Cryptographic Accelerators for Digital Signature Based on Ed25519

Mojtaba Bisheh-Niasar, Reza Azarderakhsh, *Member, IEEE*, and Mehran Mozaffari-Kermani, *Senior Member, IEEE*

***Abstract*— This article presents highly optimized implementations of the Ed25519 digital signature algorithm [Edwards curve digital signature algorithm (EdDSA)]. This algorithm significantly improves the execution time without sacrificing security, compared to exiting digital signature algorithms. Although EdDSA is employed in many widely used protocols, such as TLS and SSH, there appear to be extremely few hardware implementations that focus only on EdDSA. Hence, we propose two different field-programmable gate array (FPGA)-based EdDSA implementations, i.e., efficient and high-performance Ed25519 architectures applicable for a security level comparable to AES-128. Our proposed efficient Ed25519 scheme achieves an improvement of more than 84% compared to the best previous work by reducing the required area. It also incorporates more than 8× speedup. Furthermore, our proposed high-performance architecture shows a 21× speedup with more than 6200 digital signature algorithms per second, showing a significant improvement in terms of utilized area × time on a Xilinx Zynq-7020 FPGA. Finally, the effective side-channel countermeasures are embedded in our proposed designs, which also outperform the previous works.**

***Index Terms*— Ed25519, Edwards curve digital signature algorithm (EdDSA), elliptic curve cryptography, hardware implementation, side channel.**

I. Introduction

EDWARDS curve digital signature algorithm (EdDSA) developed by Bernstein *et al.* [1] has gained prominent attention among the existing digital signature algorithms due to its fast operations without affecting the required security. The Ed25519, as the most popular instance of EdDSA, is widely used as a digital signature method to guarantee the validity of the communications. On the other hand, the elliptic curve digital signature algorithm (ECDSA) is no longer suitable for embedded devices due to its vulnerability against side-channel analysis (SCA) attacks [2], [3]. Hence, most HTTPS websites are switching to Ed25519, suitable for higher level security requirements, which address some backdoor issues [4] in other ECDSA constructions at the same time.

Although most current cryptosystems will be broken by quantum computing based on Shor's algorithm [5], the transition to postquantum cryptography (PQC) includes an emerging field called hybrid systems [6], requiring both classic and PQC [7]. Hence, designing high security ECC-based digital signature for different applications is crucial. EdDSA is notable for high speed and constant-time implementations and was quickly implemented as a part of the TLS and OpenSSH protocols [8]. Hence, it has to be implemented in various platforms subject to the performance requirement of the target application, such as constrained IoT devices. However, EdDSA has not got sufficient study, especially in the field of hardware implementation based on field-programmable gate arrays (FPGAs). Therefore, investigation of the hardware implementation of this algorithm is required considering the advantages of FPGA-based designs to exploit parallelism, which leads to improvements in the efficiency of the overall system.

There are two main solutions to enable the hardware-based digital signature algorithm in the constrained IoT, including: 1) HW/SW approach to cope with embedded constraints and 2) pure HW method that includes all in hardware instructions. The HW/SW method makes the design smaller, slower, and more controllable/programmable compared to pure HW schemes. Although the pure HW approach leads to better performance, HW/SW can be a better choice for IoTs since it provides flexibility to switch security levels based on performance targets. In [9], the comparison of the CryptoCell API over nRF52840 as an internal HW/SW solution and the external cryptochip ATECC608A as a pure HW is thoroughly studied. Furthermore, to address higher security needs, new NIST and IETF recommendations make Curve448 suitable for higher level security requirements [10], [11]. Hence, implementing HW/SW architecture brings the required flexibility among different security levels, while a general architecture can be implemented in HW and controlled by instruction set processors such that the hardware remains flexible to a great extent, which is beyond the scope of this work.

A. Related Work

As one of the first FPGA-based works in ECC-based digital signature, Glas *et al.* [12] proposed architecture for 128-bit security to integrate into a vehicle-to-vehicle communication system. Furthermore, Panjwani [13] presented a scalable hardware implementation in prime fields over NIST recommended field sizes up to 521 bit, employing hardware–software codesign approach. The work of Vliegen *et al.* [14] introduced a compact core over the NIST P-256 curve resis-

Manuscript received December 30, 2020; revised April 1, 2021; accepted May 1, 2021. Date of publication May 20, 2021; date of current version June 29, 2021. This work was supported in part by the National Institute of Standards and Technology (NIST) under Grant 60NANB16D246, in part by NSF under Grant 1801341, and in part by the Army Research Office (ARO) under Grant W911NF-17-1-0311. *(Corresponding author: Mojtaba Bisheh-Niasar.)*

Mojtaba Bisheh-Niasar and Reza Azarderakhsh are with the Department of Computer & Electrical Engineering and Computer Science (CEECS), Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: mbishehniasa2019@fau.edu; razarderakhsh@fau.edu).

Mehran Mozaffari-Kermani is with the Department of Computer Science and Engineering (CSE), University of South Florida, Tampa, FL 33620 USA (e-mail: mehran2@usf.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3077885.

Digital Object Identifier 10.1109/TVLSI.2021.3077885

tant against simple power analysis (SPA) attacks. Moreover, Zhang and Bai [15] proposed a core with a security level 128 bit over the SM2 curve.

Recently, a number of hardware implementations have been introduced to implement an elliptic curve point multiplication (ECPM) core over Curve25519. Sasdrich and Güneysu [16] proposed the first Curve25519 implementation using a DSP-based single-core architecture. This work has been extended by adding side-channel countermeasures in [17] and [18] to provide an evaluation against common physical attacks. In [19], fast and compact implementations of ECPM were proposed. This architecture employs a semisystolic bit-serial multiplier and carry-compact addition to provide a high-performance architecture. The work of Koppermann *et al.* [20], [21] presented a high-speed prime field multiplier with a latency of 92 μs for a point multiplication. In addition, in [22], a low-latency ECPM was proposed employing a pipelined arithmetic architecture on FPGA and ASIC platforms. It should be noted that FPGA implementations of Curve25519 in the literature cannot be directly compared to ours because the ECPM core in EdDSA occupies more resources for implementing hash core and module L reduction. Furthermore, it requires more time for a point multiplication since this architecture is reused for nonmodular multiplication and module L reduction.

A non-DSP-based Ed25519 point multiplication core was presented by Mehrabi and Doche [23] using the double-and-add algorithm. Hence, this architecture is a nonconstant-time core vulnerable to SPA attacks. Notably, the reported area does not include all the required modules for providing a digital signature, such as hash function and modulus L reduction. We explore that SHA-512 increases almost 25% utilized area in Ed25519. Moreover, Turan and Verbauwhede [24] proposed an Ed25519 architecture combined with the X25519 key exchange. This design targets resource-constrained devices on a Zynq SoC. Turan and Verbauwhede [24, Sec. 3.3] claimed that the cost of computing using restricted-X coordinates of a point on the Montgomery curve is more than extended coordinates on the twisted Edwards curve due to the complexity of coordinate conversion. Therefore, the core works over the twisted Edwards curve. Besides, although side-channel countermeasures are considered for the ECPM core, the authors do not include a resistant SHA-512 core, allowing vulnerability against SCA, as shown in [25].

