy-preserving computation* technologies [6] have become the cornerstone for ensuring data security and confidentiality in the low-trust environment.

Typical privacy-preserving computation techniques include Fully Homomorphic Encryption (FHE) [7], Multi-Party Computation (MPC) [8], and Zero-Knowledge Proofs (ZKPs) [9]. Among them, FHE incurs the highest computational overhead, while MPC suffers from the largest communication cost. ZKPs lie between the two in both computation and communication complexity, making them more practical for near-term deployment with various promising applications [10]. The concept of ZKP originates from the seminal work [11] of Goldwasser et al. in the 1980s, where a prover demonstrates that $F(x, w) = 1$ holds for a secret input x and a public input w, without revealing any information about x. A standard ZKP scheme satisfies three properties: correctness, soundness, and zero-knowledge. The first two ensure that neither party can deceive the other, while the third guarantees that the verifier gains no useful information beyond the validity of the statement.

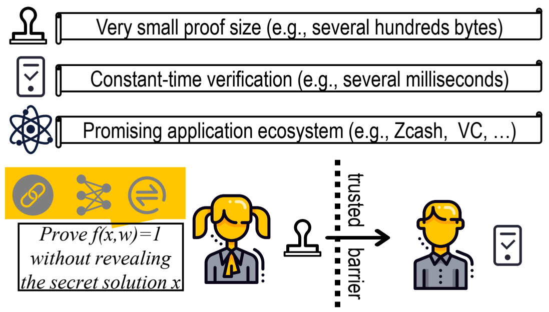
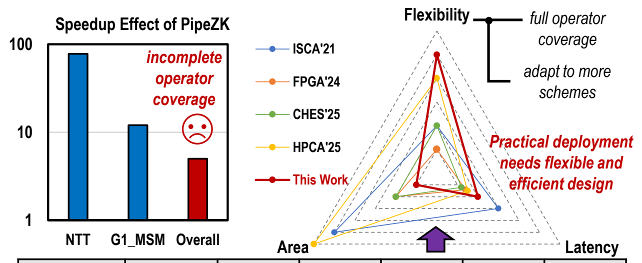

Very small proof size (e.g., several hundreds bytes)
Constant-time verification (e.g., several milliseconds)
Promising application ecosystem (e.g., Zcash, VC, ...)
Prove f(x,w)=1 without revealing the secret solution x
trusted barrier

Fig. 1. Features of Pairing-based zkSNARK.

Modern ZKP protocols are universal, meaning that proofs can be generated for arbitrary computational programs. Over the past decades, various ZKP schemes have been proposed, differing in setup, proof size and computation overhead. For example, the zkSTARK (Zero-Knowledge Scalable Transparent Argument of Knowledge) family [12] requires no trusted setup but typically results in very large proofs (e.g., 8 MB for Orion [13]) and higher proving and verification costs, making it less suitable for cost-efficient scenarios. Different application domains emphasize different characteristics. In privacy-preserving transactions, for instance, proofs must be efficiently broadcast across numerous distributed blockchain nodes, motivating the use of schemes with compact proof sizes. One of the most mature and widely adopted protocols is the Pairing-based zkSNARK (e.g., Groth16 [1], Plonk [2], Halo2 [14]), which is the focus of this work. As shown in Fig 1, Pairing-based zkSNARKs feature succinct proof sizes and constant-time verification independent of the circuit size. They have been explored or deployed in various domains, such as blockchain [15], cryptocurrencies (e.g., Zcash [16]), Web3 [17], verifiable machine learning (zkML) [18], verifiable outsourced computation (VC) [19], and anonymous e-voting [20]. Although lattice- and hash-based ZKPs have recently emerged, they remain at an early stage. Their proof sizes and proving times are orders of magnitude larger, and verification costs remain heavy.



Features	Core Area	Latency	Reconfig	G2 MSM	Pairing	Tapeout
ISCA'21[21]	Very High	High	—	—	—	—
FPGA'24[22]	High	Low	✓	—	—	—
CHES'25[23]	Medium	Medium	✓	—	—	—
HPCA'25[4]	Very High	Low	✓	✓	—	—
This Work	Very Low	Medium	✓	✓	✓	✓

Fig. 2. Feature comparison of existing accelerators

In recent years, numerous hardware accelerators targeting Pairing-based zkSNARKs have emerged [4], [21]–[23], covering both ASIC and FPGA platforms. These works can be broadly categorized into two levels: (i) end-to-end proof acceleration systems, and (ii) core subsystem modules such as Number Theoretic Transform (NTT), Inverse NTT (INTT) and Multi-Scalar Multiplication (MSM). As summarized in Fig 2, most existing designs overemphasize latency reduction while neglecting flexibility, often leading to significant area overheads that hinder widespread deployment and practical adoption. PipeZK [21] represents the first publicly reported synthesis-based ASIC accelerator for ZKPs. Although it achieves decent speedups for the NTT and G1-MSM operations individually, the overall proof generation is accelerated by only 5$\sim$10$\times$. This is because other operations, such as vector multiplication and G2-MSM, remain executed in software and thus become new bottlenecks (Amdahl's law). Moreover, PipeZK employs separate data paths for NTT and MSM, resulting in low overall hardware utilization. The work in [22] implements a split CPU–FPGA G1-MSM architecture using the Hardcaml language, which won first place in the 2022 ZPrize FPGA track. However, it supports only single operator, exhibiting low latency at the cost of substantial area and power consumption. Another work [23] focuses on NTT and G1-MSM operators, proposing a unified and reconfigurable architecture that achieves notable improvements in area efficiency and performance over PipeZK. Nevertheless, it does not implement the full proof protocol, leaving the end-to-end acceleration unassessed. A recent study [4] introduces a synthesis-based ZKP accelerator based on a Network-on-Chip (NoC) architecture, capable of supporting the complete proof generation. Despite its comprehensiveness, the design still incurs large area (68.79 mm^2) and high power consumption (11.39 W). Moreover, it relies on costly HBM to achieve high bandwidth, without analyzing the balance between on-chip computation and off-chip memory access, which easily lead to inefficient resource utilization.

Nearly all existing works focus solely on accelerating the proving phase, without considering hardware support for verification. There remains much room for improvement in both area efficiency and flexibility. In fact, the main operator in the verification phase is the Pairing operation, which remains computationally intensive with a hierarchical and complex dataflow. On the BN254 curve, a single Pairing requires approximately 10,000 modular multiplications and 57,000 modular additions [24]. On desktop CPUs, Pairing latency is more than twice that of a digital signature operation [25], motivating numerous prior studies on hardware acceleration of Pairing [25]–[27]. We observe that the core computational unit of Pairing remains the large-bitwidth modular multiplier, which is also the most resource-consuming component. Therefore, the computational units used in the proving phase can be reused to accelerate verification and improve energy efficiency, with negligible additional area overhead. Moreover, Pairing is also a core operation in many other cryptographic schemes, such as BLS signatures, functional encryption (FE), and identity-based encryption (IBE). While many studies focus on low-power solutions [5], [27], [28], this work explores a high-performance accelerator designed for environments where resources are not constrained, such as server systems. By designing a programmable Pairing architecture, broader support for various PBC schemes can be potentially achieved, reducing both hardware redesign costs and development time.

Based on the above insights, we aim to design an accelerator that enables broad deployment and practical realization of ZKP and PBC systems. The proposed design targets latency that meets real-world application requirements (less than 1 second per typical proof) while greatly reducing area/energy overhead and still maintaining flexibility. The main contributions of this work are summarized as follows:

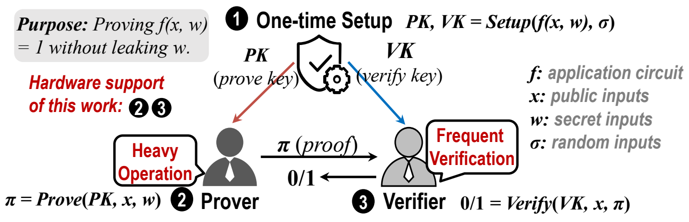
- We propose a scheme-adaptive crypto-processor that supports all related operators, including Pairing and G2-MSM. Compared with LegoZK [4], our design reduces area overhead by 6.4$\times$ and bandwidth demand by 10$\times$. It supports complete proof generation and verification, and can further adapt to other PBC schemes. This makes it the most cost-effective candidate for practical ZKP deployment while enabling reuse across diverse PBC schemes to avoid redesign cost.
- At the architectural level, we design a hybrid-grained instruction set to support on-chip computation and off-chip memory access. A shared and configurable memory system is introduced to jointly support polynomial computation and MSM, reducing memory usage by 6$\times$. We further propose a configurable modular multiplier tailored for the BLS12-381 curve, supporting orthogonal combinations of dataflow modes (fully pipelined, iterative) and bit-widths (256b, 284b). Based on this multiplier, we develop a highly reconfigurable datapath that intensively reused for various operators, achieving around 3.9$\times$ area reduction.
- At the operator level, we mainly design several engines for the programmable Pairing, the configurable polynomial operation (Poly-OP) and the unified G1/G2-MSM.

Specifically, we develop a custom Pairing compiler that performs near-optimal instruction scheduling and memory allocation automatically, enabling rapid development of generic tower-field arithmetic and elliptic curve cryptography (ECC) related functions. For the Poly-OP engine, we innovatively reduce the theoretical complexity of coset (I)NTT and design a conflict-free I/O interconnect along with two computing groups that enables task overlapping. To achieve near 100% utilization for the MSM engine, we introduce a state-aware scheduling method to handle potential colliding points and devise a handshake mechanism to efficiently synchronize the cascaded-loop-based PADD.

We integrate the proposed crypto-processor into a complete SoC (System-on-Chip) system that includes a practical DDR interface, CPU, system bus, and related peripherals. To the best of our knowledge, this is the first 28nm silicon-verified crypto-processor that supports both proof generation and verification, while remaining adaptable to other PBC. The processor achieves a sound balance between flexibility and latency, with a core area of only 8.3mm^2, operating at 500MHz under 0.9V. Its performance highlights are summarized as follows: 1) As a case study, the processor performs a 2^{16}-gate proof generation/verification with 0.15 J/0.11 mJ energy, achieving two orders of magnitude energy savings over a 14-core 2.4 GHz CPU. 2) For a 2^{20}-size G1-MSM, it achieves approximately 12.2×/6.2× improvements in area and energy efficiency, respectively, compared with the advanced work PipeZK [21] under similar off-chip memory settings. 3) For the Pairing operator, it achieves development agility to enable fast design space exploration with a custom Pairing compiler. It provides 17×/1.8× gains in ATP and energy over the prior ASIC design [5].

II. Background and Motivation

In this section, we first present the typical workflow and main computational procedures of Pairing-based zkSNARKs, followed by a quantitative analysis of the computational cost. We then briefly introduce other PBC, outlining the mathematical foundations of Pairing and its role in these schemes to motivate our pursuit of a hybrid ZKP–PBC application. Finally, we discuss the potential challenges in designing a scheme-adaptive and cost-efficient crypto-processor.



Purpose: Proving f(x, w) = 1 without leaking w.
❶ One-time Setup PK, VK = Setup(f(x, w), σ)
PK (prove key)
VK (verify key)
Hardware support of this work: ❷❸
Heavy Operation
π (proof)
0/1
Frequent Verification
f: application circuit
x: public inputs
w: secret inputs
σ: random inputs
π = Prove(PK, x, w) ❷ Prover
❸ Verifier 0/1 = Verify(VK, x, π)

Fig. 3. The typical workflow of zkSNARK (Groth16).

