

REPORT DOCUMENTATION PAGE			Form Approved OMB NO. 0704-0188		
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 27-02-2023		2. REPORT TYPE Final Report		3. DATES COVERED (From - To) 28-Jul-2017 - 27-Sep-2020	
4. TITLE AND SUBTITLE Final Report: Emerging side-channel resistant and resource-friendly elliptic curve algorithms and architectures			5a. CONTRACT NUMBER W911NF-17-1-0311		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER 611102		
6. AUTHORS			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAMES AND ADDRESSES Florida Atlantic University 777 Glades Road PO Box 3091 Boca Raton, FL 33431 -0991			8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211			10. SPONSOR/MONITOR'S ACRONYM(S) ARO		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S) 70374-NC.4		
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Reza Azarderakhsh
a. REPORT UU	b. ABSTRACT UU	c. THIS PAGE UU			19b. TELEPHONE NUMBER 561-297-4759

RPPR Final Report

as of 28-Feb-2023

Agency Code: 21XD

Proposal Number: 70374NC

Agreement Number: W911NF-17-1-0311

INVESTIGATOR(S):

Name: Reza Azarderakhsh
Email: razarderakhsh@fau.edu
Phone Number: 5612974759
Principal: Y

Organization: **Florida Atlantic University**

Address: 777 Glades Road, Boca Raton, FL 334310991

Country: USA

DUNS Number: 004147534

EIN: 596001874

Report Date: 27-Dec-2020

Date Received: 27-Feb-2023

Final Report for Period Beginning 28-Jul-2017 and Ending 27-Sep-2020

Title: Emerging side-channel resistant and resource-friendly elliptic curve algorithms and architectures

Begin Performance Period: 28-Jul-2017

End Performance Period: 27-Sep-2020

Report Term: 0-Other

Submitted By: Reza Azarderakhsh

Email: razarderakhsh@fau.edu

Phone: (561) 297-4759

Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: 1

STEM Participants:

Major Goals: Major goal of this project was around the efficient and secure implementations of elliptic curves cryptography and provide secure against the side-channel attacks. We have completed all of the tasks promised in the proposal and successfully published several papers and trained three PhD students working on this project.

Accomplishments: Various technical objectives proposed in this project and we have successfully completed them. Those include:

Fast and efficient implementations of elliptic curve cryptography on NIST newly recommended that appeared in standards in 2019 such as curve25519.

Optimized architectures for elliptic curve cryptography for Curve448 with hardware and FPGA implementations.

Efficient software implementations of Curve448 on embedded microcontrollers and security analysis.

Optimized implementations of SIKE one of the elliptic curve candidates for NIST post-quantum cryptography and its performance evaluation on hardware and FPGA.

Training Opportunities: three PhD students got trained working partly on this grant and published papers and acknowledged this grant.

Results Dissemination: The results of this work has been published and presented in various conferences such as ARITH, ISCIS, INDOCRYPT and IEEE Transactions on Circuits and Systems (TCAS).

Honors and Awards: Nothing to Report

Protocol Activity Status:

Technology Transfer: The PI of this grant is planning to commercialize the funding from this research and through a startup company formed at FAU incubator called PQSecure Technologies.

PARTICIPANTS:

Participant Type: Co PD/PI

RPPR Final Report

as of 28-Feb-2023

Participant: Mehran Mozaffari Kemrani

Person Months Worked: 1.00

Project Contribution:

National Academy Member: N

Funding Support:

Participant Type: Co PD/PI

Participant: Koray Karabina

Person Months Worked: 1.00

Project Contribution:

National Academy Member: N

Funding Support:

ARTICLES:

Publication Type: Journal Article

Peer Reviewed: Y

Publication Status: 1-Published

Journal: Journal of Cryptographic Engineering

Publication Identifier Type: DOI

Publication Identifier: 9-017-0177-2

Volume:

Issue:

First Page #: 1

Date Submitted: 8/30/18 12:00AM

Date Published: 12/7/17 5:00AM

Publication Location: United States

Article Title: Constructing multidimensional differential addition chains and their applications

Authors: Aaron Hutchinson, Koray Karabina

Keywords: Differential addition chains Side-channel resistance Elliptic curves Scalar multiplication Cryptographic algorithms

Abstract: We propose new algorithms for constructing multidimensional differential addition chains and for performing multidimensional scalar point multiplication based on these chains. Our algorithms work in any dimension and offer some key efficiency and security features. In particular, our scalar point multiplication algorithm is uniform, it can be parallelized, and differential addition formulas can be deployed. It also allows trading speed for precomputation cost and storage requirements. These key features and our theoretical estimates indicate that this new algorithm may offer some performance advantages over the existing point multiplication algorithms in practice. We also report some experimental results and verify some of our theoretical findings, and a simplistic Magma implementation is provided.

Distribution Statement: 3-Distribution authorized to U.S. Government Agencies and their contractors

Acknowledged Federal Support: Y

CONFERENCE PAPERS:

Publication Type: Conference Paper or Presentation

Publication Status: 1-Published

Conference Name: Cryptographers' Track at the RSA Conference

Date Received: 30-Aug-2018

Conference Date: 16-Apr-2018

Date Published: 07-Mar-2018

Conference Location: CA, USA

Paper Title: An Exposure Model for Supersingular Isogeny Diffie-Hellman Key Exchange

Authors: Brian Koziel, Reza Azarderakhsh, David Jao

Acknowledged Federal Support: Y

RPPR Final Report as of 28-Feb-2023

Publication Type: Conference Paper or Presentation

Publication Status: 1-Published

Conference Name: International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE)

Date Received: 30-Aug-2018

Conference Date: 17-Dec-2017

Date Published: 22-Nov-2017

Conference Location: India

Paper Title: Efficient Software Implementation of Laddering Algorithms Over Binary Elliptic Curves

Authors: Diego F. Aranha, Reza Azarderakhsh, Koray Karabina

Acknowledged Federal Support: **Y**

Partners

,

I certify that the information in the report is complete and accurate:

Signature: Reza Azarderakhsh

Signature Date: 2/27/23 3:09PM

Fast, Small, and Area-Time Efficient Architectures for Key-Exchange on Curve25519