Based on the aforementioned discussions, the tradeoff explorations between resource utilization and performance to implement an efficient Ed25519 implementation from different optimization perspectives have not been thoroughly studied. Particularly, designing a unified architecture consisting of physical protection against SCA in all submodules to perform secure key generation, signature generation, and signature verification is required. Besides, employing the fast and efficient Karatsuba-based multiplier for designing a high-performance Ed25519 architecture should be investigated. Eventually, the signature computation cost over the Edwards domain compared to the Montgomery domain for a highly parallel design should be investigated.

B. Contributions

To the best of our knowledge, there appear to be very few hardware implementations that focus only on Ed25519 and make the best of all its features. In this work, we present two different architectures, i.e., efficient and high-performance design of Ed25519 implementation considering different performance levels for time-constrained and area-constrained applications.

Our contributions to this work are listed as follows.

1) We propose a new approach for implementing the EdDSA accelerator on FPGA. We analyze the computation of the restricted-X coordinates of a point on the Montgomery curve with additional coordinate conversion and design two novel, highly parallel hardware architectures based on these algorithms. In this article, we show how to leverage the advantages of computation over the Montgomery curve while implementing Ed25519 accelerator circuits so that the true benefits of the accelerator circuits can be achieved.
2) We explore the tradeoffs of area and performance to accomplish different optimization perspectives. We demonstrate various optimization techniques in order to achieve an overall optimization in terms of efficiency, including the parallelization, resource sharing, redundant number presentation, adoption of distributed RAM and ROM blocks, and interleaved architecture, which achieves above 84% efficiency improvement of the area–time product compared to the leading FPGA implementations.
3) We instantiate the proposed architecture in a Xilinx Zynq-7020 FPGA and provide performance evaluations. The effective countermeasures against SCA are embedded to enhance the resistance of the proposed architectures against timing, SPA, and differential power analysis (DPA) attacks.

The remainder of this article is organized as follows. Section II presents the background. Section III conducts our proposed architectures. The experimental results and comparison are given in Section IV. We conclude this article in Section V.

II. Preliminaries

A. Background

A point $\mathcal{P} = (x, y)$ lies on a twisted Edwards curve E if $E = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : ax^2 + y^2 = 1 + dx^2y^2\}$. The Ed25519 is a type of Schnorr's signature employing (twisted) Edwards curves developed by Bernstein *et al.* [1]. Ed25519 includes three different phases, i.e., key generation, signing, and verifying. In the key generation, KeyGen(s) takes a parameter s and computes a signing key sk and a public key pk with associated message space $\mathcal{M}$. In signing, a signature (R, S) is generated by Sign(sk, m), taking an sk and a message $m \in \mathcal{M}$. The signature (R, S) can be verified by Verify(pk, m, R, S) considering the public key pk and message $m \in \mathcal{M}$. The Appendix gives these algorithms. For details, we refer interested readers to [26]. Moreover, Ed25519 is equivalent to a Montgomery curve called Curve25519, introduced by Bernstein [27] in 2006.

For group arithmetic based on Ed25519, the computation can be performed on extended homogeneous coordinates [1], [26]. A mapping between affine coordinates (x, y) and extended coordinates (X, Y, Z, T) for a point $\mathcal{P}$ is defined by $x = X/Z$, $y = Y/Z$, and $x \times y = T/Z$.

Let $\mathcal{P}_1 = (X_1, Y_1, Z_1, T_1)$ and $\mathcal{P}_2 = (X_2, Y_2, Z_2, T_2)$; $\mathcal{P}_3 = \mathcal{P}_1 + \mathcal{P}_2$ can be computed using the following formula:

$$\begin{aligned} &A = (Y_1 - X_1) \cdot (Y_2 - X_2), \quad B = (Y_1 + X_1) \cdot (Y_2 + X_2) \\ &C = 2d \cdot T_1 \cdot T_2, \; D = 2Z_1 \cdot Z_2, \quad E = B - A \\ &F = D - C, \quad G = D + C, \; H = B + A \\ &X_3 = E \cdot F, \quad Y_3 = G \cdot H, \; T_3 = E \cdot H, \; Z_3 = F \cdot G. \end{aligned} \tag{1}$$

Hisil *et al.* [28] introduced an efficient unified point addition and a dedicated point doubling formula. Hamburg [29] suggested a method for mixed readdition using extended coordinates. However, an efficient computation can be performed using the restricted-X coordinate on the Montgomery curve. In addition, the Y-coordinate result is required to recover, proposed by Okeya and Sakurai [30]. Eventually, the achieved point should be mapped to twisted Edwards space.

B. Side-Channel Protection

Although both EdDSA and ECDSA rely on an ephemeral and secret random number to sign a message, generating this random number is not determined in the ECDSA procedure. Hence, the security of ECDSA is based on the quality of random number generators (RNGs) and how to implement them securely. Nevertheless, EdDSA employs a hash function to generate a random number in a secretly deterministic way.

ECDSA vulnerability against SCA has been shown in several research works [2], [3]. Recently, Aranha *et al.* [31] show breaking ECDSA exploiting even less than one-bit leakage against 192- and 160-bit elliptic curves. Several countermeasures, including Z-coordinate randomization and constant-time implementation of group law, are suggested to avoid these vulnerabilities [31].

Constant-time and secret-independent computations are popular countermeasures against timing and SPA attacks, respectively. Simple point randomization [32] provides protection against DPA attack using a random value, whereas the scalar multiplication output is not changed. Let $\mathcal{B} = (X, Y, Z)$ be the base point presentation in projective coordinates and $\lambda \in \mathbb{Z}_p \backslash \{0\}$ be a random number. The base point can be altered such that $\mathcal{B}_r = (\lambda \cdot X, \lambda \cdot Y, \lambda \cdot Z) = (\lambda \cdot X, \lambda \cdot Y, \lambda)$, which yields to different point representations, due to the fact that $x_{\mathcal{B}} = (X/Z) = ((\lambda \cdot X)/(\lambda \cdot Z)) \bmod p$ and $y_{\mathcal{B}} = (Y/Z) = ((\lambda \cdot Y)/(\lambda \cdot Z)) \bmod p$.

A continuous point randomization approach can be applied to the projective coordinate representation of points after each iteration of the Montgomery ladder. This approach was implemented in a research work presented in [18].

Samwel *et al.* [25] proposed an attack on Ed25519 by measuring the power consumption of approximately 4000 traces. This work also suggested a countermeasure that kills the deterministic signature properties.

III. Target Architectures for Ed25519

This article introduces two different architectures for Ed25519, i.e., high-performance and efficient schemes, and discusses their primitives to achieve the considered optimization objectives. The arithmetic multiplier unit in the high-performance scheme is derived from our previous work presented in [19].

A. Design I: High-Performance Architecture

To design a high-performance Ed25519 scheme, we need to accelerate the scalar multiplication procedure as the more time-consuming part of the signature algorithm, particularly its modular multiplication unit. Hence, we design a low latency modular multiplier followed by an interleaved reduction. In this scheme, the full width of 255-bit is implemented to minimize data transition latency and maximize parallelization within the arithmetic logic unit (ALU). Therefore, loading and storing data take only one cycle to accelerate ALU throughput. Addition/subtraction between two operands is performed in 255-bit data width in one clock cycle. Moreover, the interleaved reduction is performed at the cost of one additional cycle in a pipeline fashion.