A. Pairing-based zkSNARK

zkSNARK is constructed based on Pairing-friendly elliptic curves. This work adopts the IETF (Internet Engineering Task Force)-recommended BLS12-381 curve of about 128-bit security level rather than the BN254 curve of about 100-bit security level. Compared with the BLS12-377 curve [29], BLS12-381 curve also features low-cost point addition on its twisted curve. The BLS12-381 curve and its twisted form are defined as: $E(F_q): y^2 = x^3 + 4 \bmod q$, $E(F_{q^2}): y^2 = x^3 + 4 \cdot (i+1)$, where q denotes the 381-bit modulus. The generators on the curves are $G_1 = (x, y)$ and $G_2 = (x_0 + i \cdot x_1, y_0 + i \cdot y_1)$, with coordinates represented as 381-bit integers over the finite field $\mathbb{F}_q$. Typical elliptic curve group operations include point addition (PADD), point doubling (PDBL), and scalar point multiplication (PM) $s \cdot P$, where s is a 255-bit integer from another scalar field $\mathbb{F}_p$. In zkSNARKs, the polynomial computations like NTT are executed over $\mathbb{F}_p$, while MSM involve inner-product computations between vectors over $\mathbb{F}_p$ and point vectors over $\mathbb{F}_q/\mathbb{F}_{q^2}$.

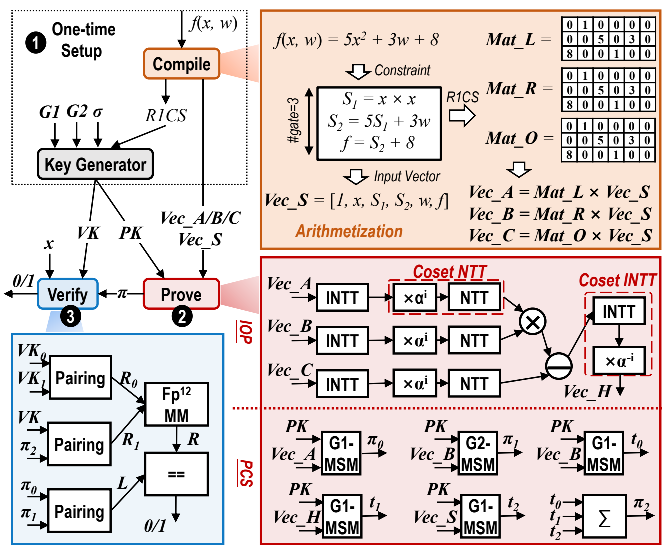
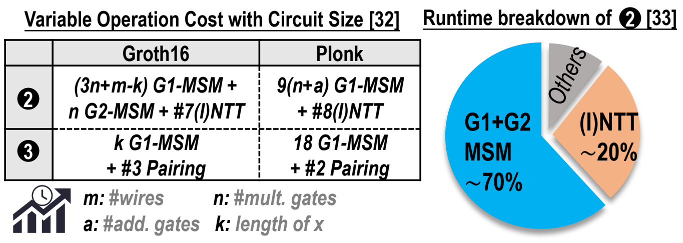


Fig. 4. The main computation process of Pairing-based zkSNARK (Groth16).

Fig 3 illustrates the typical workflow of a zkSNARK protocol. In the initial trusted setup phase, the system receives the application circuit, public/private/random inputs, generating the proving key (PK) and verification key (VK), which are distributed to the prover and verifier, respectively. For a given function f, most computations in the setup phase are performed only once, and their cost is amortized across multiple proofs. The prover then uses PK to generate proofs, which incurs significant computational overhead. Finally, the verifier uses VK and the proof to check the validity of the statement. Our hardware accelerator targets both the proving and verification phases, since a participant can act as either prover or verifier. Section I also discusses the motivation of supporting Pairing operator. In certain scenarios, a user needs to verify many proofs from others to validate blocks, confirm transactions, or maintain local state. Consequently, verification occurs frequently as seen in applications like Ethereum [30] or zkRollup [31]. Therefore, providing hardware acceleration for both proving and verification, without incurring significant resource overhead, is nothing short of meaningfulness.

Using the Groth16 algorithm [1] as an example, the main computational flow of the three phases is illustrated in Figure 4. The setup phase consists of compilation and key generation. Compilation transforms the target circuit f into an arithmetic representation, producing a rank-1 constraint system (R1CS). Based on the R1CS and the input vector Vec_S, three output vectors Vec_$A/B/C$ are derived. Fig 4 provides an example using a quadratic equation. The key generation step then takes the R1CS, random inputs, and elliptic curve parameters to produce the PK and VK. The proving phase mainly consists of two components: the Interactive Oracle Proof (IOP) and the Polynomial Commitment Scheme (PCS), whose main operations are Poly-OP and MSM, respectively. The IOP computes the quotient polynomial $H(x) = \frac{A(x)B(x)-C(x)}{x^n-1}$, while the PCS transforms the polynomials $A(x)$, $B(x)$ and $H(x)$ into commitments $\pi_0 \sim \pi_2$. The proof π consists of two G_1 points (π_0, π_2) and one G_2 point (π_1). For BLS12-381 curve, the Groth16 proof size is 192 bytes. In the verification phase, the bilinearity property of Pairing is used to check the relationships among $\pi_0 \sim \pi_2$ in a blinded manner, thus ensuring the zero-knowledge property.



	Groth16	Plonk
❷	*(3n+m-k) G1-MSM + n G2-MSM + #7(I)NTT*	*9(n+a) G1-MSM + #8(I)NTT*
❸	*k G1-MSM + #3 Pairing*	*18 G1-MSM + #2 Pairing*

Fig. 5. Analysis of operation cost for Pairing-based zkSNARK.

Fig 5 quantifies the main operations and their proportions in the proof generation and verification phases for two representative zkSNARK schemes: Groth16 and Plonk [32]. It shows that the proving phase is dominated by (I)NTT and MSM, whose computational cost scales with the number of gates and wires in the R1CS, i.e., the circuit size. For large-scale application circuits, such as a single Zcash transaction, n and m can be around 2^{20}. On software platforms, MSM accounts for approximately 70% of the proving phase, while (I)NTT accounts for about 20% [33]. In the verification phase, the primary operation is Pairing. Although its latency on software platforms is relatively small, verification can occur far more frequently than proof generation in certain scenarios, consuming significant time and energy. Consequently, our hardware acceleration targets also include Pairing, but with negligible additional area overhead.

B. Other Pairing-based Cryptography

Bilinear Pairings on elliptic curves have become a cornerstone of modern cryptography, enabling advanced constructions such as short signatures, identity-based encryption, and functional encryption. A Pairing is a bilinear, non-degenerate map $e : G_1 \times G_2 \rightarrow G_T$ where G_1, G_2, and G_T are cyclic groups of prime order p, satisfying the bilinearity property $e(aP, bQ) = e(P, Q)^{ab}$ for all $a, b \in \mathbb{Z}_p$. The algebraic structure of Pairings allows cryptographic protocols to express multiplicative relations across different group elements, which is infeasible in conventional public-key settings.

BLS signature. The Boneh–Lynn–Shacham (BLS) signature scheme [34] is one of the most well-known applications of bilinear Pairings. It provides short, constant-size signatures and supports signature aggregation, where multiple signatures on distinct messages can be merged into a single compact one. This property makes BLS widely adopted in blockchain systems (e.g., Ethereum 2.0 and Chia) and distributed consensus protocols, where communication bandwidth and storage are critical. Verification relies on a single Pairing equation: $e(\sigma, g_2) = e(H(m), pk)$, where $(H(m))$ is the message hash mapped to G_1. The simplicity and efficiency of this verification process have made BLS an industry-standard for scalable digital signatures.

Identity-Based Encryption (IBE). The Boneh–Franklin scheme [35] was the first practical realization of IBE using Pairings. In this system, a user's identity string (e.g., email address) acts as a public key, and a trusted authority derives the private key using the main secret. The bilinear Pairing enables a sender to compute a shared group element with the recipient's identity without prior key exchange. IBE removes the need for traditional PKI infrastructures, making it ideal for ad hoc networks, sensor systems, and military communications, where pre-distribution of certificates is impractical.

Functional Encryption (FE). Functional encryption generalizes traditional encryption by allowing a key holder to learn a specific function $f(m)$ of the plaintext rather than the message itself. Pairing-based FE schemes [36], [37] exploit the bilinearity of Pairings to encode controlled relationships among ciphertext components and keys, thereby enabling expressive access structures and fine-grained data sharing. Applications include privacy-preserving machine learning, secure cloud computation, and data monetization systems.

Discussions on hybrid ZKP-PBC applications. ZKPs provide integrity, while PBC schemes like IBE ensure confidentiality, making them complementary. In some cases, they can be used independently; in others, they are combined [38]. A user may act as both encryptor and prover, requiring secure communication or access control (via IBE) while proving statements without revealing identity or sensitive data (via ZKP). For example, in privacy-preserving transactions, IBE binds real-world identity for regulatory purposes, while ZKP attests to transaction validity. In anonymous authentication, IBE registers the user, and ZKP proves possession of a valid identity without disclosing it.

C. Design Challenges

Challenge 1: diverse and large-scale computation. Fig 6 illustrates the hierarchical operator framework of the proof generation and verification (Pairing). The Pairing computation mainly consists of two parts: *Miller Loop* and *Final Exponentiation*. The Miller Loop involves iterative line functions such as line add., double, and multiplication. Final Exponentiation is divided into an easy part and a hard part—the former can be simplified to lightweight operations via the Frobenius map, while the latter is computation-intensive, involving operations like Cyclotomic Square and Cyclotomic Exponentiation over the extension field $\mathbb{F}_{p^{12}}$. Pairing relies on modular arithmetic across tower extension fields to construct higher-level functions. Overall, it features diverse and serialized scalar operations with complex data dependencies. In the proof generation phase, the major computations include coset (I)NTT and vectorized modular operations over the 256-bit scalar field, and G1/G2-MSM. The modular multiplications required by G2-PADD are roughly four times those of G1-PADD, since each G2 coordinate lies in $\mathbb{F}_{p^2}$, composed of two 381-bit elements. Previous works either support only G1-MSM or handle G2-MSM with considerable area overhead, still leaving room for optimization. Although the proof generation involves fewer operation types than Pairing, its vectorized computation scale can be around 2^{20}. Both Pairing and MSM/NTT rely on large-width modular multipliers, motivating the design of a flexible and efficient multiplier to support all target schemes. Fig 7 further summarizes the number of $\mathbb{F}_p$ multiplications required for different tower fields, along with vector lengths, bit-widths, and data formats of NTT/MSM.

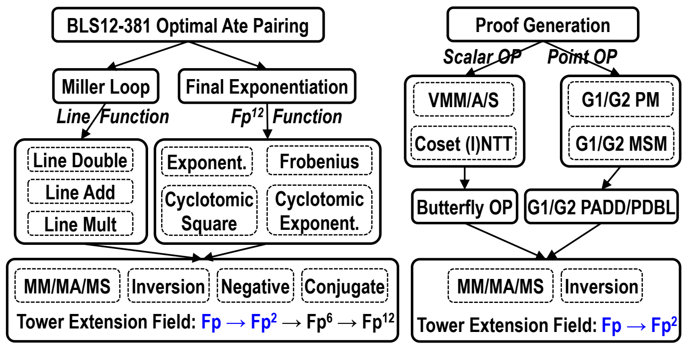
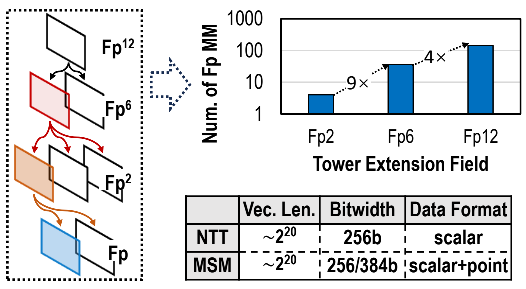


Fig. 6. Challenge-1: heavy and diverse operations of proof generation and verification (Pairing).



	Vec. Len.	Bitwidth	Data Format
NTT	$\sim 2^{20}$	256b	scalar
MSM	$\sim 2^{20}$	256/384b	scalar+point

Fig. 7. The modular multiplication cost of tower extension field and typical size of MSM/NTT in ZKP.