Mojtaba Bisheh Niasar

Rami El Khatib

Reza Azarderakhsh

Mehran Mozaffari-Kermani

CEECs Department
Florida Atlantic University
Boca Raton, FL
mbishehniasa2019@fau.edu

CEECs Department
Florida Atlantic University
Boca Raton, FL
relkhatib2015@fau.edu

CEECs Department
Florida Atlantic University
Boca Raton, FL
razarderakhsh@fau.edu

Dept. of CSE.
University of South Florida
Tampa, FL
mehran2@usf.edu

Abstract—This paper demonstrates fast and compact implementations of Elliptic Curve Cryptography (ECC) for efficient key agreement over *Curve25519*. *Curve25519* has been recently adopted as a key exchange method for several applications and included in the National Institute of Standards and Technology (NIST) recommendations for public key cryptography. This paper presents three different performance level designs including lightweight, area-time efficient, and high-performance architectures. Lightweight hardware implementations are used for several Internet of Things (IoT) applications due to their resources being at premium. Our lightweight architecture utilizes 90% less resources compared to the best previous work while it is still more optimized in term of $A \cdot T$ (area $\times$ time). For efficient implementation from either time or utilized resources, our area-time efficient architecture can establish almost 7,000 key sessions per second which is 64% faster than the previous works. The area-time efficient architecture uses well scheduled interleaved multiplication combined with a reduction algorithm. Additionally, we offer a fast architecture for high performance applications based on the 4-level Karatsuba method and Carry-Compact Addition (CCA). Our high-performance architecture also outperforms previous work in terms of $A \cdot T$. The results show 9% and 29% improvement in $A \cdot T$ and $A_d \cdot T$ (DSP_count $\times$ time), respectively. All architectures are variable-base-point implemented on the Xilinx Zynq-7020 FPGA family where performance and implementation metrics are reported and compared. Finally, various side-channel attack countermeasures are embedded in the proposed architectures.

Index Terms—Curve25519, elliptic curve Diffie-Hellman (ECDH), field-programmable gate array (FPGA), point multiplication.

I. INTRODUCTION

Since the initial recommendation and standardization of Elliptic Curve Cryptography (ECC) in [1], [2] and [3], there has been significant progress towards security and performance of computations in various levels of elliptic curves including finite field arithmetic, curve level arithmetic, and protocol level computations [4]. Due to a concern from the community regarding the generation of potential weaknesses of the original curves defined and recommended by National Institute of Standards and Technology (NIST) in [1], organizations shifted their attention to a newly developed family of elliptic curves based on Montgomery curves (mainly *Curve25519*) with some attractive properties [5]. Recently, *Curve25519*

became the most popular scheme for elliptic curve Diffie-Hellman (ECDH) Key-Exchange which in late 2019 has been recently recommended by IETF RFC 7748 [6], RFC 8032 [7], and NIST 800-186 [8]. *Curve25519* is a 255-bit elliptic curve offering approximately 128-bit classical security. In comparison to the older versions of NIST recommended curves for ECDH for prime fields, *Curve25519* is the obvious winner as it is fast and secure with uniform computations, and no weaknesses known as of today. *Curve25519* has been employed in for several applications with software implementations and integration with SSL/TLS libraries. However, its usage in hardware and small device applications is not widely spread yet. For instance, Microchip Inc. recently developed an IoT device with ECC-based authentication and key exchange schemes over Google cloud based on NIST-P256 curve which originally was recommended by NIST back in 1999 [1]. In comparison to NIST-P256, *Curve25519* is smaller and faster and even more secure against known side-channel attacks including timing and cache attacks. Microsoft recently introduced their first IoT device called Azure Sphere which does not include *Curve25519* for public key cryptography. One reason could be the fact that due to interoperability, *Curve25519* was not included in early NIST recommendation and governments sectors were not allowed to use it. The other factor could be related to the migration to a new elliptic curve which takes several years to happen.

In this paper, we would like to investigate the performance of *Curve25519* in hardware for various performance levels so that IoT and embedded device manufacturing companies could adopt *Curve25519* as the mainstream for key exchange mechanism. More specifically three different architectures are provided to address wide range of applications including: (i) implementations on low-end devices with small foot prints, (ii) implementations on medium-end devices with silicon area and speed trade-offs, and (iii) implementations on high-end devices with fast computations. We prototype our implementations on Xilinx FPGAs and report computation times and occupied silicon areas for comparison to the prior work available in the literature. Our implementations for all mentioned three performance level implementations outperform the counterparts available in the open literature.

Prior work: The first hardware implementation of *Curve25519* appeared in [9] by Sasdrich and Güneysu which performs the point multiplication in 397 μ s and operates in 200 MHz. The focus in this work is on lightweight applications. Same authors in [10] added side-channel resistance in their earlier design to provide security against physical attacks. Moreover, more improvement in architecture security is done in [11] with adequate assessment from leakage point of view. In [12], [13], Kopperman *et al.* proposed high performance implementations and the reported total time for point multiplication on *Curve25519* is 92 μ s operating at 115 MHz. Salarifard *et al.* [14], also presented an optimized version targeting high performance applications which is considered as the fastest design with 44 μ s computation time operating at 87 MHz. Turan *et al.* [15] proposed lightweight design with the total time of 714 μ s at 105 MHz. Mehrabi and Doche [16] presented an optimized lightweight architecture which does not employ any DSPs and operates in 137.5 MHz with total time of 512 μ s.

The main contributions of this paper are summarized below:

- We propose three different hardware architectures for three performance levels of computing point multiplication over *Curve25519*.
- Efficient scheduling mechanism are proposed to compute one step of Montgomery ladder which improves the achieved time and area in comparison with those presented in previous.
- We develop optimized field arithmetic unit for both addition and multiplication. Our adder is based on Carry-Compact Addition (CCA) which allows choosing different parameters for area-time efficiency. Also, we investigated four different levels of Karatsuba multiplication for fast implementations of field multiplication.
- Our proposed architectures are implemented on a Xilinx FPGA family and show that our results outperform the counterparts available in the literature.

The rest of the paper is organized as follows. In Section II, we discuss the preliminaries of ECC based on *Curve25519*. In Section III, our proposed algorithms and architectures are discussed. In Section IV, details of FPGA implementations are provided. We discuss our results and compare to the counterparts in Section V. Finally, we conclude the paper in Section VI.