1) Modular Multiplication: Different multiplication approaches are investigated for resource and area optimization, such as Schoolbook or Toom-3, while the Karatsuba multiplication consumes fewer resources and less time than other mentioned multipliers [22]. The Karatsuba multiplication can be performed for n-bit integer A and B such that $C = A \cdot B = (a_1\phi + a_0) \cdot (b_1\phi + b_0) = a_1b_1\phi^2 + a_0b_0 + ((a_1 + a_0) \cdot (b_1 + b_0) - a_1b_1 - a_0b_0)\phi$, where $A = (a_1\phi + a_0)$, $B = (b_1\phi + b_0)$, $\phi = 2^{(n/2)}$, and $A, B, C \in \mathrm{GF}(p)$.

Hence, we implement different levels of the Karatsuba multiplication to investigate their efficiency in terms of $A \cdot T$, where A and T are the required resources and time, respectively. By applying the k-level Karatsuba multiplication, an $n \times n$-bit multiplier is broken to 3^k multipliers, while they perform an $(n/2^k) \times (n/2^k)$-bit multiplication. Therefore, the maximum level of the consecutive Karatsuba multiplication before decreasing performance can be four levels due to DSP block specifications.

Our modular arithmetic units for the proposed high-performance design are illustrated in Fig. 1. In this scheme, a 255×255-bit multiplication is decomposed to 81 16×16-bit multipliers in four consecutive levels. All partial products work in one cycle simultaneously. An addition tree is designed in a backward direction to merge the products and build the final result. The pipelined multiplier has five stages, of which three are required for the multiplication and the remaining ones for the interleaved reduction in a pipeline fashion. Hence, the full five cycles are taken only for the first multiplication, and then, a 255×255-bit multiplication computation is becoming available with a latency of only one cycle. The proposed scheduling for performing a Montgomery ladder step is depicted in Fig. 2.

2) Mod p Reduction: Employing the Karatsuba multiplication in the first level can be also used for implementing the fast modular reduction to optimize computations. This multiplication includes two main stages: breaking inputs and merging the results. Breaking stage decomposes A to a_1, a_0, and $a_1 + a_0$ (B is decomposed similar to A), and the merging stage computes addition between $C_2 = a_1b_1$, $C_0 = a_0b_0$, and $C_1 = (a_1 + a_0) \cdot (b_1 + b_0)$, where $\phi = 2^{(256/2)} = 2^{128}$ in the first level. Due to the fact that $2p \equiv 2^{256} - 38 \bmod p$, the merging stage in the first level of the Karatsuba multiplication can be used for the fast reduction such that $C = A \cdot B = (a_1 2^{128} + a_0) \cdot (b_1 2^{128} + b_0) = a_1b_1 2^{256} + a_0b_0 + ((a_1 + a_0) \cdot (b_1 + b_0) - a_1b_1 - a_0b_0)2^{128} = 38C_2 + C_0 + (C_1 - C_2 - C_0)2^{128}$.

Hence, the computed C can be presented in 387 bit. Thus, the first reduction stage optimizes the obtained result

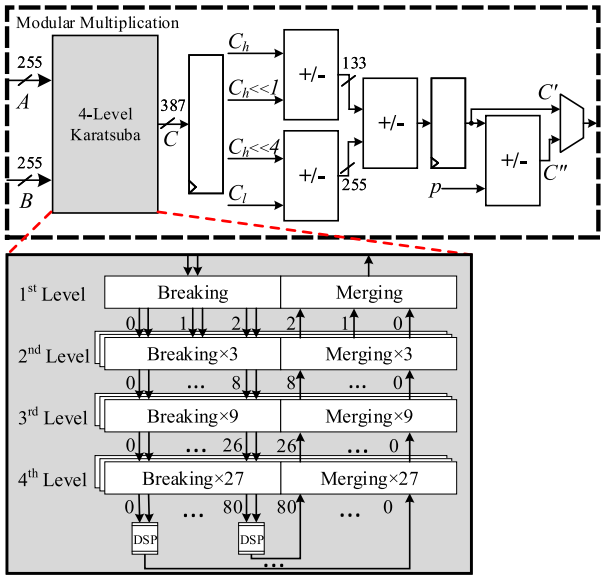

Fig. 1. Highly parallel modular multiplier in the high-performance scheme.

width from 512 to 387 bit, which increases our expected performance. Suppose that C is presented in two parts: C_l and C_h, which are its first 255-bit and rest 132 bit such that $C = C_h 2^{255} + C_l$. Therefore, the subsequent reduction stage is applied to C such that $C' = 19C_h + C_l$. In addition, $C'' = C' - p$ is computed in the case of $C' > p$, and the output is chosen between C' and C'' considering subtraction borrow flag.

B. Design II: Efficient Architecture

Fig. 3 shows the lower level arithmetic operations for our proposed efficient architecture. In this architecture, we consider decreasing the required resources as the main optimization objective, while the area–time factor is simultaneously improved. Furthermore, DSP components as the critical resource in FPGA significantly affect architecture performance. Therefore, improvement of $A_d \times T$ metrics should be considered as another vital factor to describe efficiency, where A_d is the number of employed DSPs.

In this scheme, the data width of 128-bit is implemented within ALU to decrease the CPD. However, in the modular reduction unit, redundant representation providing more 8 bits, i.e., 136-bit is implemented to avoid the cost of carry propagation between digits. Moreover, addition/subtraction between two operands, i.e., $C = A \pm B$, is performed in 128-bit data width, which takes two clock cycles. Hence, the carry is propagated between digits employing a register. Furthermore, the reduction stage performs $C' = C \mp p$ at the cost of two additional cycles. Both C and C' are stored in the memory unit, and the correct result is determined by a flag obtained from the previous carry/borrow.

1) Modular Multiplication: Modular multiplication can be computed by four 128×128-bit partial products, i.e., a_0b_0, a_0b_1, a_1b_0, and a_1b_1. Operands can be read from memory unit in a cycle to feed two input registers. Then, four multiplications are consecutively performed for these required products. For example, a_0b_0 is computed by $a_{00}b_{00}$, $a_{00}b_{01}$, $a_{01}b_{00}$, and $a_{01}b_{01}$, where $a_0 = a_{01}2^{64} + a_{00}$ and $b_0 = b_{01}2^{64} + b_{00}$. The centerpiece of the modular multiplication unit is a 64×64 pipelined schoolbook multiplier implemented by 16 DSPs.

The architecture of our proposed multiplication core is illustrated in Fig. 3. In order to accumulate the partial products, a 256-bit register and a 128-bit adder are designed. Thus, the partial product is accumulated with the upper half of the register. Furthermore, according to the sequence of multiplications, i.e., start from $a_{00}b_{00}$, then $a_{00}b_{01}$, $a_{01}b_{00}$, and eventually $a_{01}b_{01}$, the register is shifted downward by 64 bit before accumulating the second and fourth partial products. Thus, when the pipeline stages are full, 64×64-, 128×128-, and 255×255-bit multiplications are becoming available with a throughput of one, four, and 16 cycles, respectively. The proposed scheduling for performing a Montgomery ladder step is depicted in Fig. 4.

Furthermore, the field inversion is considered based on Fermat's little theorem (FLT) together with the addition chain method executing 254 squaring and 11 multiplications. We also utilize an additional dedicated ROM for performing inversion to decrease the required size in the main ROM.