Challenge 2: frequent strided accesses to On-/Off-chip memory. In the proof generation, the large vector length and bit-width exceed the capacity of limited on-chip memory. Hence, vector computations must be partitioned into independent sub-tasks. Fig 8 illustrates the computation flow and access pattern of a two dimensional (2D) NTT as an example. First, both row-wise and column-wise computations require frequent data transfers between off-chip and on-chip memory—loading input data for processing and writing back intermediate results—leading to significant access latency. Second, within each on-chip sub-NTT, the address stride varies with the computation stage, resulting in complex access patterns. This complexity is further amplified when multiple banks are accessed in parallel, potentially leading to memory access conflicts [39].

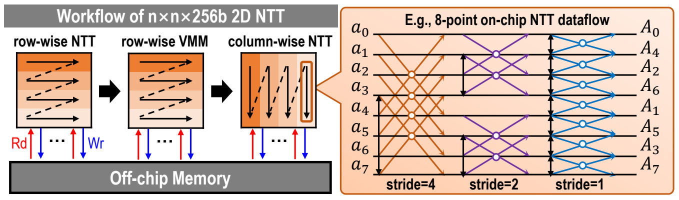

Workflow of n×n×256b 2D NTT
row-wise NTT
row-wise VMM
column-wise NTT
Rd
Wr
Off-chip Memory
E.g., 8-point on-chip NTT dataflow
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
A_0 A_4 A_2 A_6 A_1 A_5 A_3 A_7
stride=4
stride=2
stride=1

Fig. 8. Challenge-2: frequent on/off-chip memory access with complex stride address patterns.

Challenge 3: dynamic and sequential dataflow. Fig 9 illustrates the computational flow of MSM, which essentially performs a distributed dot-product operation. Taking a window size of 4 bits as an example, each 12-bit scalar in vector k is divided into three 4-bit segments, splitting the task into three independent parts. During the *bucket accumulation* phase, points P_i are assigned to buckets indexed by their corresponding 4-bit scalar segments, and points with identical scalar values are accumulated in the same bucket. The *bucket reduction* phase aggregates points within each bucket into an intermediate point ($G_1 \sim G_3$). Finally, the *group aggregation* phase combines all $G_1 \sim G_3$ points into a single output based on window weights. Since scalar values are random and unpredictable, the first phase exhibits a dynamic dataflow, which may cause colliding points and complicate controller design. Moreover, the second and third phases involve inherently sequential addition chains, leading to low hardware utilization with numerous pipeline bubbles.

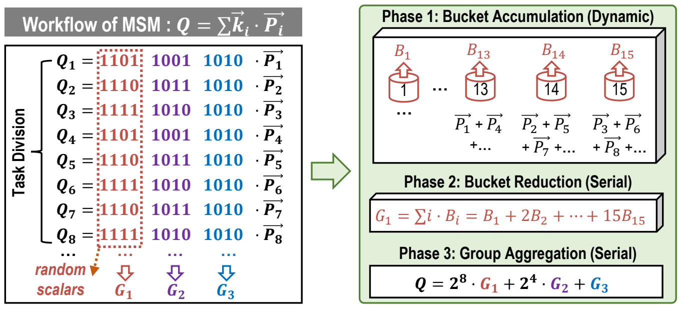

Workflow of MSM : $Q = \sum \vec{k_i} \cdot \vec{P_i}$
Task Division
Q_1 = 1101 1001 1010 · $\vec{P_1}$
Q_2 = 1110 1011 1010 · $\vec{P_2}$
Q_3 = 1111 1010 1010 · $\vec{P_3}$
Q_4 = 1101 1001 1010 · $\vec{P_4}$
Q_5 = 1110 1011 1010 · $\vec{P_5}$
Q_6 = 1111 1010 1010 · $\vec{P_6}$
Q_7 = 1110 1011 1010 · $\vec{P_7}$
Q_8 = 1111 1010 1010 · $\vec{P_8}$
random scalars
G_1 G_2 G_3
Phase 1: Bucket Accumulation (Dynamic)
B_1 B_{13} B_{14} B_{15}
1 13 14 15
$\vec{P_1}+\vec{P_4}$ +...
$\vec{P_2}+\vec{P_5}$ $+\vec{P_7}$ +...
$\vec{P_3}+\vec{P_6}$ $+\vec{P_8}$ +...
Phase 2: Bucket Reduction (Serial)
$G_1 = \sum i \cdot B_i = B_1 + 2B_2 + \cdots + 15B_{15}$
Phase 3: Group Aggregation (Serial)
$Q = 2^8 \cdot G_1 + 2^4 \cdot G_2 + G_3$

Fig. 9. Challenge-3: dynamic and serial dataflow incurs complex controller design and resource underutilization.

III. Overall System Architecture

In this section, we present the system-level architecture of the proposed crypto-processor. The discussion covers the design of hybrid-grained instruction set, programmable controller, shared and configurable memory system, and PE array built upon the reconfigurable modular multiplier.

A. Programmable and Hierarchical Controller

Fig 10 depicts the system architecture, including a top controller, multiple sub-controllers, configurable data paths,

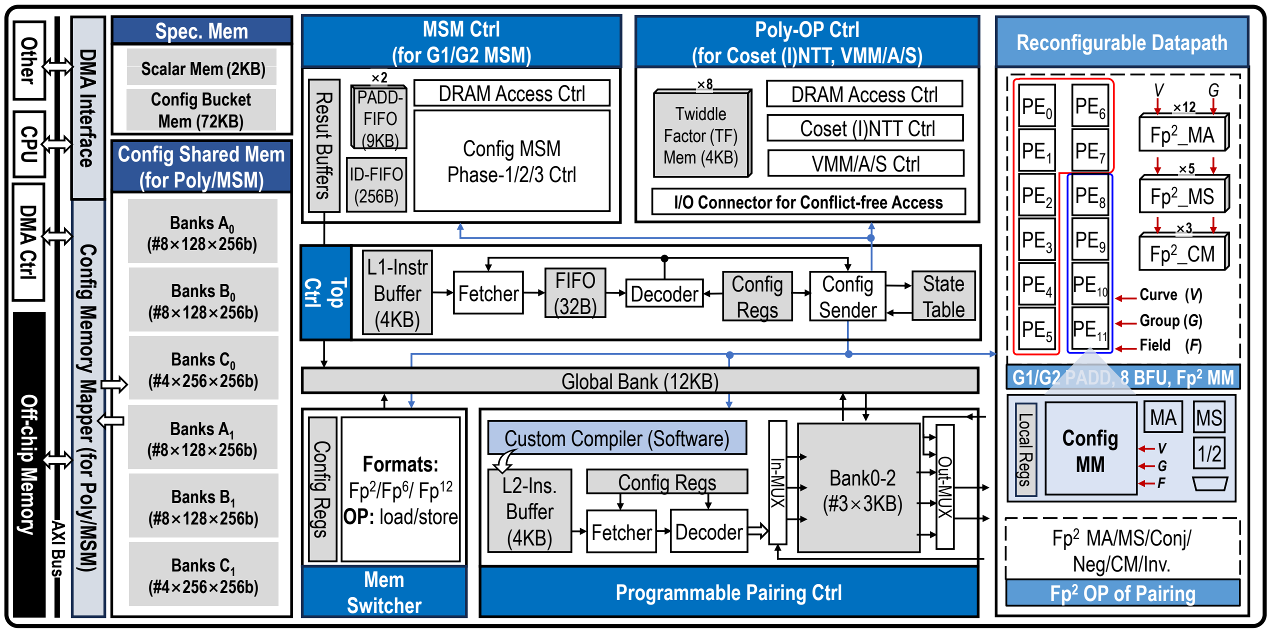
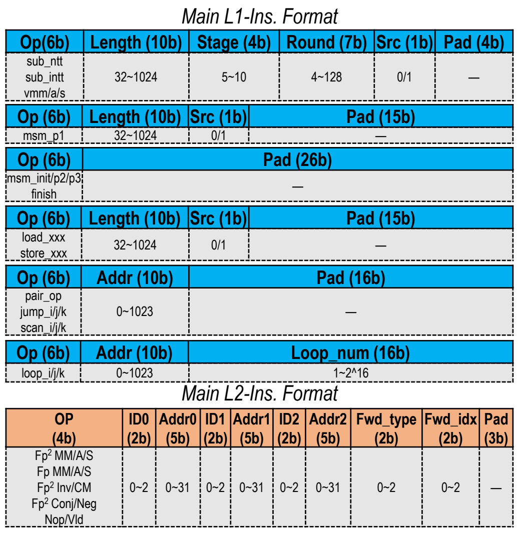


Fig. 10. The overall system architecture.



Main L1-Ins. Format

Op(6b)	Length (10b)	Stage (4b)	Round (7b)	Src (1b)	Pad (4b)
sub_ntt sub_intt vmm/a/s	32~1024	5~10	4~128	0/1	—

Op (6b)	Length (10b)	Src (1b)	Pad (15b)
msm_p1	32~1024	0/1	—

Op (6b)	Pad (26b)
msm_init/p2/p3 finish	—

Op (6b)	Length (10b)	Src (1b)	Pad (15b)
load_xxx store_xxx	32~1024	0/1	—

Op (6b)	Addr (10b)	Pad (16b)
pair_op jump_i/j/k scan_i/j/k	0~1023	—

Op (6b)	Addr (10b)	Loop_num (16b)
loop_i/j/k	0~1023	1~2^16

Main L2-Ins. Format

OP (4b)	ID0 (2b)	Addr0 (5b)	ID1 (2b)	Addr1 (5b)	ID2 (2b)	Addr2 (5b)	Fwd_type (2b)	Fwd_idx (2b)	Pad (3b)
Fp^2 MM/A/S Fp MM/A/S Fp^2 Inv/CM Fp^2 Conj/Neg Nop/Vld	0~2	0~31	0~2	0~31	0~2	0~31	0~2	0~2	—

Fig. 11. The primary L1/L2-instruction formats with hybrid-grain.

and shared/specific memory. The L1-Memory of top controller stores coarse-grained L1-instructions to configure and schedule sub-controllers for fulfilling the target scheme. The Configurable sender looks up the state table of data path and memory to dynamically dispatch the L1-instruction, which supports out-of-order execution to enable the parallelism of operations. The programmable Pairing controller also stores fine-grained L2-instructions to support underlying operations over the tower extension field, which are grouped to further form various high-level functions. The Poly-OP and MSM controllers afford to manage both DRAM access and on-chip computations. The Mem. Switcher exchanges data in different formats between Bank0-2 and the Global bank, updating the I/O data during function switching.

Fig 11 illustrates the primary hybrid-grained instruction formats, including the L1- and L2-level instructions, which are categorized into computation, memory, and control types. The computation-type L1 instructions cover polynomial computation, vector operations, MSM and general ECC operations. For polynomial operations, the instruction format contains fields such as `length`, `stage`, and `round`, allowing desirable configurations of the sub-(I)NTT size and 2D (I)NTT size. The instruction format for the first MSM stage (`msm_op1`) is similar, where the `length` field specifies the vector size processed on-chip per iteration. Instructions for Pairing computation are organized into two levels: L1 and L2. The L1 instruction specifies the starting address of each Pairing subroutine, while each subroutine consists of L2 instructions. The L2 instruction enables fine-grained operations, including modular multiplication, inversion, addition and subtraction over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. These primitive operations can be composed into higher-level ECC functions. Additionally, the L2 format explicitly encodes data forwarding types to simplify control logic and minimize pipeline bubbles. Memory-access-type instructions manage communication between on-chip and off-chip banks. They specify, via the `Op` field, which on-chip bank needs the load/store operation, and the `length` field indicates the data transfer size. The CPU reads these instructions to configure the Direct Memory Access (DMA) engine, enabling flexible data movement between off-chip and on-chip memory. Control-related instructions implement loop, branch and conditional execution. Three independent loop types (`loop_i/j/k`) are supported, each with standalone counters, iteration counts and jump addresses, thereby allowing nested loop structures. By combining loop instructions with computation or memory related instructions, the architecture can efficiently support data movement and vectorized computations of desirable sizes.