II. PRELIMINARIES

In this section, some relevant mathematical background are provided. Then, side-channel analysis attack protection will be described.

A. Field Arithmetic and ECDH Key Exchange

The elements of Galois Field $GF(p)$ are defined as $\{0, 1, \dots, p-1\}$ which is a finite field. *Curve25519* over $GF(p)$ is defined by $E: y^2 = (x^3 + 48666x^2 + x) \bmod p$ where $p = 2^{255} - 19$.

Curve25519 can be used for accelerating ECDH Key-Exchange X25519 [17] and provides efficient modular arithmetic operations including modular addition, subtraction, mul-

tiplication and inversion over $GF(p)$ using pseudo Mersenne prime reduction algorithm [18]. According to Fermat's Little Theorem (FLT), modular inversion over $GF(p)$ is computed by $a^{-1} \equiv a^{p-2}$. In [5], authors proposed an addition chain method using 254 squarings and 11 multiplications to compute a^{p-2} .

ECDH key exchange protocol computes a shared secret key over *Curve25519* between two parties using a set of public parameters. Hence, they can communicate securely by generated shared secret key, through, for instance, the current symmetric-key cryptography standard, i.e., Advanced Encryption Standard (AES).

B. Scalar Multiplication and Montgomery Ladder

Scalar multiplication $Q = k \cdot P$ is computed efficiently for public base point $P \in E$ and secret key k employing Montgomery ladder point multiplication algorithm.

In order to achieve efficient and resistant implementations of point addition (PA) and point doubling (PD) the Montgomery ladder, introduced by Peter L. Montgomery [19], is implemented. In each iteration of Montgomery ladder point multiplication shown in Algorithm 1, one PA and one PD are performed in constant execution time. This algorithm needs only x -coordinate of base point P to compute $k \cdot P$. Therefore, y -coordinate can be eliminated during computations when performed in projective coordinates. In the first stage of this algorithm, a map from affine to projective coordinates is executed such that $(x_p, y_p) = (X/Z, Y/Z)$. Suppose $P = (x_P, y_P)$ is given as the base point. Then, (1) to (4) describe PA and PD in each iteration of Montgomery ladder where $P_1 = (X_1, Z_1)$ and $P_2 = (X_2, Z_2)$:

$$X_{PD} = (X_1 - Z_1)^2 (X_1 + Z_1)^2 \quad (1)$$

$$Z_{PD} = X_1 Z_1 (X_1^2 + 121666 X_1 Z_1 + Z_1^2) \quad (2)$$

$$X_{PA} = 4(X_1 X_2 - Z_1 Z_2)^2 \quad (3)$$

$$Z_{PA} = 4x_p (X_1 Z_2 - Z_1 X_2)^2 \quad (4)$$

Operations in Montgomery ladder require field addition, subtraction and multiplication. After all Montgomery ladder iterations are done, field inversion is computed via FLT to map from projective to affine coordinates in the last stage.

C. Side-Channel Protection

Several considerations should be taken into account for protecting cryptographic implementations against side-channel analysis attacks (SCA). Using Montgomery ladder algorithm makes the scalar multiplication resistant against timing and simple power analysis attacks (SPA). Furthermore, authors in [20] introduced some countermeasures in order to implement resistant architectures against differential power analysis attacks (DPA). These approaches including point randomization, scalar blinding and memory address scrambling are implemented in research works presented in [10], [11].

- In point randomization, the base point $P = (X, Z)$ at the beginning of Montgomery ladder algorithm is projected by using a random value $\lambda \in \mathbb{Z}_{2^{255}} \setminus \{0\}$ such that $P_r = (\lambda \cdot X, \lambda \cdot Z)$. However, the scalar multiplication output

Algorithm 1 Montgomery Ladder Point Multiplication [19]**Input:** x_P s.t. $P = (x_P, y_P)$ and scalar $k = (k_{254}, k_{253}, \dots, k_0)_2$ **Output:** x_Q s.t. $Q = k.P = (x_Q, y_Q)$

```

1:  $X_1 \leftarrow x_P, Z_1 \leftarrow 1, X_2 \leftarrow 1, Z_2 \leftarrow 0, X_{1-2} \leftarrow x_P$ 
2: for  $i=254$  downto 0 do
3:   if  $k_i = 1$  then
4:      $(X_1, X_2) \leftarrow (X_2, X_1)$ 
5:      $(Z_1, Z_2) \leftarrow (Z_2, Z_1)$ 
6:   end if
7:    $T_1 \leftarrow X_1 + Z_1$ 
8:    $T_2 \leftarrow X_1 - Z_1$ 
9:    $T_3 \leftarrow X_2 + Z_2$ 
10:   $T_4 \leftarrow X_2 - Z_2$ 
11:   $T_5 \leftarrow T_1 \times T_1$ 
12:   $T_6 \leftarrow T_2 \times T_2$ 
13:   $T_7 \leftarrow T_2 \times T_3$ 
14:   $T_8 \leftarrow T_1 \times T_4$ 
15:   $T_9 \leftarrow T_5 \times T_6$ 
16:   $T_{10} \leftarrow T_6 - T_5$ 
17:   $T_{11} \leftarrow T_7 + T_8$ 
18:   $T_{12} \leftarrow T_7 - T_8$ 
19:   $T_{13} \leftarrow T_{10} \times 121666$ 
20:   $T_{14} \leftarrow T_{11} \times T_{11}$ 
21:   $T_{15} \leftarrow T_{12} \times T_{12}$ 
22:   $T_{16} \leftarrow T_6 + T_{13}$ 
23:   $T_{17} \leftarrow T_{16} \times T_{10}$ 
24:   $T_{18} \leftarrow T_{15} \times X_{1-2}$ 
25:   $(X_1, Z_1) \leftarrow (T_9, T_{17}), (X_2, Z_2) \leftarrow (T_{14}, T_{18})$ 
26: end for
27:  $Z_1 \leftarrow Z_1^{-1}$ 
28:  $x_Q \leftarrow X_1 \times Z_1$ 
29: return  $x_Q$ 

```

is not changed due to the fact that $x_P = \frac{X}{Z} = \frac{\lambda \cdot X}{\lambda \cdot Z} \pmod{p}$.