2) Mod p Reduction: Modular multiplication is interleaved by a reduction unit, which accumulates partial products to perform a fast reduction. As mentioned earlier, modular multiplication performs a 255×255-bit multiplication in four sequential partial products, which takes 16 clock cycles. Thus, the modular reduction unit is fed by the multiplier every four cycles to implement a fast reduction as follows:

$$\begin{aligned} C &= A \cdot B = (a_1 2^{128} + a_0) \cdot (b_1 2^{128} + b_0) \\ &= a_1 b_1 2^{256} + a_0 b_0 + a_0 b_1 2^{128} + a_1 b_0 2^{128} \\ &= 38C_3 + C_0 + C_1 2^{128} + C_2 2^{128} \end{aligned} \tag{2}$$

where $C_3 = a_1b_1$, $C_2 = a_1b_0$, $C_1 = a_0b_1$, and $C_0 = a_0b_0$.

In order to diminish the cost of carry propagation, redundant representation is employed in the proposed reduction architecture. This unit uses several registers and adders with a 136-bit datapath providing more 8 bits for each digit. Single-pair registers, i.e., R_1 and R_2, take partial products from the multiplier, and the accumulated data are computed using the second pair, i.e., S_1 and S_2. According to the sequence of multiplications, i.e., start from C_3, then C_0, C_1, and, eventually, C_2, multiplication with a small integer $38 = (100110)_2$ is performed using the shift and addition approach in the first four-cycle period. Then, the accumulated data are stored in the second register pair to add with C_0. After that, the S-registers are shifted downward by 136-bit, and the accumulation is continued until adding the last partial product. Hence, the result represented in S-registers is computed to perform the last stage of reduction.

The accumulated result is prepared in S-registers for the last stage of reduction by applying a shift such that $C = S_2 2^{255} + S_1 2^{128} + S_0 = 19S_2 + S_1 2^{128} + S_0$. In order to have an efficient implementation, again, a multiplication with a small integer $19 = (10011)_2$ is performed using the shift and addition approach, which takes three additional cycles. According to the described scheduling, 16 cycles are required to perform these operations. Hence, the rest of the operations taking four additional cycles are stored in the T-registers that utilize R and S for new arrival data. The next two cycles are considered to accumulate $19S_2$ with $S_1 2^{128} + S_0$. Then, two cycles are required for modulus p computation.

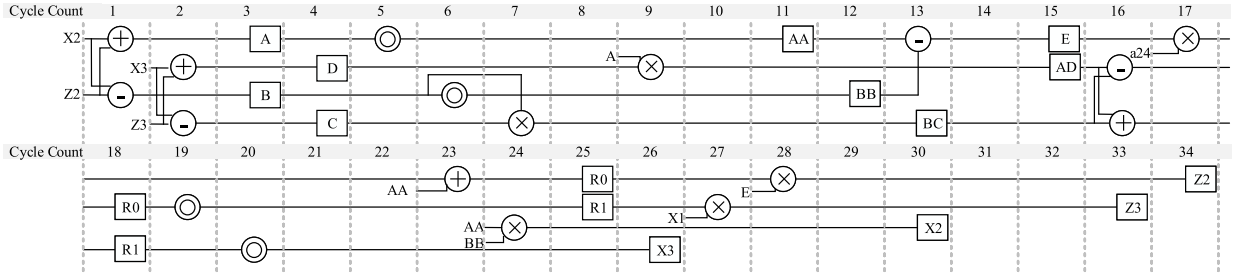

Fig. 2. Proposed Montgomery ladder scheduling in the high-performance architecture.

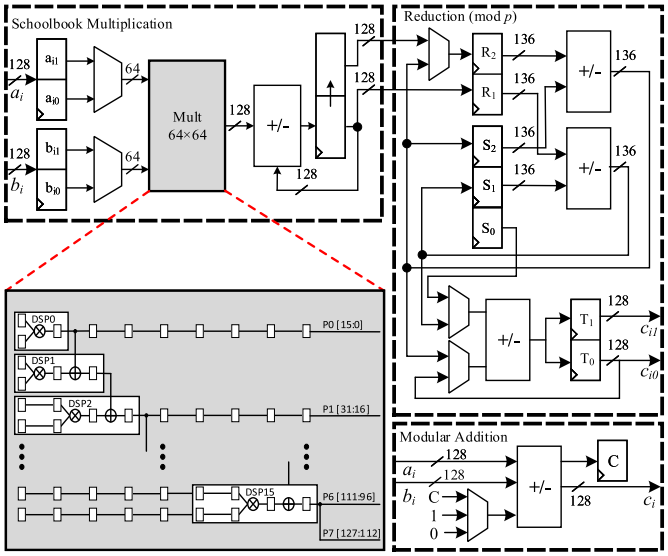

Fig. 3. Lower level arithmetic operations in the proposed fully pipelined efficient Ed25519 scheme. a_i and b_i are read from memory unit, and c_i or (c_{i1}, c_{i0}) is stored to memory unit.

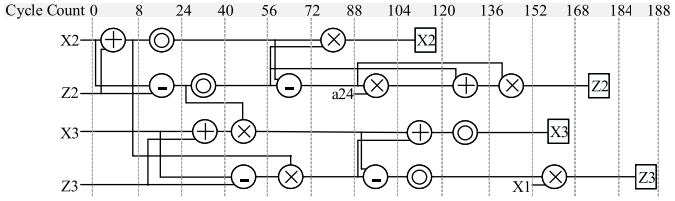

Fig. 4. Proposed Montgomery ladder scheduling in the efficient architecture.

C. Ed25519 Design Considerations

1) Hash Unit: According to RFC 8032 [26], SHA-512 is recommended by the standard to use in Ed25519. It takes arbitrary inputs in 1024-bit chunks and provides 512-bit output. In general, hash computation does not take considerable latency compared to ECPM. Therefore, lightweight hardware architecture is implemented for efficient architecture, which utilizes minimum resources.

Fig. 5 illustrates message-digest creation for N-block message. As one can see, the main part of SHA-512 is the compressor core, which works iteratively, i.e., 80 times repeated compressing for each 1024-bit chunk of input.

In order to minimize CPD, the entire data path is designed 64-bit. In addition, we use the optimal number of registers employing a dedicated finite state machine and resource sharing approach to decrease the utilized resources and complexity.

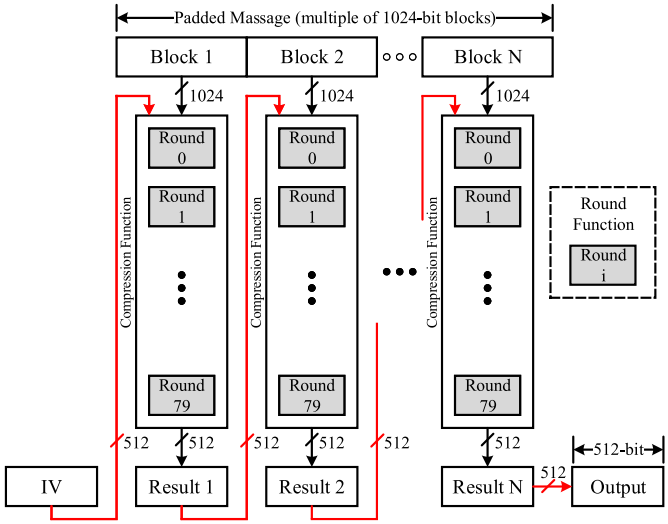

Fig. 5. Message digest creation for SHA-512.