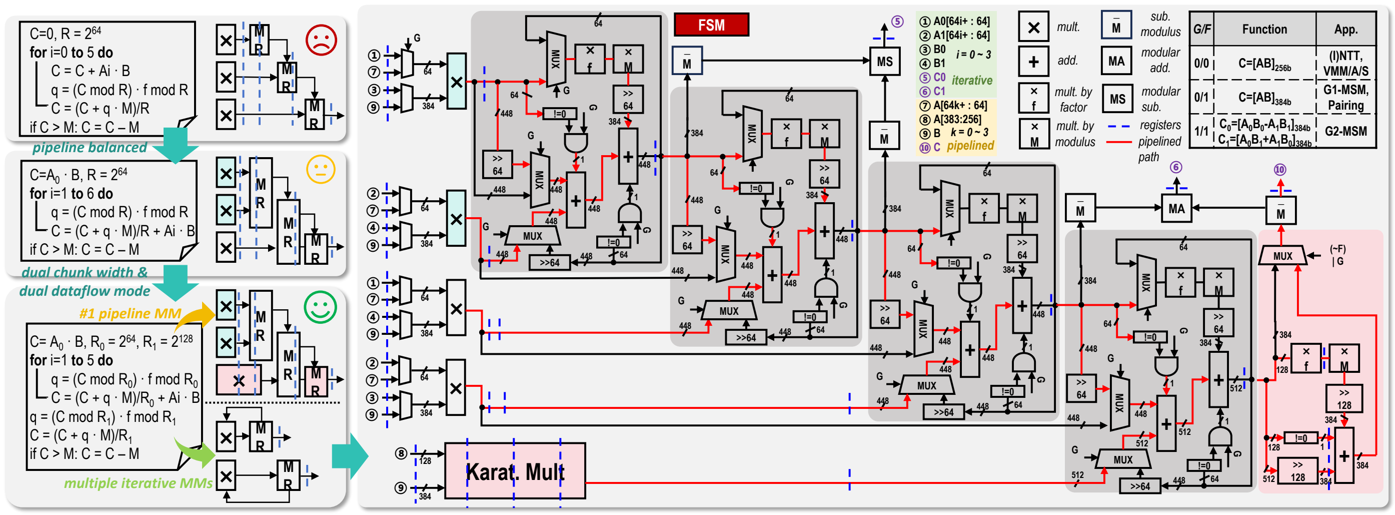


Fig. 12. Design rationale for the proposed new configurable modular multiplier with pipelined and iterative modes

B. Dedicated Memory System

To enhance area efficiency, we design a configurable and shared memory system, as shown in Fig 10, consisting of `Banks_A0/1`, `Banks_B0/1`, and `Banks_C0/1`. This organization can be configured for either polynomial computation or MSM modes. Both `Banks_A0/1` and `Banks_B0/1` comprise eight single-port banks, each with a capacity of 4096 KB (128 × 256 b), while `Banks_C0/1` includes four single-port banks, each of 4096 KB (256 × 256 b). During each stage of the coset (I)NTT computation, data are alternately transferred in a ping-pong fashion between `Banks_A0` and `Banks_B0`. Meanwhile, `Banks_A1` and `Banks_B1` serve as another set of banks, enabling one data batch to be loaded from DRAM while the other batch is processed on-chip, thereby hiding memory access latency. `Banks_C0/1` are employed for vector modular operations. During G1-MSM, each elliptic curve point is 768 bits, which can be accommodated by concatenating three 256-bit-wide banks. Thus, `Banks_A0`, `Banks_B0`, and `Banks_C0` jointly form a 1024×768 b memory organization for the G1-MSM. Similarly, `Banks_A1`, `Banks_B1`, and `Banks_C1` form another 1024×768 b configuration to prefetch data from DRAM while computation is ongoing, hiding off-chip memory access latency as well. For G2-MSM computation, each elliptic curve point is 1536 bits (i.e., 2 × 768 b), and the bank organization follows a similar structure with a slight rearrangement.

C. Reconfigurable PE Array

The reconfigurable datapath mainly consists of the Config $\mathbb{F}_{p^2}$ OP unit used for Pairing, and the Config processing elements (PEs) used for all algorithms. Besides supporting G1/G2 PADD of MSM, the 12 PEs are also reconfigured into 8 butterfly units (BFUs) or 1 $\mathbb{F}_{p^2}$ modular multiplier (MM) by setting G/F control signals, respectively used for Coset (I)NTT and Pairing. Each PE mainly contains config modular adder/subtracter (MA/S) and a novel configurable Montgomery MM.

Fig 12 illustrates the detailed circuit architecture and design rationale of the Config MM. Starting from the traditional radix-64 Montgomery modular multiplication algorithm, we derive a dual-dataflow configurable architecture. We first observe that directly unrolling and pipelining the radix-based Montgomery algorithm leads to a large number of pipeline registers being inserted at the multiplier outputs to meet synchronization. Since paths between these registers contain no combinational logic, the resulting pipeline stages are highly imbalanced, causing poor area efficiency. To mitigate this issue, we reorder the operation order so that the first two multiplications are executed in parallel. Next, we introduce dual-chunk-width techniques to enlarge the operand width in the final multiplication stage, enabling the insertion of surplus registers inside the multiplier to balance the pipeline load. The derived algorithm can then be mapped into two operational modes: a fully pipelined modular multiplier or multiple iterative modular multipliers, depending on the application requirement.

As shown on the right side of Fig. 12, taking the BLS12-381 curve as an example, the proposed Config MM can be configured via control signals G/F into (1) a single 256-bit fully pipelined MM for polynomial computation, (2) a 384-bit fully pipelined MM for G_1 point addition and $\mathbb{F}_{p^2}$ MM for Pairing, (3) four 384-bit iterative MMs for $\mathbb{F}_{p^2}$ MM in G_2 point addition. Note that large-bitwidth MMs dominate both the area and power consumption of the processor. The proposed flexible and efficient Config MM enables high hardware utilization across diverse cryptographic workloads, which is also the primary reason why our processor achieves remarkably low area overhead compared to prior works.

IV. Design and Optimization for Main Operators

This section elaborates on the design and optimization of key operators—including Pairing, polynomial computation and MSM. These components are comprehensively co-optimized across algorithmic, architectural and compiler levels.

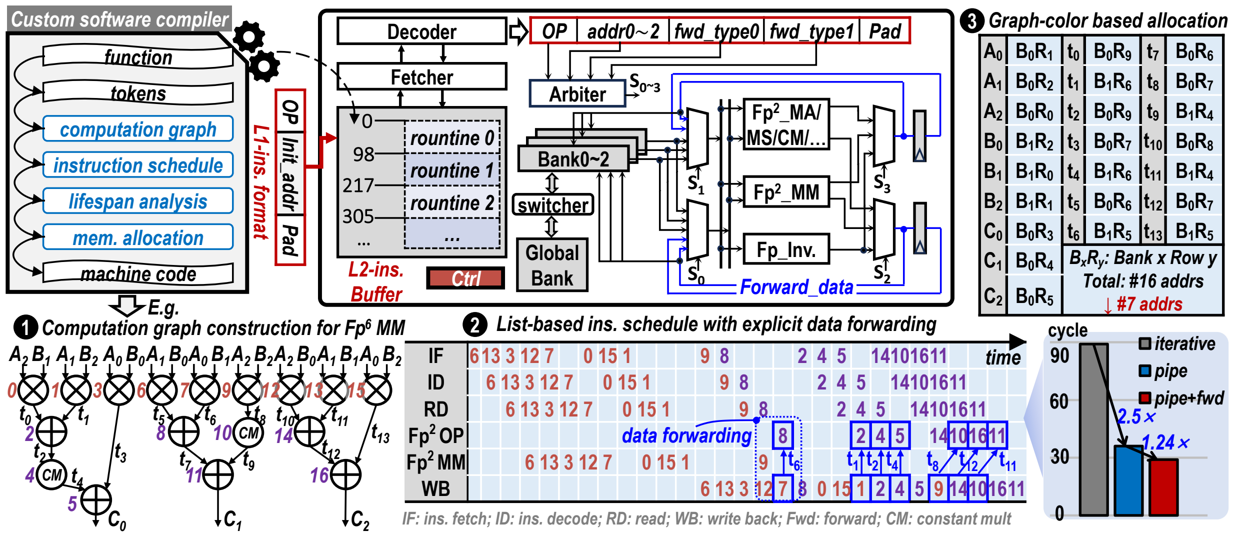


Fig. 13. The full-stack and efficient design framework for Pairing processor.

A. Agile Design Framework for Pairing Processor

As introduced in section II, Pairing computation involves a large number of operation types with intricate data dependencies. Designing a Pairing processor through manual scheduling would require substantial effort, which is hard to yield an optimal solution. To enable rapid design-space exploration and facilitate fast design iteration, we propose an agile design framework that co-optimizes the Pairing software compiler and the hardware architecture. Fig 13 illustrates the proposed full-stack design framework of the Pairing processor.

Hardware architecture. The Pairing hardware processor adopts an instruction-set architecture with a certain degree of programmability, enabling support for general ECC-related functions and various operations over tower extension fields. Similar to the top-level controller, it consists of instruction fetch, decode, execute, and write-back stages, but omits the scoreboard unit. Since Pairing instruction sequences are predefined and known in advance, instruction scheduling can be handled by the software compiler, thereby simplifying the control circuitry. Bank0–2 are pseudo-dual-port memory of size 32×768 b, where each row stores one $\mathbb{F}_{p^2}$ element and supports two reads and one write per cycle. The datapath includes three independent computation channels dedicated to fully pipelined modular multiplication over $\mathbb{F}_{p^2}$, single-cycle arithmetic over $\mathbb{F}_{p^2}$, and constant-time modular inversion over $\mathbb{F}_p$ based on Fermat's theorem. The processor also supports two types of data forwarding to minimize pipeline bubbles and improve hardware utilization.

Rationale of routine division. The Pairing algorithm is partitioned into multiple routines, with their start addresses specified by L1 instructions, enabling routine-level loops and jumps. Each routine comprises a sequence of compiler-generated L2 instructions. The way routines are divided significantly impacts scheduling efficiency and the total number of clock cycles. If a routine contains too few instructions, many pipeline bubbles may occur within the routine. These idle cycles could have been filled by instructions outside the routine without data dependencies, but improper division wastes this opportunity, increasing the overall cycle count. Switching between routines also incurs additional cycles and energy due to I/O data updates between Banks 0–2 and the global bank. Conversely, overly large routines harm reusability. For instance, if routines A and B share a common instruction set C, creating a separate routine for C allows reuse and avoids storing duplicate instructions. Beyond a certain size, increasing routine length does not improve scheduling and may lengthen compiler runtime. Considering these factors, we partitioned the Pairing algorithm into eight routines, summarized in Table I, balancing reusability and scheduling efficiency. For example, unlike prior work, we merged the line add and line mult. functions into a single routine, reducing the total clock cycles to 99, which is about 20% fewer than when scheduled separately.

TABLE I
THE SUMMARIZED ROUTINES FOR PAIRING.