- Scalar blinding changes the second term in scalar multiplication which is the secret key k . Multiple of group order $\#E$ can be added to k in order to provide data dependency between swap function in Montgomery ladder and corresponding bit in k . Let r be a random value, blinded scalar is computed by $k_r = k + r \times \#E$.
- Memory address scrambling prevents data leakage based on accessing specific location in memory. If memory addresses are revealed for attackers in each execution, key bits will be known. Therefore, memory addresses should be generated randomly in each execution. This feature can be achieved using a linear feedback shift register (LFSR) for generating a random address and a random value as a seed for the LFSR.

III. PROPOSED ALGORITHM AND ARCHITECTURE

In this section, three different architectures are proposed for different performance levels including lightweight, area-time efficient, and high-performance designs.

A. Design I: Lightweight Architecture

Fig. 1 shows the top level architecture for our proposed lightweight point multiplication scheme. It consists of Controller, ROM, Dual Port RAM and Arithmetic Unit. Arithmetic unit consists of Modular Addition and Modular Multiplication illustrated in Fig. 1-(a). In this architecture, a controller is

connected to modular arithmetic units. On the other hand, controller has an interface with ROM which gives the instruction code for each cycle. During each cycle, controller sets required addresses for arithmetic units considering corresponding bit of chosen scalar k and the instruction code.

ROM size is 560×15 -bit implemented by a 18Kb Block-RAM. ROM includes control commands and three data addresses for arithmetic units. Furthermore, a RAM is considered for decreasing resources as much as possible in order to save intermediate data. A 255-bit data is stored by 15×17 -bit words in RAM. Thus, all computations are performed sequentially and work with 17-bit data in each iteration due to the DSP block input size.

1) *Modular Addition/Subtraction*: Addition/Subtraction is computed between two 17-bit numbers in each iteration while the carry/borrow is propagated to the next digit. Although loading inputs and storing the results from/to RAM take some iterations, the architecture is designed carefully in parallel to compute in 30 iterations. In the last iteration, the carry/borrow is reported to controller to set a flag. After computing addition/subtraction, reduction computation is performed in constant time. This computation needs adding/subtracting the previous result with 19 which takes another 30 iterations to store in another address. Finally, the mentioned flag indicates RAM address for the correct result.

2) *Modular Multiplication*: Modular multiplication is designed based on product scanning approach. According to this method, 15 multiplications are performed for each 17-bit product digit. Product scanning approach is completely explained in [15] utilizing 15 parallel DSPs. In the lightweight architecture, only 1 DSP unit is implemented which performs multiplications sequentially. Computation for each product digit can have maximum 42 bits, so 25 bits should be propagated to next digit as carry. It can be shown by $C_{i+1} \times 2^{17} + P_i$, where C_i and P_i are carry and product digits in i^{th} iteration, respectively. However, C_{15} should be considered for reduction.

The most efficient reduction method for *Curve25519* is pseudo Mersenne prime algorithm. In this method, modulo $p = 2^{255} - 19$ is computed by the product of $19 \times C_{15}$. After computing $19 \times C_{15}$, we cascade a modular multiplication with modular addition to perform the reduction stage. Eventually, performing a modular addition between $19 \times C_{15}$ and $P_{14} \times 2^{14 \times 17} + \dots + P_0$ leads to modular multiplication result.

B. Design II: Area-Time Efficient Architecture

The presented area-time efficient architecture considers a trade-off between resource utilization and required time in comparison with fast and small architectures. In this architecture shown in Fig. 1-(b), modular multiplication is implemented by 2-level Karatsuba multiplication. One of the most important features for efficient architecture is utilization ratio determined by:

$$Utilization\ Ratio\ \% = \frac{Utilization\ Time}{Total\ Time} \times 100 \quad (5)$$

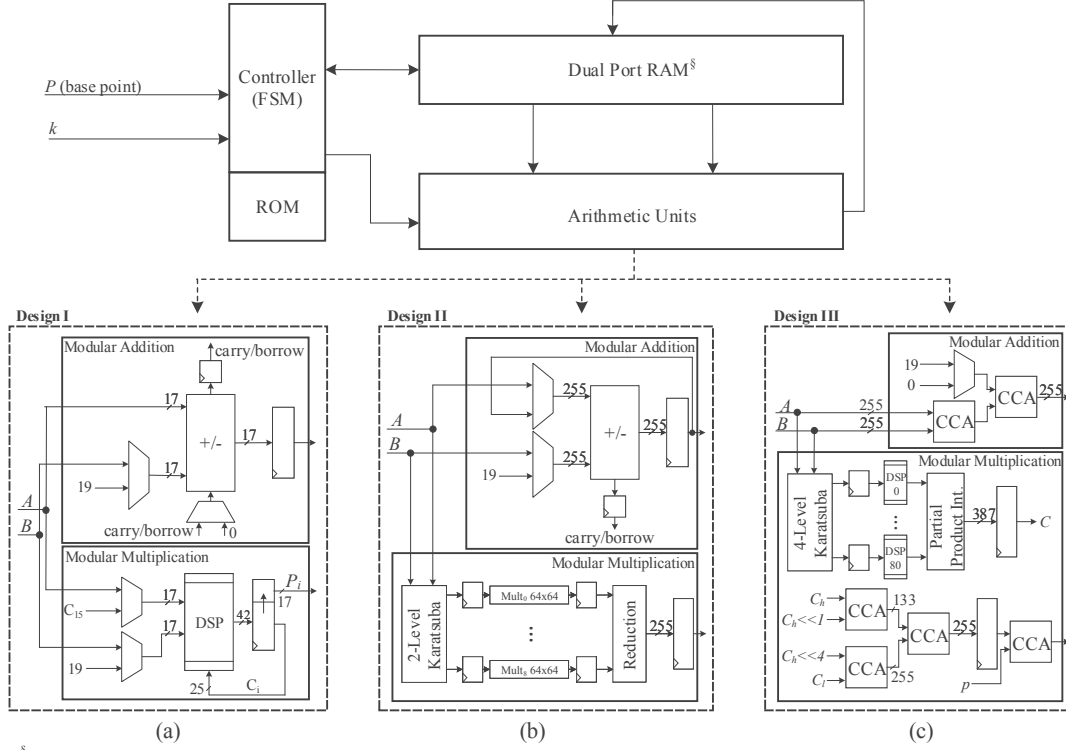


Figure 1. The proposed architectures for different performance levels: Top level architecture: (a) The arithmetic units architecture in the proposed lightweight design, (b) The arithmetic units architecture in the proposed area-time efficient design, and (c) The arithmetic units architecture in the proposed high-performance design.