2) Mod L Reduction and Nonmodular Multiplication: A 512-bit scalar achieved from hash function should be reduced by modulus L, where L is a 253-bit value. In order to implement a constant-time reduction, we design consecutive rounds, which is repeated three times to make sure that result is reduced completely. Let x have 512-bit length, which can be shown by $x = x_1 2^{256} + x_0$. The group order can be presented by $L = 2^{252} + l_0$, where l_0 has 125 bit. The first round of reduction is performed such that

$$\begin{aligned} x \bmod L &\equiv x_1 2^{256} + x_0 \equiv x_1 2^4 \times (L - l_0) + x_0 \\ &\equiv -x_1 \cdot l_0 2^4 + x_0. \end{aligned} \tag{3}$$

In (3), a 256 × 125-bit nonmodular multiplication should be performed, which utilizes the already provided modular multiplier. Then, the product is shifted by 4 bits and subtracted from x_0.

For the next round, let $x' = x'_1 2^{252} + x'_0$; hence, x'_1 and x'_0 have 134 and 252 bit, respectively. The reduction can be performed as follows:

$$\begin{aligned} x' \bmod L &\equiv x'_1 2^{252} + x'_0 \equiv x'_1 \times (L - l_0) + x'_0 \\ &\equiv -x'_1 \cdot l_0 + x'_0. \end{aligned} \tag{4}$$

Performing (4) results in a 260-bit-long value. Therefore, the third round must be performed similar to the second round, leading to a 253-bit-long value.

3) Double-Point Multiplication: Two scalar multiplications are required for a verification procedure. The verifying algorithm can be revised to improve efficiency, including two main

advantages: 1) employing double-point multiplication to halve total latency and 2) skipping a decompression. In addition, both scalars in the verifying algorithm are not secret. Hence, a nonconstant-time execution can be used for fast verification.

We use a modified version of Strauss' trick, a special case referred to as "Shamir's trick," presented in [33].

4) SCA Countermeasures: Different SCA countermeasures are embedded in the proposed designs to provide enhanced architecture against SPA and DPA. Designing an RNG is not in the scope of this study, so we assume that the randomized numbers are provided externally. Besides, since the scalar in the verifying procedure is not secret, the SCA countermeasures are not applied in this phase.

Each iteration of the ECPM algorithm requires one point addition and one point doubling per ladder step independent of the current key bit value. Furthermore, other executed operations are performed in a constant number of clock cycles. Therefore, considering a constant-time and secret-independent execution for designing our proposed schemes, our architectures are inherently resistant to timing and SPA attacks.

Base point randomization is achieved using the randomized base point $\mathcal{B}_r = (\lambda \cdot X, \lambda \cdot Y, \lambda)$ in projective coordinates. We assume that $\mathcal{B}_r$ is externally delivered to the ECPM core. Moreover, implementing variable-base-point architecture leads to achieving base point randomization without any cost. We can perform two more modular multiplications to rerandomize the Montgomery ladder outputs. Hence, the continuous point randomization increases the Montgomery ladder latency and, consequently, the total latency.

DPA-resistant SHA-512 can be achieved by padding the key proposed in [25]. In this method, the design requires 128 bits of fresh random for padding the key such that the first 1024-bit block is composed of the random value. It is to be noted that this algorithm is not compatible with the existing definition of EdDSA and destroys the deterministic signature properties. However, since a full arithmetic/Boolean masked architecture for SHA-512 is too costly, the future implementations might actually use SHA-3 with much robust and easier countermeasures [25].

IV. Implementation, Results, and Comparison

The FPGA used in our implementation is the Xilinx Zynq-7020 synthesized and implemented with Xilinx Vivado 2018.2. All given results are obtained post-place-and-route (PAR).

Table I shows the different implementations of the Karatsuba multiplier and the performance comparison results. Applying the Karatsuba multiplication has a significant effect on efficiency in terms of $A \cdot T$, where A and T are the utilized area and total time, respectively. According to this table, the first Karatsuba multiplication has an efficiency equal to 1166 slice $\times$ sec. Moreover, increasing the number of applied levels of the Karatsuba multiplication augments the CPD of architecture due to expanding its followed addition tree. Applying the second level to the fourth level improves 20%, 5%, and 3% efficiencies compared to its previous level, respectively. Furthermore, the four-level Karatsuba multiplication achieves a speedup factor of 3.6×, 2×, and 1.4× compared to one to three levels, respectively.

TABLE I
Implementation Results for Different Levels of Karatsuba

# of Karatsuba Level	LUTs	FFs	DSPs	Total Area[1] [Slices]	CPD [ns]	Total time [ms]	$A \cdot T$ [Slice·s]
0	3,841	3,309	24	4,624	6.5	477	2,206
1	4,049	2,157	24	4,664	4.7	250	1,166
2	7,364	3,995	36	6,531	5.1	143	934
3	9,607	4,038	54	9,051	7.8	98	887
4	14,337	4,107	81	12,285	16.6	70	860

[1]To compute the total area, 1 Xilinx DSP48 is estimated to 100 Slices following earlier work [34].

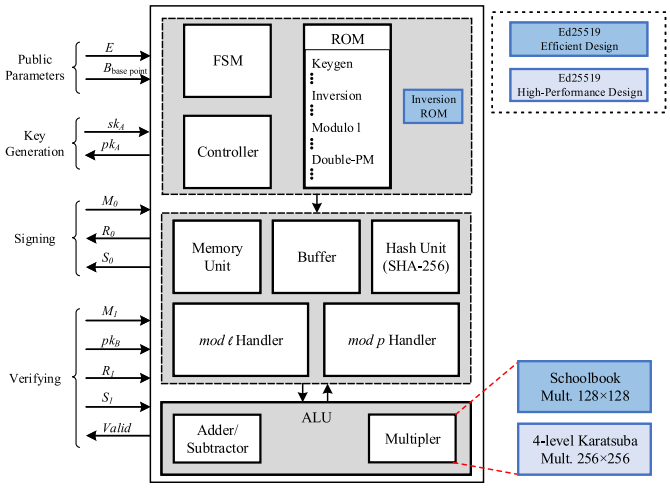

Fig. 6. Proposed architecture for high-performance and efficient Ed25519.

A. Top-Level Architecture

The top-level architecture used in our schemes is illustrated in Fig. 6, composed of three stages: 1) the top stage includes FSM, controller, and ROM; 2) the lower stage consists of the field ALU; and 3) the middle stage includes hash function, reduction handlers, memory unit, and secret key buffer.

FSM determines the state of the core and the required address number for the controller. The controller/ROM includes the main routines (fixed sequence of instructions) for point multiplication, double-point multiplication, inversion, and modulus L according to the architecture. Furthermore, the controller includes hand-optimized routines for all the operations required for computing a signature algorithm, such as enabling/disabling the modules, setting their required address, and handling their interfaces.

B. Implementation Results and Comparison

Table II summarizes the resource utilization for our proposed Ed25519 architectures broken down to the required components for our unprotected scheme. Our proposed high-performance Ed25519 architecture utilizes 9.7k slices and 81 DSPs, while the efficient Ed25519 architecture reduces 70% and 80% utilized slices and DSPs compared to our Design I, respectively. Thus, Design II requires only 2.8k slices and 16 DSPs to perform the Ed25519 signature algorithm. Both designs do not occupy any Block RAM, and they are implemented using the distributed memory.