	Rountine	#ins	Description
①	line_add_mult	98	The ins. routine for line add and line mult.
②	line_double_mult	119	The ins. routine for line double and line mult.
③	Fp^{12} MM	88	The ins. routine for Fp^{12} modular mult.
④	Fp^{12} Frobeniuis	25	The ins. routine for Fp^{12} frobenius map
⑤	Fp^{12} Frobenius2	53	The ins. routine for another Fp^{12} frobenius map
⑥	Fp^{12} Inversion	73	The ins. routine for Fp^{12} modular inversion
⑦	Cyclotomic Square	50	The ins. routine for cyclotomic square
⑧	Fp^{12} Conjugate	19	The ins. routine for Fp^{12} conjuagte

List-based scheduling with random selection and explicit data forwarding. We utilize a list-based scheduling algorithm for instruction scheduling, the pseudocode of which is presented in 1. This algorithm encompasses functions for updating the active instruction list, selecting instructions, performing scheduling, and updating the reservation table. Unlike conventional list-based scheduling, we introduce randomness in the instruction selection stage rather than always choosing the first instruction in the active list. Different selection strategies are compared in terms of total clock cycles, and the optimal scheduling method is chosen. Additionally, explicit data forwarding is encoded in each instruction field. If instruction i depends on instruction j, and ins_slot$[i] - 1 =$

Algorithm 1 List-based Scheduling with Random Selection

```
Input: ins_op_delay[ ], ins_dep_mat[ ][ ], machine[ ], RT[ ][ ]
Output: ins_slot[ ]
  active_list_ins = [ ]
  ins_slot = [0, 0, ..., 0]
  ins_state = [0, 0, ..., 0]
  while number of unscheduled instructions $\neq 0$ do
      // find instructions that can be activated based on ins_dep_mat
      active_list_ins = update_active_ins(ins_state, active_list_ins)
      if active_list_ins is not empty then
          // select schedule_ins from active_list_ins randomly
          schedule_ins = select_active_ins(ins_state, active_list_ins)
          // j $\in$ {predecessors of schedule_ins}
          ins_slot[i] = max {ins_slot[j] + ins_op_delay[j]}
          flag = True
          while flag do
              if RT[machine[i]][ideal_slot] is busy then
                  ins_slot[i] = ins_slot[i] + 1
              else
                  flag = False
              end if
          end while
          update the reservation table
          ins_state[schedule_ins] = 1
      end if
  end while
  return ins_slot
```

Notes:

- **ins_op_delay:** Cycle count for instructions of different operation types.
- **ins_dep_mat:** Data dependency matrix for all instructions.
- **machine:** The available computation units.
- **RT:** Reservation table with horizontal timing and vertical operation.
- **ins_slot:** The issue slot for each instruction.
- **ins_state:** Indicate whether the ins. is scheduled or not.

ins_slot$[j]$ + ins_op_delay$[j]$, it is assigned forwarding type 1; if ins_slot$[i]$ = ins_slot$[j]$+ins_op_delay$[j]$, it is assigned type 2; otherwise, no forwarding occurs. Specifying forwarding relationships at compile time simplifies hardware control logic and effectively improves utilization.

The three key compilation steps are illustrated using $\mathbb{F}_{p^6}$ MM as an example. Note that $\mathbb{F}_{p^6}$ MM just serves as a small-scale example for explanatory purposes. Based on the computation graph dependencies, instruction scheduling aims to minimize clock cycles by exploring the optimal operation order. Compared to iterative MM or pipelined MM without data forwarding, our pipelined design with data forwarding reduces the cycle count by 3.1× and 1.24×, respectively. After constructing a conflict graph from lifespan analysis, a graph-coloring method is applied to achieve memory allocation with minimal occupation.

B. Configurable Poly-OP Architecture

Algorithmic optimization of Coset (I)NTT. To our best knowledge, in nearly all existing libraries and prior works, such as Libsnark [40], Bellman [41], Gnark [42], and Arkworks [43], the coset NTT is computed by first performing a pointwise multiplication between the original coefficient vector of length N and a coset factor vector $[1, \alpha, \alpha^2, \ldots, \alpha^{N-1}]$, then followed by the NTT. Here, α denotes the primitive root of unity, which is 7 for the BLS12-381 curve. This pre-multiplication is necessary to avoid division by zero in computing $H(x) = (A(x)B(x) - C(x))/(x^N - 1)$, since directly applying NTT would yield $x^N - 1 = w^N - 1 = 0$, with w being an N-th root of unity. By introducing the coset factor, the denominator becomes $x^N - 1 = (w\alpha)^N - 1 = \alpha^N - 1$, eliminating the singularity. However, this approach incurs an additional N 256-bit modular multiplications, extra memory accesses, and storage for the $N \cdot 256$-bit coset factors, which becomes heavier for N on the order of 2^{20}. Inspired by NTT variants over polynomial rings [44], we replace the coset-factor vector with a $2N$-th root of unity ϕ and merge its powers into the twiddle factors at each stage, completely eliminating the explicit coset factor multiplication. Both theoretical analysis and experimental verification show that this optimization preserves correctness: the resulting quotient polynomial $H(x)$ is identical with the original solution. In this way, the computational complexity is reduced by N modular multiplications. Algorithm 2 provides the pseudocode for the low-complexity coset NTT; the similar approach also applies to coset INTT. To the best of our knowledge, this is the first work to identify this problem, optimizing the theoretical complexity of coset (I)NTT with a simple yet effective method.

Algorithm 2 Optimized Coset NTT Algorithm

```
Input: $a$ is a vector of length $N$, $\phi_{2N}$ is the $2N$-th root of unity, $\omega_N$ is the $N$-th root of unity, and $\alpha$ is the primitive root of unity (coset factor = 7);
Output: $A$ (vector of length $N$)
  // Pre-processing (coset factor multiplication) is eliminated
  // by replacing $\alpha$ with $\phi$ and merge $\phi^i$ into $\omega$
  $a = [a_0, a_1 \cdot \alpha, a_2 \cdot \alpha^2, \ldots, a_{n-1} \cdot \alpha^{n-1}]$ // Pre-processing
  $A = \text{scramble}(a)$
  for $p = 0$ to $\log_2 N - 1$ do
      $J = 2^p$
      for $j = 0$ to $J - 1$ do
          $\omega = \phi_{2N}^{(2j+1)N/2J}$   // old method $\omega = \omega_N^{jN/2J}$ is replaced
          for $k = 0$ to $N/2J - 1$ do
              $u = A_{(2kJ+j)}$
              $v = \omega \cdot A_{(2kJ+j+J)} \pmod{q}$
              $A_{(2kJ+j)} = (u + v) \pmod{q}$
              $A_{(2kJ+j+J)} = (u - v) \pmod{q}$
          end for
      end for
  end for
  return $A$
```

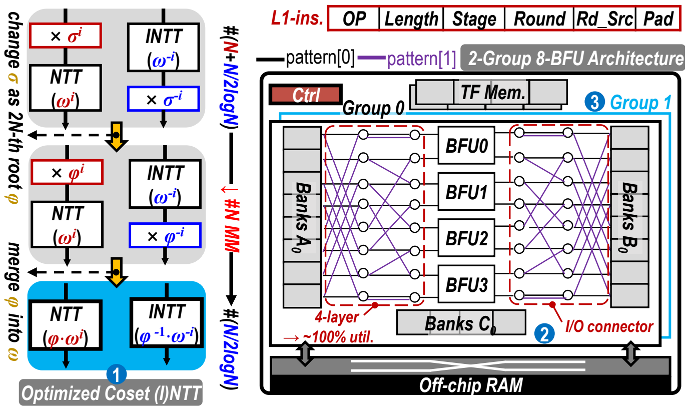


Fig. 14. (1) algorithmic optimization (2) custom 4-layer I/O connector (3) 2-group 8-BFU configurable Poly-OP architecture

Hardware architecture. Figure 14 illustrates the hardware architecture for polynomial computation, which consists of two sets of PEs, each containing four butterfly units. In in-place NTT computations, the address stride varies with the stage, so reading eight data points in parallel can easily cause bank conflicts. Prior works [44], [45] have discussed interconnect structures for small-width modulus in post-quantum cryptography; however, these either rely on fully connected topologies or are unsuitable for multi-bank architectures. In this work, we propose a lightweight interconnect topology along with the matched memory layout, as shown in Figure 14 and 16. The interconnect comprises four layers, each with multiple 2-to-1 multiplexers, which largely reduces the hardware overhead and logic path length compared to fully connected 8-input multiplexers [44]. Using a 16-point NTT as an example, Figure 16 also illustrates how the storage layout changes at each stage and round, along with the corresponding pattern selection strategies. The timing scheduling of 2D NTT is depicted in Figure 15, which leverages two PE groups to overlap the on-chip computations with off-chip memory accesses.

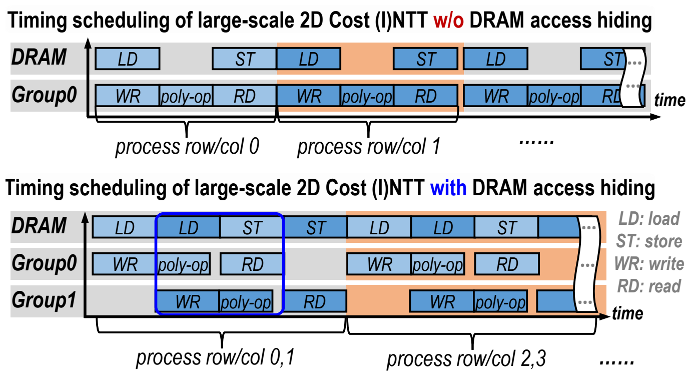
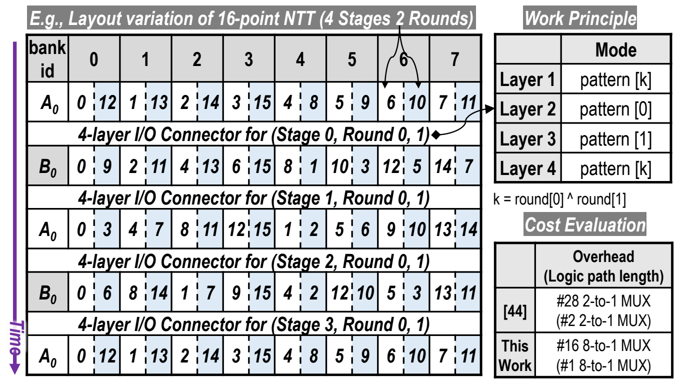


Fig. 15. The timing scheduling for large-scale 2D coset (I)NTT


E.g., Layout variation of 16-point NTT (4 Stages 2 Rounds)
Time
4-layer I/O Connector for (Stage 0, Round 0, 1)
4-layer I/O Connector for (Stage 1, Round 0, 1)
4-layer I/O Connector for (Stage 2, Round 0, 1)
4-layer I/O Connector for (Stage 3, Round 0, 1)
Work Principle
k = round[0] ^ round[1]
Cost Evaluation

bank id	0		1		2		3		4		5		6		7	
A_0	0	12	1	13	2	14	3	15	4	8	5	9	6	10	7	11
B_0	0	9	2	11	4	13	6	15	8	1	10	3	12	5	14	7
A_0	0	3	4	7	8	11	12	15	1	2	5	6	9	10	13	14
B_0	0	6	8	14	1	7	9	15	4	2	12	10	5	3	13	11
A_0	0	12	1	13	2	14	3	15	4	8	5	9	6	10	7	11

	Mode
Layer 1	pattern [k]
Layer 2	pattern [0]
Layer 3	pattern [1]
Layer 4	pattern [k]

	Overhead (Logic path length)
[44]	#28 2-to-1 MUX (#2 2-to-1 MUX)
This Work	#16 8-to-1 MUX (#1 8-to-1 MUX)

Fig. 16. The work principle and cost evaluation of I/O connector.

C. Unified G1/G2-MSM Architecture

Most prior works [21]–[23], [46], [47] only support hardware acceleration for G1-MSM and delegate G2-MSM to software execution, resulting in suboptimal overall acceleration effect. Thus, we propose the first unified architecture that fully reuses the PADD of G1-MSM for G2-MSM without adding extra MMs. Fig 17 portrays the unified G1/G2-MSM architecture that consists of 4 stages from point fetching to write back. The point memory (PM) rearranges banks from Poly-OP to serve G1 and G2 MSM, thereby reducing memory footprint by ∼3×. The Memory interface (inf.) implements the backpressure mechanism, whereby the upstream halts data output without packet loss when the downstream, per arbiter feedback, rejects the upstream. The bucket Memory is also reused to perform both G1 and G2 mode, where each curve point is mapped to the bucket of the same index to further form input pairs. Ulike prior works [46] [23] using true dual-port memory, this work adopts pseudo dual-port memory to implements the buckets, further reducing area cost without performance loss.