In the area-time efficient architecture, we maximize utilization ratio for arithmetic units by concurrent computations and pipelined multiplications.

1) *Modular Addition/Subtraction*: This unit is implemented by two addition/subtraction operations. The first operation computes $C = A \pm B$ in one cycle and sets carry/borrow flag. The second operation takes the value C and computes $C' = C \pm 19$ during next cycle. The carry/borrow flag determines the correct output between C and C' which should be stored into RAM. Therefore, modular addition/subtraction can be performed in 4 cycles including computations and RAM communication. Modular addition/subtraction is performed in parallel with modular multiplication. Thus, its latency is absorbed by multiplication cycles to increase utilization ratio.

2) *Modular Multiplication*: Multiplication is executed by combination of Karatsuba and combo methods. Centerpiece of the multiplication unit is 64-bit by 64-bit combo multiplier using 4 DSP components. To improve the delay of combo multiplier, the first and last embedded pipeline stages of DSP are used. 2-level Karatsuba multiplications are applied on 255-bit inputs to convert them to 9 partial multiplications. Then, partial multiplications are computed in parallel by combo multipliers in 8 cycles. In order to design high-throughput schemes, multiplication and reduction stages are interleaved employing pipeline architecture. Thus, modular multiplication

latency is only 8 cycles after the first multiplication.

C. Design III: High-Performance Architecture

The most important component of arithmetic unit is modular multiplication. This unit is performed in both Montgomery ladder and FLT computations, and its performance roughly determines that of the entire architecture. It consists of a multiplication algorithm followed by a reduction stage. There are several multiplication algorithms which can be considered for the area or time optimization (e.g., Schoolbook or Toom-3 methods). For ECC computation over prime field, Karatsuba multiplication [21] has suitable efficiency in comparison with other methods [12], [22].

Different levels of Karatsuba can be applied on 256-bit by 256-bit multiplication shown in Fig. 2. Using more than 4-level Karatsuba reduces utilization of DSP resources due to DSP specifications which takes 25-bit by 18-bit signed numbers as inputs. Therefore, 1 to 4 levels of Karatsuba are considered for modular multiplication. Although increasing the levels of Karatsuba reduces required latency for multiplication, it expands addition tree which limits maximum operation frequency and raises area utilization. As a result, we should consider the compromise between latency, frequency and resource utilization. Fig. 3 presents total required time, resource utilization and $A \cdot T$ (area $\times$ time) to compute a point multiplication based on different applied levels of Karatsuba

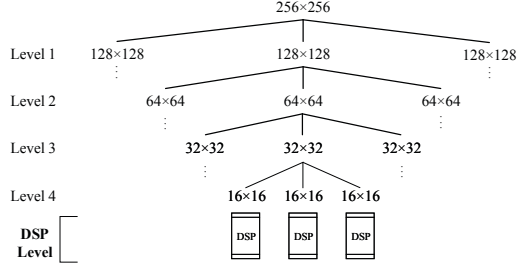


Figure 2. Karatsuba multiplication breakdown for 256-bit by 256-bit multiplication.

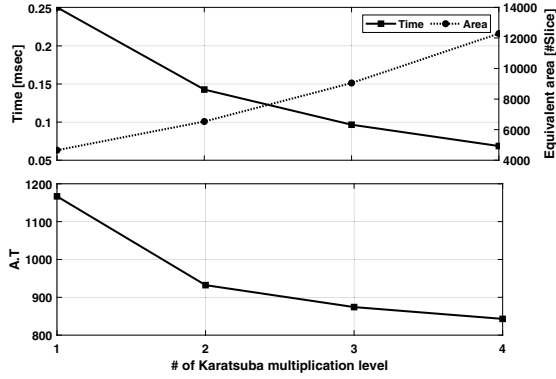


Figure 3. Different levels of Karatsuba multiplication performance: (a) Time and equivalent area for a point multiplication using 1 to 4-level of Karatsuba multiplication scheme, and (b) Implementations performance in terms of $A \cdot T$.

multiplication. According to this figure, 4-level Karatsuba results in minimum latency and utilized almost 2 times more resources compared to 2-level Karatsuba. Eventually, 4-level Karatsuba has the best performance in terms of $A \cdot T$ compared to other implementations.

The proposed high-performance architecture is illustrated in Fig. 1-(c). This architecture includes a Controller, ROM, Modular Addition and Modular Multiplication. Controller takes instruction code corresponding to each cycle from ROM to compute $Q = k \cdot P$. In order to reduce loading and storing cycles, register files are used instead of RAM in this design. Thus, all required addresses are read in one cycle from ROM to speed up the computations.

1) *Modular Addition/Subtraction* : This unit is implemented based on Carry-Compact Addition (CCA) scheme introduced in [23]. Our developed adder can significantly reduce the critical path in an FPGA target by choosing different parameters for area and time efficiency in order to combine carry-look ahead and parallel prefix adders. Two main parameters in designed CCA are L and H for level of carry chain and hierarchy level, respectively. For computing modulo p , two addition units are cascaded and both are 255-bit. The first unit performs $C = A + B$, and the second unit performs the reduction $C \bmod p$. $L = 30$ and $H = 2$ are found experimentally for our target FPGA, which causes the minimum critical path.

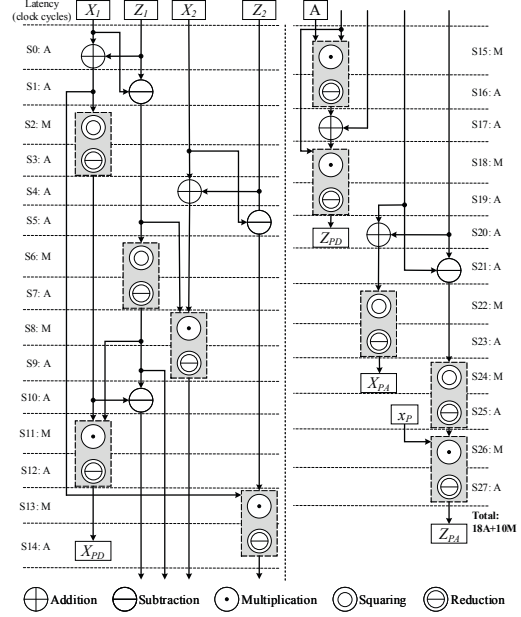


Figure 4. Design I data dependency diagram for one step Montgomery ladder execution over *Curve25519*.