The latency requirements of all operations are reported in Table III. Since the architecture works in a parallel fashion, the total latency is less than the summation between the latency of individual modules. To compare with the state-of-the-art modular multiplier in GF(p25519), as listed in Table IV, our

TABLE II

IMPLEMENTATION RESULTS IN TERMS OF UTILIZATION REQUIREMENTS

Architecture	**High-Performance**				**Efficient**			
	LUTs	FFs	DSPs	BRAMs	LUTs	FFs	DSPs	BRAMs
Hash Unit	2,961	2,229	0	0	3,063	2,022	0	0
ALU	9,864	3,885	81	0	2,703	2,154	16	0
Mod l Reduction	4,134	1,579	0	0	1,739	1,414	0	0
Memory Unit	7,469	8,960	0	0	937	1,777	0	0
Control Unit	10,522	69	0	0	1,023	39	0	0
Total	34,950	16,772	81	0	9,465	7,406	16	0

TABLE III

FPGA IMPLEMENTATION RESULTS IN TERMS OF CLOCK CYCLES

Architecture	**High-Performance**	**Efficient**
Addition/Subtraction	2	4
Multiplication	5	32
Inversion	2,131	10,146
Mod l Reduction	20	74
SHA-512	80	80
Mont. Ladder Step	36	190
Coordinate Trans.	154	642
Non-modular Mult.	14	100
Double-point Mult.	11,864	79,836
Unprotected ECPM	9,181	48,450
Protected ECPM	10,966	57,120

TABLE IV

IMPLEMENTATION RESULTS FOR POINT MULTIPLIER IN GF($p25519$)

Architecture	**Design I**[1]	**Design II**[1]	[19]	[22]	[20]	[21]	[24]
Area							
LUT	**9,864**	**2,703**	14,337	12,989	26,483	17,939	2,707
FF	**3,885**	**2,154**	4,107	2,705	21,107	21,077	962
DSP	**81**	**16**	81	182	260	175	15
BRAM	**0**	**0**	0.5	0	0	0	0
Time							
Mod. Mult.	**5**	**32**	3	3	10	8	33
Ladder step	**36**	**190**	13	13	42	31	250
Total cycle	**9,181**	**48,450**	4,117	3,858	13,639	10,465	63,890
Latency (μs)	**126**	**356**	70	44	118	92	608

[1]The ALU required resources are reported.

high-performance and efficient modular multiplication requires five and 32 cycles occupying 81 and 16 DSPs, respectively. However, our parallel designs can significantly compensate for the required clock cycles such that a modular multiplication can be performed in one and 16 cycles in our Designs I and II, respectively. In [22], a low-latency multiplier was proposed for key exchange requiring three cycles taking advantage of occupying register bank and 182 DSPs. We also introduced a low-latency architecture in our previous work [19] using the register bank requiring three cycles for key exchange. However, to develop the Ed25519 scheme in this article, we use the RAM module since the controller, hash, ALU, and module l reduction work with the memory unit. The designed architecture in [20] and [21] requires ten and eight cycles utilizing 260 and 175 DSPs. The proposed multiplier in [24] needs 33 cycles, of which 16 cycles for multiplications and the rest of it for the reduction utilizing 15 DSPs.

Our proposed high-performance architecture follows the reduction algorithm of [22] using one level of the Karatsuba multiplication, however applying the following modifications. First, we make use of the true dual-port capabilities of the RAM modules instead of register bank to decrease the required resources and avoid high fan-out circuits. Second, we implement four consecutive levels of the Karatsuba multiplication, enabling our design to save 55.5% of utilized DSP in [22] and, thus, still allowing processing in a pipeline fashion. Third, our architecture performs both modular and nonmodular multiplication in order to avoid any additional resources for performing signature algorithms.

TABLE V

PERFORMANCE RESULTS FOR UNPROTECTED AND PROTECTED SCHEME AGAINST DPA (RESULTS ARE REPORTED FOR A 1024-bit MESSAGE)

Scheme	**Design I** 73 MHz			**Design II** 136 MHz		
Unprotected	**Cycles**	**Time [ms]**	**OP/s**	**Cycles**	**Time [ms]**	**OP/s**
Keygen	11,631	0.16	6,276	59,670	0.44	2,279
Signing	11,776	0.16	6,293	59,585	0.44	2,293
Verifying	14,281	0.20	5,112	90,221	0.65	1,507
Protected[1]						
Keygen	13,416	0.18	5,441	68,340	0.50	1,990
Signing	13,540	0.18	5,462	68,153	0.50	2,003

[1]Both scalar S and k in the verifying operation are not secret. Hence, SCA countermeasures are not required for the verifying.

Besides, our efficient scheme can be used in 136 MHz, while the maximum operating frequency for our high-performance design is dropped as expected to 73 MHz due to the increasing level of the Karatsuba multiplication. Hence, an unprotected ECPM can be performed in almost 126 and 356 μs in our proposed high-performance and efficient architecture. Thus, our Design II provides a tradeoff between time and area by decreasing almost 75% of occupied resources at the cost of nearly three times more required time.

Table V reports the performance results for three algorithms in EdDSA: key generation, signing, and verifying. For the unprotected scheme, Designs I and II can generate 6276 and 2279 keys/s. Furthermore, they can sign 6293 and 2293 128-byte messages/s. Moreover, 5112 and 1507 messages with 128-byte wide can be verified every second employing our high-performance and efficient architecture, respectively. Note that increasing the size of the message increases the total latency such that each 1024-bit chunk adds 80 cycles. The proposed protected Designs I and II require 0.18 and 0.50 ms, respectively, to sign a message, while the verification does not need to be protected.

Complete signature and verification implementations with certificate handling are scarce in the literature. Hence, a direct comparison of the area utilization and performance is difficult. Nevertheless, we intend to put our results in the context with other relevant works to allow the reader a quick overview of other designs and architectures.

Table VI reports area and performance results for several digital signature schemes. As one can see, our proposed high-performance architecture achieves 27×, 21×, and 19× better performances for key generation, signing, and verifying operations compared to [24], respectively. However, our Design I is larger than this work and utilizes 3× and 5× more slice and DSP resources, respectively. Furthermore, Turan and Verbauwhede [24] have 10×, 8×, and 6× more delays than our efficient Ed25519 scheme for key generation, signing, and verifying, respectively, while ours occupies similar DSP counts and reduces 11% utilized slices. Hence, the superiority of computation over the Montgomery domain compared to the Edwards domain is shown despite coordinating conversion overheads.

Therefore, Design I has 89%, 86%, and 84% improvements in terms of $A{\cdot}T$ (Slice_count × Time) for key generation, signing, and verifying algorithm compared to [24], respectively.