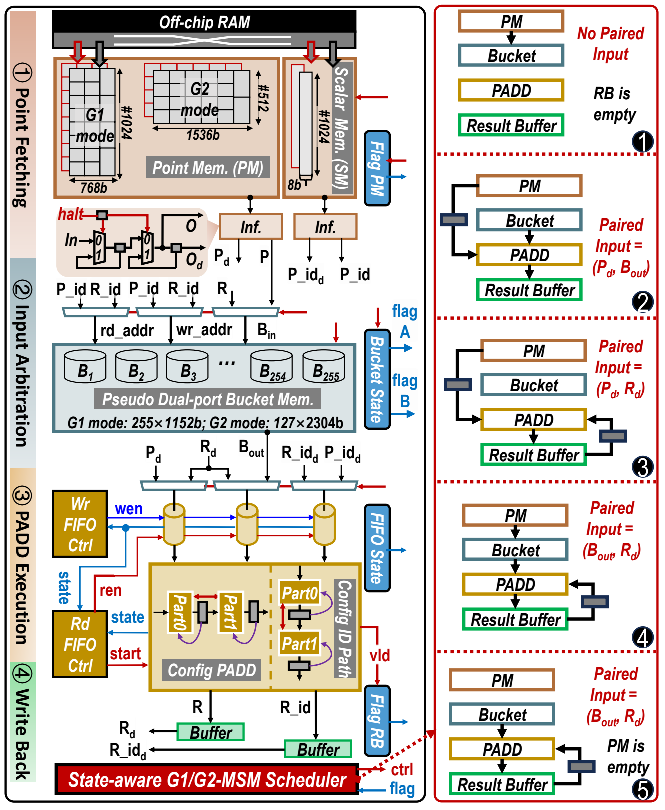


Fig. 17. The unified architecture for G1/G2-MSM.

Since the inputs of point adder (PADD) are unpredictably from PM, bucket and result buffer (RB), a state-aware scheduler is devised to dynamically handle the colliding points and improve resource utilization. The colliding Point refers to a situation where all three operands from PM, bucket, and RB are available, but the input ports of PADD can only accept two operands. Therefore, one operand must be cached, and the data source for that operand must be stalled. The scheduling method of the prior work [23] only supports fully-pipelined G1-PADD and cannot handle G2-PADD with cascaded loops. This is because G1-PADD processes one data pair per cycle, while G2-PADD requires multiple cycles. To address this issue, we insert FIFOs into the PADD inputs, thereby decoupling the input arbitration and PADD execution stages. This allows operands to be stored in the FIFO when the PADD is not ready, ensuring the correct synchronization and achieving near 100% hardware utilization. Based on the real-time flags, 5 cases are proposed to cover all arbitration scenarios for PADD and bucket. For example, case ① occurs at the initial phase when the first result of PADD is yet obtained. If index P_id == R_id, the points from PM and RB (i.e., P_d, R_d) are paired as inputs of PADD (case ③). Otherwise, we resort to other cases.

As shown in Fig. 18, based on our flexible MM proposed in section III, the PADD can be configured as G1-PADD of fully pipelined path or G2-PADD of cascaded loop without adding extra MMs. We apply the hybrid handshake and pipeline mechanism to synchronize each local loop, where the valid result of Part0 must be sent to Part1 with ready state, yielding $\sim$100% utilization for G2-MSM. For the bucket reduction phase, we decouple the serial addition chain into two independent segmented ones by index transformation as shown in Fig 19, which aims to fill up the pipelined datapath of PADD and enhance the utilization by $\sim 2\times$.

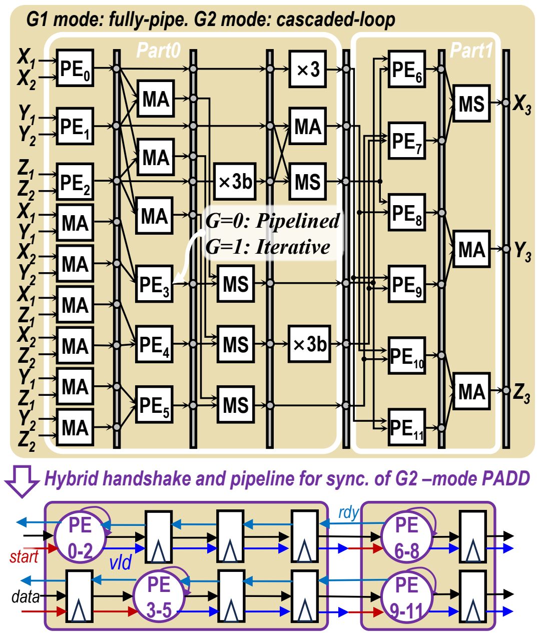


Fig. 18. The unified G1/G2-mode PADD

$$\begin{aligned} G &= \textstyle\sum_{i=1}^{15} i \cdot B_i = \boxed{B_1 + 2B_2 + \cdots + 15B_{15}} \\ &= B_{15} + (B_{15} + B_{14}) + (B_{15} + B_{14} + B_{13}) \\ &\quad + \cdots + (B_{15} + B_{14} + B_{13} + \cdots + B_1) \end{aligned}$$

index transformation:
$$\boldsymbol{i = k + 8j}$$
$\Downarrow$ ***→ Enchance Util by ~2×***

$$\begin{aligned} G &= \textstyle\sum_{j=0}^{1} \sum_{k=1}^{7} (k + 8j) \cdot B_{(k+8j)} \\ &= \textstyle\sum_{j=0}^{1} \sum_{k=1}^{7} k \cdot B_{(k+8j)} + \sum_{j=0}^{1} 8j \sum_{k=1}^{7} B_{(k+8j)} \\ &= \boxed{(B_1 + 2B_2 + \cdots + 7B_7)} + \boxed{(B_8 + 2B_9 + \cdots + 7B_{15})} \\ &\quad + 8(B_9 + B_{10} + \cdots + B_{15}) \end{aligned}$$

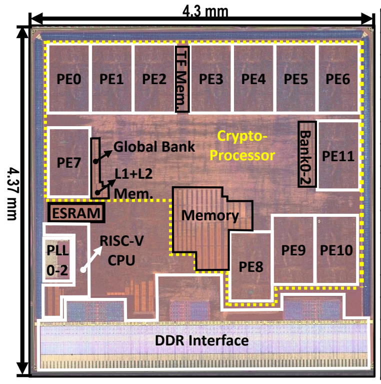
2 parallel add. chain

Fig. 19. Segmented addition chain for bucket reduction to fill up the pipelined datapath and improve hardware utilization.

V. Silicon Result and Comparison

A. Silicon Chip Result

The proposed cryptographic processor is fabricated under TSMC 28nm HPC process and adopts the BGA package. The die photography of the complete SoC system, shown in Fig 20, occupies 18.8 mm^2 and integrates the cryptographic processor, PLL, CPU, DMA, test circuits and DDR interface (including the controller and PHY). The cryptographic processor occupies 8.3 mm^2, with the physical layout of computational and memory units indicated. Due to the large area cost of each PE, they are hardened in the backend design to aid timing convergence. The 12 PEs are positioned near memory to minimize routing complexity. Unlike prior works using simulation-based off-chip memory systems, this chip includes an actual DDR interface, enabling off-chip memory interaction during on-chip computation to support large-scale applications.



Chip Specifications	
Technology	TSMC 28nm HPC
Supply Voltage	0.7~1.1V
Package	BGA
Die Size	18.8mm^2
Off-chip Memory	DDR
Cryptographic Processor	
Core Area	8.3mm^2
SRAM	275KB
Logic Gates	14.5M
Main Operators	**Coset (I)NTT, VMM/A/S, G1/G2 MSM, Pairing, Inv.**
Finite Fields	**Fp, Fp^2, Fp^6, Fp^{12}**
Applications	**ZKP, Other PBC**
Max. Freq. (MHz)	300/500/800 @0.7/0.9/1.1V
Avg. Power (W) (Pairing/NTT/MSM)	0.20/0.45/0.61 @0.7V 1.1/2.1/2.8 @1.1V

Fig. 20. The die photography and chip summary

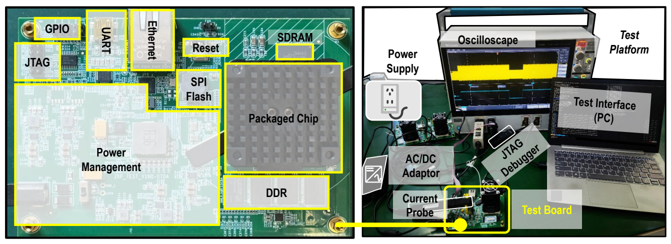


Fig. 21. The chip test environment.

The processor supports different finite fields, covering all operators for ZKP proof generation and verification, with extendability for additional PBC schemes. Pairing is implemented by reusing four large PEs, leading to negligible area overhead without impacting performance. The chip operates between 0.7V and 1.1V, with a frequency range of 300MHz to 800MHz. The critical path is located within the configurable modular multiplier. Different operators require different number of PEs, resulting in varying average power consumption. Specifically, Pairing, NTT, and MSM use 4, 8, and 12 PEs, with power consumption of 0.2W, 0.45W, and 0.61W at 0.7V, respectively. Fig 22 provides a breakdown of the chip's area and power consumption, with the 12 PEs occupying 72% of the chip area and consuming 86% of total power. PEs 8-11 are reused for Pairing's $\mathbb{F}_{p^2}$ modular multiplication, and PEs 0-7 serve as BFUs. Fig 23 also shows the variation of frequency and power with voltage for Pairing. The chip's test environment and measurement methods are presented in Fig 21, including the oscilloscope, PCB, and power supply.

B. Evaluations on Crypto-prcocessor

Fig. 24 presents the performance and area efficiency of key zkSNARK proving operators on the proposed crypto-processor. Their results are compared against software counterparts and prior hardware designs. The evaluation is performed

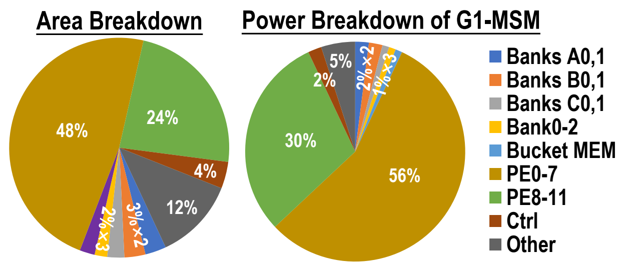


Fig. 22. The area and power breakdown analysis.

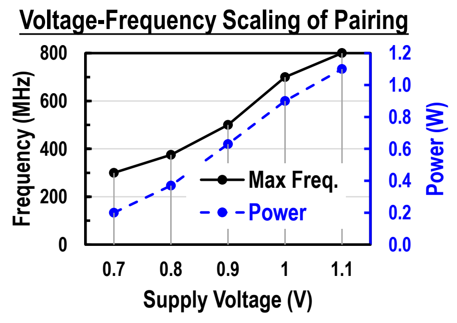


Fig. 23. The frequency and power scaling with voltage.