2) *Modular Multiplication*: In proposed high-performance architecture interleaved multiplication is designed for maximizing the performance. All partial multiplication steps are performed in one cycle using 4-level Karatsuba multiplication, while 81 DSP components work in parallel. Using modulo $2p = 2^{256} - 38$ in intermediate steps for the fast reduction, $C = A \times B$ with 387-bit is achieved from consecutive Karatsuba multiplication.

Reduction computation follows multiplications using modulo $p = 2^{255} - 19$ at the end of computation using developed CCA. Reduction is applied on C as follows:

$$C_l \leftarrow C(254 \dots 0) \quad (6)$$

$$C_h \leftarrow C(386 \dots 255) \quad (7)$$

$$t_0 \leftarrow (C_h + (C_h << 1)) + (C_l + (C_h << 4)) \quad (8)$$

$$t_1 \leftarrow t_0 - p \quad (9)$$

Additionally, we employ several registers between stages to design high-throughput architecture. Therefore, low latency modular multiplication is performed in 3 clock cycles.

IV. FPGA IMPLEMENTATIONS

In this section, scheduling, latency, implementation considerations and SCA protection methods are discussed. In all designs, we implement modular inversion with 265 modular multiplications performed after point multiplication.

A. Design I: Lightweight Implementation

Fig. 4 demonstrates lightweight architecture data dependency for executing Montgomery ladder. As described in Section III-A, reduction is implemented by modular addition which follows multiplication. Modular addition and modular

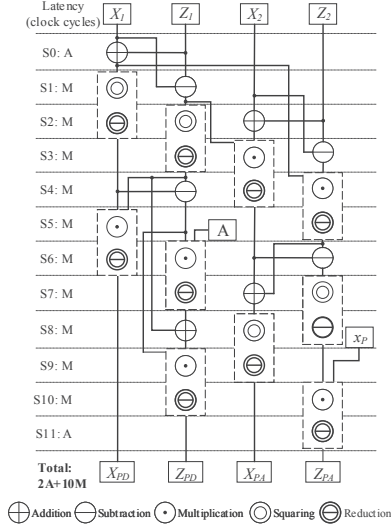


Figure 5. Design II data dependency diagram for one step Montgomery ladder execution over *Curve25519*.

multiplication required cycles are 61 and 277 clock cycles, respectively.

B. Design II: Area-Time Efficient Implementation

Fig. 5 shows Montgomery ladder scheduling for area-time efficient architecture. The latency of modular addition/subtraction, multiplication and reduction are 4, 8 and 4 clock cycles, respectively. As detailed in Section III-B, modular addition and reduction latencies are absorbed by modular multiplication due to interleaved architecture. Additionally, multiplier is kept busy most of the time to maximize the utilization ratio which can be calculated by $\frac{80}{88} \times 100 \simeq 91\%$.

C. Design III: High-Performance Implementation

Montgomery ladder scheduling for fast architecture is illustrated in Fig. 6. According to this figure, PA and PD are executed in only 13 cycles using carry-compact addition and 4-level consecutive Karatsuba multiplication. Modular multiplication requires 3 clock cycles, while it is interleaved by the pipeline strategy described in Section III-C.

The critical path of the circuit is almost 17 ns. According to (8), an addition tree is implemented using one 256-bit CCA and two parallel CCAs which have 133-bit and 255-bit width. CCA implementation causes 11.3% improvement in comparison with regular addition due to its low latency architecture.

D. Side-Channel Protected Implementation

Various side-channel attack countermeasures are embedded in proposed architectures. 387-bit fresh random value (255 bits for λ , 129 bits for r and 3 bits for address randomization) are required for SCA resistant scheme. Since designing the random number generator is not in scope of this study, we assume it is delivered externally. Point randomization is achieved by using $(\lambda \cdot x_P, \lambda)$ as a randomized projective coordinate where $X_P = \lambda \cdot x_P$ and $Z_P = \lambda$. This approach does not change

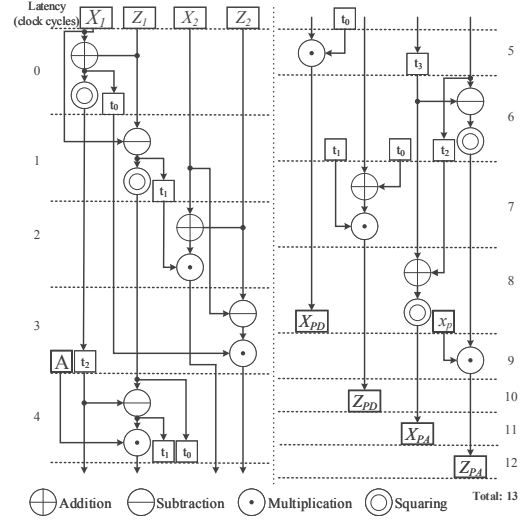


Figure 6. Design III data dependency diagram for one step Montgomery ladder execution over *Curve25519*.

area utilization, since the proposed architectures are variable-base-point implemented and accept different base points for every operation.

In [11], authors proposed that at least half of bit size $\#E$ for r is required in order to prevent advanced SCA [24]. Addition between k and $r \times \#E$ generates a 384-bit scalar which causes executing 384 Montgomery ladder steps instead of 255 steps. Therefore, applying scalar blinding increases latency. Similar to [10], we suppose blinded scalar is provided externally.

Memory address scrambling as the third strategy does not have significant overhead for Design I and Design II, because just the LFSR is added to them. In Design III, the register files are expanded to provide 2^3 times larger space compared to the required memory.