TABLE VI
COMPARISON OF DIFFERENT DESIGNS FOR THE DIGITAL SIGNATURE ALGORITHM

Scheme		Platform	Security	SCA[1]	Area			Performance							Ref.
					Slices	DSP	BRAM	Freq. MHz	Keygen K CCs	OP/s	Signing K CCs	OP/s	Verifying[4] K CCs	OP/s	
NIST P-224	ECDSA	Virtex 6	112	(-)	8,132	124	-	100	-	-	100	996	117	854	[13]
Ed25519	EdDSA	ESP32	127	(-)	-	-	-	240	-	47	-	-	-	-	[35]
	EdDSA	RISC-V	127	(-)	-	-	-	320	-	41	-	-	-	-	[35]
	EdDSA	STM32F401	127	(+)	-	-	-	84	389	216	544	154	1,331	63	[36]
	EdDSA	Cortex A15	127	(+)	-	-	-	-	242	-	246	-	730	-	[37]
	EdDSA	ZYNQ 7000	127	(-)	2,192[2]	0	0	137	75	1,838[3]	-	-	-	-	[23]
	EdDSA	Zynq SoC	127	(+)	3,264	16	0	82	355	231	271	303	301	272	[24]
	EdDSA	**XC7Z020**	**127**	**(+)**	**2,893**	**16**	**0**	**136**	**60**	**2,279**	**59**	**2,293**	**90**	**1,507**	**Design II**
	EdDSA	**XC7Z020**	**127**	**(+)**	**9,723**	**81**	**0**	**73**	**12**	**6,276**	**12**	**6,293**	**14**	**5,112**	**Design I**
	EdDSA	Zynq SoC	127	(++)	4,303	16	0	82	904	91	690	119	301	272	[24]
	EdDSA	**XC7Z020**	**127**	**(++)**	**3,024**	**16**	**0**	**136**	**68**	**1,990**	**68**	**2,003**	**90**	**1,507**	**Design II**
	EdDSA	**XC7Z020**	**127**	**(++)**	**9,836**	**81**	**0**	**73**	**13**	**5,441**	**14**	**5,462**	**14**	**5,112**	**Design I**
NIST P-256	ECDSA	Virtex 6	128	(-)	10,625	136	-	100	-	-	127	787	148	674	[13]
	ECDSA	Virtex-II Pro 30	128	(+)	2,085	7	9	68	1079	63[3]	-	-	-	-	[14]
	ECDSA	Xilinx Virtex 5	128	(-)	3,564[2]		-	20	-	-	143	140	182	110	[12]
SM2	ECDSA	Altera EP2S30	128	(+)	4,742	8	-	62	48	1,298[3]	-	-	-	-	[15]

[1](-): without countermeasures, (+): SPA countermeasures, (++): DPA countermeasures
[2]Slices are estimated as KLUT/4.
[3]These values are only for the core operations scalar multiplication, without all pre- and post-processing and hashing.
[4]Since verifying operation does not need to be protected, the optimized and non-constant-time results are reported.

TABLE VII
EDDSA PARAMETERS FOR ED25519 [26]

Parameter	Value	Parameter	Value
p	$2^{255}-19$	b	256
c	3	n	254
d	$-121665/121666$	a	-1
$H(x)$	SHA-512(x)	Encoding of $GF(p)$	255-bit encoding
L	$2^{252}+27742317777372353535853951937790883648493$	$PH(x)$	x

Algorithm 1 EdDSA Key Generation Operations [26]

Input: a b-bit private key SK_A
Output: a scalar s, a public key $\underline{A}$, and a $prefix$
1: $h = (h_0, \ldots, h_{2b-1}) \leftarrow H(SK_A)$
2: $prefix \leftarrow (h_b, h_{b+1}, \ldots, h_{2b-1})$
3: $s \leftarrow 2^n + \sum_c^{n-1} 2^i h_i$
4: $\mathcal{A} \leftarrow [s]\mathcal{B}$
5: $\underline{A} \leftarrow enc(\mathcal{A})$
6: **return** s, $\underline{A}$, $prefix$

Algorithm 2 EdDSA Signing Operations [26]

Input: a scalar s, a b-bit public key $\underline{A}$, a b-bit $prefix$, a message M, a constant string C, an integer F indicating signature scheme
Output: $2b$-bit signature $(\underline{R}, \underline{S})$
1: $h = (h_0, \ldots, h_{2b-1}) \leftarrow H(dom(F, C) \mid prefix \mid M)$
2: $r \leftarrow \sum_0^{2b-1} 2^i h_i$
3: $\overline{r} \leftarrow r \bmod L$
4: $\mathcal{R} \leftarrow [\overline{r}]\mathcal{B}$
5: $\underline{R} \leftarrow enc(\mathcal{R})$
6: $h = (h_0, \ldots, h_{2b-1}) \leftarrow H(dom(F, C) \mid \underline{R} \mid \underline{A} \mid M)$
7: $k \leftarrow \sum_0^{2b-1} 2^i h_i$
8: $\overline{k} \leftarrow k \bmod L$
9: $S \leftarrow (\overline{r} + \overline{k} \times s) \bmod L$
10: $\underline{S} \leftarrow enc(S)$
11: **return** $(\underline{R}, \underline{S})$

Algorithm 3 EdDSA Verifying Operations [26]

Input: a b-bit public key $\underline{A}$, $2b$-bit signature $(\underline{R}, \underline{S})$, a message M, a constant string C, an integer F indicating scheme
Output: an acceptance validation
1: $h = (h_0, \ldots, h_{2b-1}) \leftarrow H(dom(F, C) \mid \underline{R} \mid \underline{A} \mid M)$
2: $k \leftarrow \sum_0^{2b-1} 2^i h_i$
3: $\overline{k} \leftarrow k \bmod L$
4: $\mathcal{X} \leftarrow \mathcal{R} + [\overline{k}]\mathcal{A}$
5: $\mathcal{Y} \leftarrow [S]\mathcal{B}$
6: **if** $\mathcal{X} = \mathcal{Y}$ **then**
7: **return** True
8: **else**
9: **return** False
10: **end if**

Moreover, Design II shows a significant improvement in terms of $A \cdot T$, i.e., 91%, 88%, and 84% for key generation, signing, and verifying algorithm compared to [24], respectively.

Considering the importance of utilized DSP in the FPGA-based architecture, we present a comparison in terms of $A_d \cdot T$ (DSP_count $\times$ Time). Thus, our Design I improves 81%, 76%, and 73% efficiencies in terms of $A_d \cdot T$ for key generation, signing, and verifying algorithm compared to [24], respectively. Furthermore, our proposed Design II improves 90%, 87%, and 82% efficiencies in this term for key generation, signing, and verifying algorithm compared to [24], respectively.

Moreover, the work in [23] proposed a nonconstant-time point multiplication core for Ed25519. Although it can compute 1838 ECPM per second, the architecture is vulnerable to SPA. Notably, the reported area does not include all the required modules for providing a digital signature, such as hash function and modulus L reduction. We explore that SHA-512 increases almost 25% utilized area in Ed25519.

Applying DPA countermeasures decreases efficiency due to executing more operations. However, our protected Design I (and II) improves almost 95% (96%) and 89% (94%) efficiency in terms of $A \cdot T$ and $A_d \cdot T$ for signing a message.

To compare different implementation approaches, some software-based and heterogeneous configurations of Ed25519 are listed in Table VI. With a variety of performance optimizations in hardware implementations, the throughput is significantly increased compared with software-based

implementation and heterogeneous computing. Hence, our design achieves almost 40 times speedup compared to [36].

V. Conclusion

In this article, we have proposed hardware design strategies for recently proposed Edwards curve digital signatures Ed25519 on Xilinx Zynq-7020 FPGA, including advanced protection against side-channel attacks. The proposed architectures achieve above 84% efficiency improvement of the area–time product using pipelined architecture and interleaved multiplication. Our high-performance and efficient architectures compute more than 6200 and 2200 signings and 5100 and 1500 verifications per second, respectively. We also show the design can outperform recently presented works using only moderate resource requirements.

Appendix

Ed25519 has some critical parameters shown in Table VII. EdDSA algorithms are described in Algorithms 1–3, respectively. According to [26], an encoded integer $\underline{S} = \text{enc}(S)$ can be shown in its little-endian convention. In addition, when an element $\mathcal{P} = (x, y)$ is encoded, its y-coordinate should be encoded first, and then, its most significant bit is substituted by the least significant bit of its x. The $\text{dom}(x, y)$ string function is blank for Ed25519.