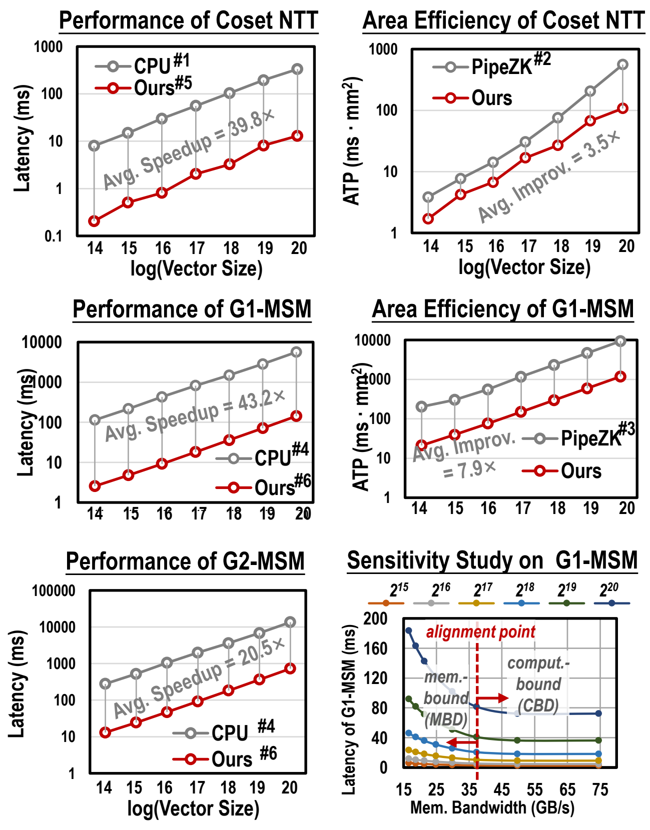


Fig. 24. Evaluations on the performance and area-efficiency across varying vector size. #1: reported by ISCA'21 [21] #2: w/o coset factor mult. latency #3: only supports bucket accumulation phase #4: reported by CHES'23 [48] #5: @0.9V, 500MHz, CBD #6: @0.9V, 500MHz, MBD

at 500 MHz, considering the practically achievable off-chip memory bandwidth (less than 38.4 GB/s). For Coset NTT/G1-MSM across different sizes, this work achieves 39.8×/43.2× average speedup versus CPU platform [21], [48], and improves the area efficiency of PipeZK [21] by 3.5×/7.9× as well. Compared to the higly optimized software implementation [21] of G2-MSM, our work obtains 20.5× speedup on average. To the best of our knowledge, there is no previous work reporting the performance of G2-MSM on the hardware processor. The impact of memory bandwidth (MBW) on the latency of G1-MSM is evaluated across various vector sizes in Fig 24, which guides us to adjust the off-chip memory and DMA configurations. Fig. 25 further evaluates the latency and energy consumption of proof generation and verification across three application scales. The comparisons are between the proposed hardware implementation and its software counterpart using the popular and open-source libraries. The proof generation for typical three applications completes within one second on the hardware crypto-processor, which achieves over 20× speedup and two orders of magnitude energy savings compared to CPU. For example, consider the 2^{16}-gate proof generation and verification of a Merkle Tree using Groth16. In this case, our processor exhibits 44× and 13× lower latency, respectively, compared with a 14-core CPU. Moreover, it achieves 768× and 909× reductions in energy consumption for proof generation and verification, respectively.

C. Comparisons with Prior Works

Comparison of Pairing operators. Table II lists recent works implementing Pairing on the IETF-recommended BLS12-381 curve across AISC, FPGA and software platforms. [5] presents the first silicon-validated Pairing hardware processor for the BLS12-381 curve, designed for IoT applications with minor area and power consumption. Its hardware datapath utilizes a CIOS-based Montgomery modular multiplier, which needs small area overhead but incurs significant clock cycle consumption. The design adopts a manually serialized timing schedule characterized by low parallelism and simplified control logic. Despite this limitation, it demonstrates remarkable scalability. In particular, it supports 256-bit modular operations on the Jubjub curve and the Hash-to-Curve functionality. Under the normalization method from [49], our design outperforms the prior work by 17× in ATP and 1.8× in energy efficiency. The observed outcome is likely a result of automatic optimization enabled by a customized Pairing compiler. Such optimization reduces clock cycle overhead, thereby ensuring high overall hardware utilization. The significantly reduced latency of our processor makes the incurred area and power penalties acceptable. We also evaluate our design against another two recent works: a high-throughput 16nm FPGA implementation [3] and a highly optimized software counterpart on the 14nm 3.4GHz CPU platform [50]. In terms of latency, our design achieves speedups of 6.4× and 9.6×, respectively. Furthermore, our Pairing processor provides high programmability and supports agile development. This inherent flexibility enables it to adapt to a wide range of ECC-related functions and operations over tower extension fields, positioning it as a promising candidate for various PBC applications.

Comparison of accelerator designs. Table III lists recent works on accelerating Pairing-based zkSNARKs, including

TABLE II
THE COMPARISON TABLE ABOUT THE PAIRING OPERATOR.

Table 1[#1]	Process	Platform	Voltage	Freq.	Cycle	Latency	Energy	Area	ATP	Curve	Optimize	Extension
This Work	28nm	ASIC (Tapeout)	1.1V	800MHz	55.9K	**69.9μs**	**76.9μJ**	2.09[#4] mm^2	**146.1** μs·mm^2	BLS12-381	**Automatic**	HtC[#5], Jubjub
SSCL'21 [5]	40nm	ASIC (Tapeout)	1.1V	90MHz	3393K	37.7[#2]/ 26.4[#3] ms	276.2 / 135.4μJ[#3]	0.098[#3] mm^2	2586.2[#3] μs·mm^2	BLS12-381	Manual	HtC, Jubjub
TVLSI'24 [3]	FinFET+ 16nm	XCVU9P FPGA	—	458MHz	207.1K	452 μs	—	—	—	BLS12-381	Manual	—
CHES'23 [50]	14nm	i7-6700 CPU	—	3.4GHz	2290K	673.5 μs	—	—	—	BLS12-381	—	All (Software)

***#1:** comparison of Pairing operator* ***#2:** from thesis (Miller Loop + Final Exponent.)* ***#3:** normalized to 28nm [49]* ***#4:** partial area of entire processor* ***#5:**HtC: Hash-to-Curve*

TABLE III
THE COMPARISON TABLE ABOUT THE ENTIRE ACCELERATOR.

		ISCA'21 [21]	ASPLOS'23 [51]	ISSCC'25 [52]	HPCA'25 [4]	CHES'25 [23]	This Work
Process		28nm	16nm	28nm	28nm	28nm	28nm
Platform		ASIC (Synthesis)	V100 GPU	ASIC(Tapeout)	ASIC (Synthesis)	ASIC (Synthesis)	ASIC(**Tapeout**)
Core Area (mm^2)		49.3[#1]	815 (Die)	5.4	53.89[#2]	11.31	**8.3**
Voltage (V)		0.9	—	0.7~1.1	0.9	0.9	0.7~1.1
Frequency (Hz)		300M	1.23G	125~625M	1G	800M	300~800M
Application		ZKP	All (Software)	FHE	ZKP	ZKP	**ZKP, Other PBC**
Supported Operators	**Coset (I)NTT**	✓	✓	✓	✓	✓	✓
	G1-MSM	✓	✓	×	✓	✓	✓
	G2-MSM	×	✓	×	✓	×	✓
	Pairing	×	×	×	×	×	✓
	VMM/A/S	×	✓	✓	✓	×	✓
Programmability		×	✓	✓	✓	×	✓
Off-chip Memory		DDR	HBM2	NA	HBM2e	**2** DDR Ctrl	**1** DDR Ctrl
Coset NTT (N=2^{20})	**Complexity (#MM)**	N+N/2logN	N+N/2logN	NA	N+N/2logN	N+N/2logN	**N/2logN**
	Latency[#3] (ms)	11	1.07	NA	2.5[#4]	0.91	8.1[#9,+]
	Area Effi. (Gbps/mm^2)[#11]	0.46	0.28	37.62[#5]	1.9	24.1	**3.7**[#9,+]
	Energy Effi. (Gb/J)	16.7	3.05[#6]	NA	9.8[#8]	NA	**26.3**[#10,+]
G1/G2-MSM (N=2^{20})	**Latency (ms)**	184[#7]/NA	62/NA	NA/NA	50[#4]/NA	72.9/NA	(89.2/454.9)[#9,+]
	Area Effi. (Gbps/mm^2)	0.11/NA	0.02/NA	NA/NA	0.37/NA	1.21/NA	(**1.35/0.46**)[#9,+]
	Energy Effi. (Gb/J)	1.14/NA	0.05[#6]/NA	NA/NA	1.96[#8]/NA	NA/NA	(**7.1/2.4**)[#10,+]

***#1:** (I)NTT+G1-MSM* ***#2:** w/o HBM2e PHY* ***#3:** w/o coset factor mult.* ***#4:** @460GB/s MBW, absolute number is not given, evaluated from the figure.* ***#5:** N=4096, bits=32b, w/o DRAM access* ***#6:**TPD=300W* ***#7:** only bucket accumulation phase* ***#8:** power=10.16W (w/o HBM2e PHY)* ***#9:** @1.1V, 800MHz, MBW<38.4GB/s* ***#10:** @0.7V, 300MHz* ***+**: NTT is CBD, MSM is MBD.* ***#11:** Gbps=N×bits/latency; bits=256b for NTT, (256b+width×2) for G1-MSM , (256b+width×4) for G2-MSM; width=384b for BLS12-381*

ASIC synthesis and GPU platforms. Pipezk [21] represents the first simulation-based accelerator for zkSNARKs; however, it supports only NTT and G1-MSM. While each operator achieves over a 10x speedup compared to software implementations, the overall end-to-end proof generation experiences a modest acceleration due to incomplete operator coverage (Amdahl's Law). GZKP [51] is a GPU-based acceleration work for zkSNARKs, leveraging a large number of computing units and expensive HBM, yielding significant improvements in latency. However, the high energy consumption and large area overhead of the GPU platform result in substantial deployment costs, making it less suitable for widespread applications. The work [52] introduces a multi-scheme accelerator targeting FHE. The NTT implementation is limited to the small-scale and narrow-bit-width scenario, which also takes no account of the practical DRAM access latency. Legozk [4] is a recent synthesis-based work that claims to support complete proof generation. It employs a NoC to interconnect multiple configurable computational units. This architecture enables the reconstruction of various high-level operations. Despite its flexibility, the design still incurs significant core area and power consumption, ignoring the optimizations for low-level large-width modular multipliers. Additionally, the use of expensive HBM without bandwidth alignment analysis may lead to considerable bandwidth waste [53]. Recent work [23] proposes a unified hardware accelerator to improve the area efficiency. However, it only supports NTT and G1-MSM operations. Furthermore, the design utilizes two DDR interfaces to fully hide off-chip memory latency for the 2D NTT. This results in considerable area overhead, as the layout

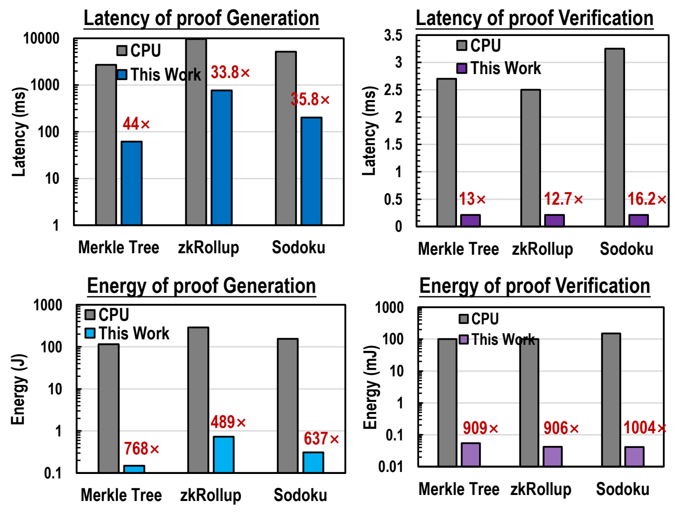


Fig. 25. End-to-end evaluations on the latency and energy consumption of full applications, including the practical DRAM access latency. The number of R1CS gate for Merkle Tree, zkRollup and Sodoku is 2^{16}, 2^{20} and 2^{17}, respectively. The latency and energy of accelerator is measured at (800MHz, 1.1V) and (300MHz, 0.7V), respectively. The software measurement condition: @2.4GHz, 14-core CPU, Library: Arkwork [43].

confirms the significant area consumption by DDR interfaces. Compared to other ZKP accelerators, our first silicon-proven design consumes the lowest area cost ($8.3\,\text{mm}^2$) to cover all related operators (including G2-MSM, Pairing, etc.) and acquires adaptivity to other PBC. It operates at 300–800 MHz across 0.7–1.1 V. For 2^{20}-size Coset NTT/G1-MSM, our processor offers 8.8×/12.2× higher area efficiency (@1.1 V) and 1.7×/6.2× higher energy efficiency (@0.7 V) relative to [21] under similar and practical DRAM settings.