V. IMPLEMENTATION RESULTS AND COMPARISON

Our proposed point multiplication architectures for lightweight, area-time efficient, and high-performance designs are synthesized with Xilinx Vivado 2019.2 on a Xilinx Zynq-7020 FPGA device. All given results are obtained after place-and-route (PAR). We report the area and time specifications for our proposed work and compare to the counterparts in Table I. As one can see, the results are compared in terms of $A \cdot T$ and $A_d \cdot T$, where A , A_d and T are required area, DSP block, and time, respectively. To have a fair comparison, we also employ an equivalent area utilization to provide a baseline comparison. To investigate the equivalent slice number corresponding, the Vivado synthesis tool configuration has been changed to use only LUT instead of DSP. Therefore, we recognized one DSP is almost equivalent to 100 occupied Slices.

Results for Design I: The lightweight architecture requires only few resources on our target device, i.e., 110 (0.8%) of the Slices, 1 (0.5%) of DSP block and 1 (0.7%) BlockRAM. It occupies 90% less resources compared to [9], [10]. According

Table I
FPGA IMPLEMENTATION RESULTS FOR DIFFERENT PERFORMANCE LEVEL ARCHITECTURES

Work	Platform	Area					Time			$A \cdot T$ (10^{-3})	$A_d \cdot T$ (10^{-3})
		LUTs	FFs	Slices	DSPs	BRAMs	Latency [cc]	Freq [MHz]	Total time [μ s]		
Lightweight Architecture											
[9], [10]	XC7Z020	2,783	3,592	1,029	20	2	79,400	200	397	1,203 (6.1%)	7.9 (31.6%)
[15]	Zynq SoC	2,707	962	775	15	0	75,000	105	714	1,624 (30.4%)	10.7 (49.5%)
[16]	Zynq 7000	7,380	3,141	~	0	0	70,370	137.5	512	1,511 (25.2%)	~
[25] [†] Mont.	XC7Z020	1,069	1,894	565	16	7	58,967	190	310	671	5.0
Our Design I	XC7Z020	290	277	110	1	1	1,076,251	200	5381	1,130	5.4
Area-Time Efficient Architecture											
[9], [10]	XC7Z020	2,783	3,592	1,029	20	2	79,400	200	397	1,203 (22.4%)	7.9 (35.4%)
[16]	Zynq 7000	7,380	3,141	~	0	0	70,370	137.5	512	1,511 (38.2%)	~
[25] [†] End.	XC7Z020	4,217	4,413	1,691	27	10	29,739	190	157	689	4.2
Our Design II	XC7Z020	7,364	3,995	2,951	36	5.5	27,973	196	143	934	5.1
High-Performance Architecture											
[14]	XC7Z020	12,989	2,705	3,362	182	0	3,858	87	44	948 (9.3%)	8.0 (28.8%)
[12]	Zynq 7030	26,483	21,107	8,639	260	0	13,639	115	118	4,087 (79%)	30.7 (81.4%)
[13]	Zynq 7030	17,939	21,077	6,161	175	0	10,465	115	92	2,177 (60.5%)	16.1 (64.6%)
Our Design III	XC7Z020	14,337	4,107	4,237	81	0.5	4,117	60	70	860	5.7

[†]These designs use FourQ Curve.

Table II
FPGA IMPLEMENTATION RESULTS FOR SIDE-CHANNEL PROTECTED ARCHITECTURES

Work	Platform	Area					Time			$A \cdot T$ (10^{-3})	$A_d \cdot T$ (10^{-3})
		LUTs	FFs	Slices	DSPs	BRAMs	Latency [cc]	Freq [MHz]	Total time [μ s]		
[11]	XC7Z020	2,077	4,223	1,006	20	2	11,4980	200	575	1,728	11.5
[14]	XC7Z020	23,709	9,837	5,928	170	0	5,289	81	65	1,490	11
Our Design I	XC7Z020	344	401	160	1	1	1,692,412	200	8,462	2,200	8.5
Our Design II	XC7Z020	7,462	4,188	3,018	36	5.5	42,464	196	217	1,434	7.8
Our Design III	XC7Z020	24,817	9,285	6,183	81	0.5	6,178	60	103	1,471	8.3

to our Design I in Fig. 4, one step Montgomery ladder is performed in $18A + 10M = 18 \times 61 + 10 \times 277 = 3,868$ clock cycles. As shown 255 Montgomery ladder computations followed by 265 modular multiplications are executed in 1,076,251 clock cycles to compute a point multiplication. This lightweight architecture computes more than 185 key sessions per second. We observe that $A \cdot T$ is 1,130 which shows 6.1%, 30.4% and 25.2% improvement compared to [9], [10], [15] and [16], respectively. Moreover, the lightweight architecture reduces 31.6% and 49.5% $A_d \cdot T$ in comparison to [9], [10] and [15], respectively.

Results for Design II: Our proposed area-time efficient architecture requires 143 μ s for computing a point multiplication. Design II needs $2A + 10M = 88$ cycles for performing Montgomery ladder shown in Fig. 5. Each iteration of inversion needs to execute a multiplication and reduction procedure which take $(8 + 4) \times 265 = 3,180$ clock cycles. Considering required cycles for handling the units and loading and storing in RAM, its latency is 27,973 clock cycles to compute a point multiplication. This design uses silicon area more than lightweight architecture and less than high-performance design, i.e., 2,951 Slices, 36 DSPs, and 5.5 BlockRAMs. Additionally, its latency is 38 times less than the proposed lightweight design and 6.8 times more than the high-performance architecture. Thus, this design can

be considered for some applications which needs a trade-off between area and time. According to Table I, the area-time efficient architecture shows 22.4% and 38.2% decrease in $A \cdot T$ compared with [9], [10] and [16], respectively. Also, it is almost 64% faster than [9], [10] occupying only 16 more DSPs.

Results for Design III: Our proposed high-performance design employs 4-level consecutive Karatsuba multipliers utilizing 4,237 Slices, 81 DSPs and 0.5 BlockRAM, and its total latency is equal to $13 \times 255 + 3 \times 265 = 4,110$ plus 7 clock cycles for initialization/handling the units. The circuit maximum operation frequency is dropped to 60 MHz due to addition tree circuit. However, a session key can be computed in 70 μ s. This architecture has 9.3% improvement in terms of $A \cdot T$ metrics in comparison to [14]. Furthermore, the critical resource in FPGA is DSP component which has significant effect on performance. In [14], 182 DSPs are utilized which are 82.7% of total resources. Our proposed design saves 55.5% DSPs compared to [14], and its performance is 28.8% more efficient in terms of $A_d \cdot T$ metric.