Acknowledgment

The authors would like to thank the reviewers for their comments.

References

[1] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-speed high-security signatures," in *Proc. 13th Int. Workshop*, Nara, Japan, Sep./Oct. 2011, pp. 124–142.

[2] A. C. Aldaya, C. P. García, and B. B. Brumley, "From A to Z: Projective coordinates leakage in the wild," Cryptol. ePrint Arch., Tech. Rep. 2020/432, 2020.

[3] K. Ryan, "Return of the hidden number Problem: A widespread and novel key extraction attack on ECDSA and DSA," *Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 1, pp. 146–168, Nov. 2018.

[4] D. J. Bernstein and T. Lange. (2011). *Security Dangers of the Nist Curves*. [Online]. Available: https://www.hyperelliptic.org/tanja/vortraege/20130531.pdf

[5] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Santa Fe, NM, USA, Nov. 1994, pp. 124–134.

[6] N. Bindel, U. Herath, M. McKague, and D. Stebila, "Transitioning to a quantum-resistant public key infrastructure," in *Proc. IACR*, 2017, p. 460.

[7] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for CRYSTALS-Kyber post-quantum cryptograsphy," Cryptol. ePrint Arch., Tech. Rep. 2021/563, 2021.

[8] (2020). *Things That Use Ed25519*. [Online]. Available: https://ianix.com/pub/ed25519-deployment.html

[9] P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch, "A performance study of crypto-hardware in the low-end IoT," in *Proc. IACR*, 2021, p. 58.

[10] M. Bisheh Niasar, R. Azarderakhsh, and M. Mozaffari Kermani, "Efficient hardware implementations for elliptic curve cryptography over Curve448," in *Proc. 21st Int. Conf. Cryptol. India*, Bangalore, India, Dec. 2020, pp. 228–247.

[11] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Area-time efficient hardware architecture for signature based on Ed448," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, early access, Mar. 23, 2021, doi: 10.1109/TCSII.2021.3068136.

[12] B. Glas, O. Sander, V. Stuckert, K. D. Müller-Glaser, and J. Becker, "Prime field ECDSA signature processing for reconfigurable embedded systems," *Int. J. Reconfigurable Comput.*, vol. 2011, Oct. 2011, Art. no. 836460.

[13] B. Panjwani, "Scalable and parameterized hardware implementation of elliptic curve digital signature algorithm over prime fields," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2017, pp. 211–218.

[14] J. Vliegen *et al.*, "A compact FPGA-based architecture for elliptic curve cryptography over prime fields," in *Proc. 21st IEEE Int. Conf. Appl.-Specific Syst., Architectures Processors*, 2010, pp. 313–316.

[15] D. Zhang and G. Bai, "High-performance implementation of SM2 based on FPGA," in *Proc. 8th IEEE Int. Conf. Commun. Softw. Netw. (ICCSN)*, Jun. 2016, pp. 718–722.

[16] P. Sasdrich and T. Güneysu, "Efficient elliptic-curve cryptography using curve25519 on reconfigurable devices," in *Proc. 10th Int. Symp.*, D. Goehringer, M. D. Santambrogio, J. M. P. Cardoso, and K. Bertels, Eds., Vilamoura, Portugal, 2014, pp. 25–36.

[17] P. Sasdrich and T. Güneysu, "Implementing Curve25519 for side-channel-protected elliptic curve cryptography," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 9, no. 1, pp. 1–15, Nov. 2015.

[18] P. Sasdrich and T. Gäneysu, "Exploring RFC 7748 for hardware implementation: Curve25519 and Curve448 with side-channel protection," *J. Hardw. Syst. Secur.*, vol. 2, no. 4, pp. 297–313, Dec. 2018.

[19] M. Bisheh Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani, "Fast, small, and area-time efficient architectures for key-exchange on Curve25519," in *Proc. IEEE 27th Symp. Comput. Arithmetic (ARITH)*, Jun. 2020, pp. 72–79.

[20] P. Koppermann, F. DeSantis, J. Heyszl, and G. Sigl, "X25519 hardware implementation for low-latency applications," in *Proc. Euromicro Conf. Digit. Syst. Design*, P. Kitsos, Ed., Limassol, Cyprus, 2016, pp. 99–106.

[21] P. Koppermann, F. De Santis, J. Heyszl, and G. Sigl, "Low-latency X25519 hardware implementation: Breaking the 100 microseconds barrier," *Microprocessors Microsyst.*, vol. 52, pp. 491–497, Jul. 2017.

[22] R. Salarifard and S. Bayat-Sarmadi, "An efficient low-latency point-multiplication over Curve25519," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 10, pp. 3854–3862, Oct. 2019.

[23] M. A. Mehrabi and C. Doche, "Low-cost, low-power FPGA implementation of ED25519 and CURVE25519 point multiplication," *Information*, vol. 10, no. 9, p. 285, Sep. 2019.

[24] F. Turan and I. Verbauwhede, "Compact and flexible FPGA implementation of Ed25519 and X25519," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 3, pp. 1–21, 2019.

[25] N. Samwel, L. Batina, G. Bertoni, J. Daemen, and R. Susella, "Breaking Ed25519 in WolfSSL," Cryptol. ePrint Arch., Tech. Rep. 2017/985, 2017.

[26] S. Josefsson and I. Liusvaara, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, document RFC 8032, 2017, pp. 1–60.

[27] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *Proc. 9th Int. Conf. Theory Pract. Public-Key Cryptogr.*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., New York, NY, USA, 2006, pp. 207–228.

[28] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson, "Twisted edwards curves revisited," Cryptol. ePrint Arch., Tech. Rep. 2008/522, 2008.

[29] M. Hamburg, "Fast and compact elliptic-curve cryptography," in *Proc. IACR*, 2012, p. 309.

[30] K. Okeya and K. Sakurai, "Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the Y-coordinate on a montgomery-form elliptic curve," in *Proc. Int. Workshop*, Paris, France, May 2001, pp. 126–141.

[31] D. F. Aranha, F. R. Novaes, A. Takahashi, M. Tibouchi, and Y. Yarom, "Ladderleak: Breaking ECDSA with less than one bit of nonce leakage," Cryptol. ePrint Arch., Tech. Rep. 2020/615, 2020.

[32] J. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Proc. Cryptograph. Hardw. Embedded Syst.*, Ç. K. Koç and C. Paar, Eds., Worcester, MA, USA, 1999, pp. 292–302.

[33] P. Schwabe. (Sep. 2013). *Scalar-Multiplication Algorithms*. [Online]. Available: https://cryptojedi.org/peter/data/eccss-20130911b.pdf

[34] M. Bisheh Niasar, R. Azarderakhsh, and M. Mozaffari Kermani, "Optimized architectures for elliptic curve cryptography over Curve448," in *Proc. IACR*, 2020, p. 1338.

[35] M. Scott, "On the deployment of curve based cryptography for the Internet of Things," in *Proc. IACR*, 2020, p. 514.

[36] H. Fujii and D. F. Aranha, "Curve25519 for the cortex-M4 and beyond," in *Proc. 5th Int. Conf. Cryptol. Inf. Secur. Latin Amer.*, Havana, Cuba, Sep. 2017, pp. 109–127.

[37] D. Bernstein and T. Lange. *EBACS: ECRYPT Benchmarking of Cryptographic Systems*. Accessed: Mar. 22, 2021. [Online]. Available: https://bench.cr.yp.to