VI. Conclusion

This paper presents the first silicon-verified 28nm cryptographic processor for hybrid ZKP–PBC applications that is scheme-adaptive and cost-efficient. It supports the complete proof generation and verification phases, while also being adaptable to other PBC schemes. Existing works often prioritize latency reduction at the expense of flexibility, incurring considerable area overhead and high deployment costs. Many of these implementations either rely on separated datapaths or neglect low-level optimization of large-width modular arithmetic, leading to poor area efficiency and hardware utilization. In contrast, the proposed processor maintains flexibility while minimizing area, satisfying practical latency requirement [54] for target applications. It supports all operators required in proof generation and verification, reducing area and bandwidth consumption by 6.4× and 10×, respectively. In a case study involving 2^{16}-gate proof generation and verification, the processor consumes 0.15J and 0.11mJ, respectively, improving energy efficiency by two orders of magnitude over a 14-core 2.4GHz CPU. For the Pairing operator, it achieves development agility to enable fast and efficient prototype of various ECC-related functions. It offers 17× higher ATP and 1.8× better energy efficiency compared to prior ASIC implementations [5]. This work contributes to ongoing efforts to enhance the efficiency of ZKP-PBC schemes, marking a significant step towards making the accelerator more efficient for real-world applications.

References

[1] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[2] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019.

[3] Junichi Sakamoto, Daisuke Fujimoto, Riku Anzai, Naoki Yoshida, and Tsutomu Matsumoto. High-throughput bilinear pairing processor for server-side fpga applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 32(8):1498–1511, 2024.

[4] Zhengbang Yang, Lutan Zhao, Peinan Li, Han Liu, Kai Li, Boyan Zhao, Dan Meng, and Rui Hou. Legozk: A dynamically reconfigurable accelerator for zero-knowledge proof. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 113–126, 2025.

[5] Utsav Banerjee and Anantha P. Chandrakasan. A low-power bls12-381 pairing cryptoprocessor for internet-of-things security applications. *IEEE Solid-State Circuits Letters*, 4:190–193, 2021.

[6] Leon Visscher, Mohammed Alghazwi, Dimka Karastoyanova, and Fatih Turkmen. Poster: Privacy-preserving genome analysis using verifiable off-chain computation. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 3475–3477. ACM, 2022.

[7] Craig Gentry. A fully homomorphic encryption scheme. *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 169–178, 2009.

[8] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. Secure multiparty computation. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 265–276. ACM, 1989.

[9] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. Proofs that yield nothing but their validity. *Journal of the ACM*, 32(4):891–939, 1985.

[10] Bineet Kumar Joshi, Bansidhar Joshi, Anupama Mishra, Varsha Arya, Avadhesh Kumar Gupta, and Dragan Perakovic. A comparative study of privacy-preserving homomorphic encryption techniques in cloud computing. *Int. J. Cloud Appl. Comput.*, 12(1):1–11, 2022.

[11] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.

[12] Eli Ben-Sasson, Alberto Chiesa, Daniel Genkin, Amos Lerer, Michael Luby, Osvald Raza, and Eran Tromer. Scalable transparent arguments of knowledge. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 120–132. ACM, 2018.

[13] Eli Ben-Sasson, Alberto Chiesa, Amos Lerer, Michael Luby, Osvald Raza, and Eran Tromer. Orion: Transparent zero-knowledge proofs with optimal prover complexity. In *Proceedings of the 2021 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 170–185. IEEE, 2021.

[14] Eli Ben-Sasson, Alberto Chiesa, Daniel Genkin, Amos Lerer, Michael Luby, and Eran Tromer. Halo 2: Recursive proofs and succinct arguments without a trusted setup. *Cryptology ePrint Archive*, (2020/137), 2020.

[15] Sushmita Ruj. Zero-knowledge proofs for blockchains. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, pages 67–68, 2024.

[16] Tom Massey, Todd Oscherwitz, and Jan Jürjens. Zcash: A privacy-preserving digital currency. *Proceedings of the 2018 International Conference on Financial Cryptography and Data Security (FC)*, pages 14–27, 2018.

[17] Vitalik Buterin. A next-generation smart contract and decentralized application platform. *Ethereum Whitepaper*, 2014.

[18] Payman Mohassel, Xin Zhang, Pascal Rindal, and Reza Shokri. Secureml: A system for scalable privacy-preserving machine learning. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–34. IEEE, 2017.

[19] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 863–874, New York, NY, USA, 2013. Association for Computing Machinery.

[20] David Chaum and Josh Benaloh. The theory of broadcast encryption and group signatures. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 64–75. IEEE, 1988.

[21] Ye Zhang, Shuo Wang, Xian Zhang, Jiangbin Dong, Xingzhong Mao, Fan Long, Cong Wang, Dong Zhou, Mingyu Gao, and Guangyu Sun. Pipezk: Accelerating zero-knowledge proof with a pipelined architecture. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Virtual Event / Valencia, Spain, June 14-18, 2021*, pages 416–428. IEEE, 2021.

[22] Andy Ray, Benjamin Devlin, Fu Yong Quah, and Rahul Yesantharao. Hardcaml msm: A high-performance split cpu-fpga multi-scalar multiplication engine. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '24, page 33–39, New York, NY, USA, 2024. Association for Computing Machinery.

[23] Xiangren Chen, Bohan Yang, Wenping Zhu, Hanning Wang, Qichao Tao, Shuying Yin, Min Zhu, Shaojun Wei, and Leibo Liu. A high-performance ntt/msm accelerator for zero-knowledge proof using load-balanced fully-pipelined montgomery multiplier. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):275–313, Dec. 2024.

[24] D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini. The realm of the pairings. In *Selected Areas in Cryptography—SAC*, volume 8282 of *Lecture Notes in Computer Science*, pages 3–25. Springer, 2013.

[25] Tianwei Pan, Tianao Dai, Jianlei Yang, Hongbin Jing, Yang Su, Zeyu Hao, Xiaotao Jia, Chunming Hu, and Weisheng Zhao. Finesse: An agile design framework for pairing-based cryptography via software/hardware co-design. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, page 65–77, New York, NY, USA, 2025. Association for Computing Machinery.

[26] Makoto Ikeda. Hardware acceleration of elliptic-curve based crypto-algorithm, ECDSA and pairing engines. In Fan Ye and Ting-Ao Tang, editors, *14th IEEE International Conference on ASIC, ASICON 2021, Kunming, China, October 26-29, 2021*, pages 1–4. IEEE, 2021.

[27] Arthur Lavice, Nadia El Mrabet, Alexandre Berzati, Jean-Baptiste Rigaud, and Julien Proy. Hardware implementations of pairings at updated security levels. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 189–209. Springer, 2021.

[28] Ahmad Salman, William Diehl, and Jens-Peter Kaps. A light-weight hardware/software co-design for pairing-based cryptography with low power and energy consumption. In *International Conference on Field Programmable Technology, FPT 2017, Melbourne, Australia, December 11-13, 2017*, pages 235–238. IEEE, 2017.

[29] Youssef El Housni and Aurore Guillevic. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. In Stephan Krenn, Haya Schulmann, and Serge Vaudenay, editors, *Cryptology and Network Security - 19th International Conference, CANS 2020, Vienna, Austria, December 14-16, 2020, Proceedings*, volume 12579 of *Lecture Notes in Computer Science*, pages 259–279. Springer, 2020.

[30] Christian Delgado-von-Eitzen, Manuel J. Fernández-Iglesias, Luis E. Anido-Rifón, and Fernando A. Mikic-Fonte. Blockchain beyond immutability: Application firewalls on ethereum-based platforms. *Comput. Stand. Interfaces*, 95:104038, 2026.

[31] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 1777–1793. IEEE, 2024.

[32] Gautam Botrel and Youssef El Housni. Faster montgomery multiplication and multi-scalar-multiplication for snarks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):504–521, 2023.

[33] Tao Lu, Chengkun Wei, Ruijing Yu, Chaochao Chen, Wenjing Fang, Lei Wang, Zeke Wang, and Wenzhi Chen. cuzk: Accelerating zero-knowledge proof with A faster parallel multi-scalar multiplication algorithm on GPUs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):194–220, 2023.

[34] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2001)*, pages 514–532. Springer, 2001.

[35] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference (CRYPTO 2001)*, pages 213–229. Springer, 2001.

[36] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Proceedings of the 11th International Conference on Theory of Cryptography (TCC 2011)*, pages 253–273. Springer, 2011.

[37] Nuttapong Attrapadung and Brent Waters. Functional encryption for inner product functionality. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS 2013)*, pages 1295–1306. ACM, 2013.

[38] Yangyang Bao, Lingrui Pan, Xiaochun Cheng, and Liming Nie. Enabling privacy-preserving and distributed intelligent credit scoring by zero-knowledge proof and functional encryption. *Peer Peer Netw. Appl.*, 18(3):138, 2025.

[39] Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu. Highly efficient architecture of newhope-nist on FPGA using low-complexity NTT/INTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):49–72, 2020.

[40] SCIPR LAB. libsnark: a c++ library for zksnark proofs, 2022.

[41] ZKCrypto Corp. bellman: a crate for building zk-snark circuits, 2022.

[42] https://pkg.go.dev/github.com/consensys/gnark. Accessed: 2025-10-19.

[43] https://github.com/arkworks-rs. Accessed: 2025-10-19.

[44] Xiangren Chen, Bohan Yang, Shouyi Yin, Shaojun Wei, and Leibo Liu. CFNTT: scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):94–126, 2022.

[45] Yihong Zhu, Wenping Zhu, Chongyang Li, Min Zhu, Chenchen Deng, Chen Chen, Shuying Yin, Shouyi Yin, Shaojun Wei, and Leibo Liu. Repqc: A 3.4-uj/op 48-kops post-quantum crypto-processor for multiple-mathematical problems. *IEEE J. Solid State Circuits*, 58(1):124–140, 2023.

[46] Baoze Zhao, Wenjin Huang, Tianrui Li, and Yihua Huang. Bstmsm: A high-performance fpga-based multi-scalar multiplication hardware accelerator. In *2023 International Conference on Field Programmable Technology (ICFPT)*, pages 35–43, 2023.

[47] Xander Pottier, Thomas de Ruijter, Jonas Bertels, Wouter Legiest, Michiel Van Beirendonck, and Ingrid Verbauwhede. OPTIMSM: FPGA hardware accelerator for zero-knowledge MSM. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(2):489–510, 2025.

[48] Guiwen Luo, Shihui Fu, and Guang Gong. Speeding up multi-scalar multiplication over fixed points towards efficient zksnarks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(2):358–380, 2023.

[49] Dongyun Kam, Sangbu Yun, Jeongwon Choe, Zhengya Zhang, Namyoon Lee, and Youngjoo Lee. 2.8 A 21.9ns 15.7 gbps/mm^2 (128,15) BOSS FEC decoder for 5g/6g URLLC applications. In *IEEE International Solid-State Circuits Conference, ISSCC 2024, San Francisco, CA, USA, February 18-22, 2024*, pages 50–52. IEEE, 2024.

[50] Patrick Longa. Efficient algorithms for large prime characteristic fields and their application to bilinear pairings. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):445–472, 2023.

[51] Weiliang Ma, Qian Xiong, Xuanhua Shi, Xiaosong Ma, Hai Jin, Haozhao Kuang, Mingyu Gao, Ye Zhang, Haichen Shen, and Weifang Hu. GZKP: A GPU accelerated zero-knowledge proof system. In Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift, editors, *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, pages 340–353. ACM, 2023.

[52] Sijia Lu, Wenping Zhu, Bohan Yang, Jiajun Yang, Tongwei Dai, Chen Chen, Xiangdong Han, Jinjiang Yang, Hanning Wang, Min Zhu, Shaojun Wei, Aoyang Zhang, and Leibo Liu. 17.2 a 28nm 4.05µj/encryption 8.72khmul/s reconfigurable multi-scheme fully homomorphic encryption processor for encrypted client-server computing. In *2025 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 68, pages 01–03, 2025.

[53] Marian Verhelst, Luca Benini, and Naveen Verma. How to keep pushing ml accelerator performance? know your rooflines! *IEEE Journal of Solid-State Circuits*, 60(6):1888–1905, 2025.

[54] Paige Peterson. Reducing shielded proving time in sapling. Blog post, Electric Coin Company, December 2018. https://electriccoin.co/blog/reducing-shielded-proving-time-in-sapling/.