Between different architectures for providing 128-bit security level, the work in [25] shows better performance employing FourQ curve. These architectures use simpler arithmetic in $\mathbb{F}_{p^2}$ over Mersenne prime. However, FourQ curve has not

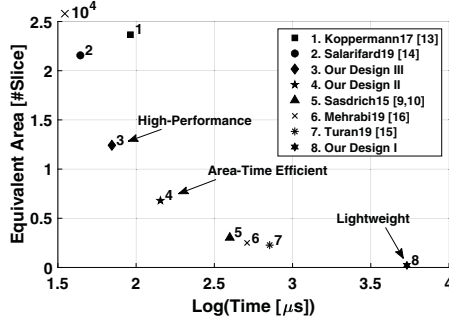


Figure 7. FPGA implementation performance in terms of $A \cdot T$ for different architectures.

been investigated for side-channel analysis and has not been standardized/recommended for standardization by NIST.

All reported implementations over *Curve25519* are illustrated in Fig. 7 for better visualization and comparison. According to this figure, there are different optimization goals based on application. Our lightweight architecture is close to x -axis where area-constraints are considered, while the proposed high-performance design is so nearby to y -axis where time-constraints have more priority. Meanwhile, the area-time efficient architecture is placed between them. As one can see from Fig. 7, the proposed implementations are all superior in their respective use case.

We also added side-channel resistant considerations to the proposed architectures, and the results are reported in Table II. According to this table, our proposed architectures outperform the designs in [11] and [14].

VI. CONCLUSION

In this paper, we offer three hardware design strategies for recently proposed elliptic curve *Curve25519* implemented on Xilinx Zynq-7020 FPGA. In these designs, one scalar point multiplication is computed in 5.4 ms, 143 μs and 70 μs for lightweight, area-time efficient and high-performance architectures, respectively. Additionally, the required resources for these architectures are less than the one in previous work.

The proposed architectures improve point multiplication over *Curve25519* in terms of $A \cdot T$ and $A_d \cdot T$ employing pipelined architecture, low latency CCA and interleaved multiplication. Additionally, efficient scheduling for parallel computation is designed to high performance. Various side-channel attack countermeasures are embedded in proposed architectures.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their comments. This work is supported in parts by NSF CNS-1801341 and W911NF-17-1-0311.

REFERENCES

- [1] U.S. Department of Commerce/National Institute of Standards and Technology, "Digital signature standard (DSS)," *Federal Information Processing Standards Publication*, pp. 24–70, 2000.
- [2] D. McGrew, K. Igoe, and M. Salter, "Fundamental elliptic curve cryptography algorithms," *Internet Engineering Task Force (IETF)*, 2011.

- [3] C. Research, "Sec 1: Elliptic curve cryptography," *Standards for Efficient Cryptography*, 2000.
- [4] C. McIvor, M. McLoone, and J. V. McCanny, "Hardware elliptic curve cryptographic processor over GF(p)," *IEEE Trans. on Circuits and Systems*, vol. 53-1, no. 9, pp. 1946–1957, 2006.
- [5] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, Proceedings* (M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, eds.), pp. 207–228, 2006, New York, NY, USA.
- [6] A. Langley, M. Hamburg, and S. Turner, "Elliptic curves for security," *Internet Engineering Task Force (IETF)*, 2016.
- [7] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," *Internet Engineering Task Force (IETF)*, 2017.
- [8] L. Chen, D. Moody, A. Regenscheid, and K. Randall, "Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters," *Computer Security, Draft NIST Special Publication, National Institute of Standards and Technology*, vol. 800-186, 2019.
- [9] P. Sasdrich and T. Güneysu, "Efficient elliptic-curve cryptography using curve25519 on reconfigurable devices," in *Reconfigurable Computing: Architectures, Tools, and Applications - 10th International Symposium, ARC 2014, Proceedings* (D. Goehring, M. D. Santambrogio, J. M. P. Cardoso, and K. Bertels, eds.), pp. 25–36, 2014, Vilamoura, Portugal.
- [10] P. Sasdrich and T. Güneysu, "Implementing curve25519 for side-channel-protected elliptic curve cryptography," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 9, no. 1, pp. 3:1–3:15, 2015.
- [11] P. Sasdrich and T. Güneysu, "Exploring RFC 7748 for hardware implementation: Curve25519 and curve448 with side-channel protection," *J. Hardware and Systems Security*, vol. 2, no. 4, pp. 297–313, 2018.
- [12] P. Koppermann, F. D. Santis, J. Heyszl, and G. Sigl, "X25519 hardware implementation for low-latency applications," in *2016 Euromicro Conference on Digital System Design, DSD 2016* (P. Kitsos, ed.), pp. 99–106, 2016, Limassol, Cyprus.
- [13] P. Koppermann, F. D. Santis, J. Heyszl, and G. Sigl, "Low-latency X25519 hardware implementation: breaking the 100 microseconds barrier," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 52, pp. 491–497, 2017.
- [14] R. Salarifard and S. B. Sarmadi, "An efficient low-latency point-multiplication over curve25519," *IEEE Trans. on Circuits and Systems*, vol. 66-1, no. 10, pp. 3854–3862, 2019.
- [15] F. Turan and I. Verbaauwhede, "Compact and flexible FPGA implementation of ed25519 and X25519," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 3, pp. 24:1–24:21, 2019.
- [16] 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, 2019.
- [17] D. J. Bernstein, "25519 naming," *posting to the cfrg mailing list*, pp. 243–264, 2014.
- [18] *2017 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2017, McLean, VA, USA, May 1-5, 2017*, IEEE Computer Society, 2017.
- [19] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, pp. 243–264, 1987.
- [20] J. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Cryptographic Hardware and Embedded Systems, CHES'99*, pp. 292–302, 1999, Worcester, MA, USA.
- [21] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet physics doklady*, vol. 7, p. 595, 1963.
- [22] S. Ali and M. Cenk, "Faster residue multiplication modulo 521-bit mersenne prime and an application to ECC," *IEEE Trans. on Circuits and Systems*, vol. 65-1, no. 8, pp. 2477–2490, 2018.
- [23] T. B. Preußner, M. Zabel, and R. G. Spallek, "Accelerating computations on FPGA carry chains by operand compaction," in *20th IEEE Symposium on Computer Arithmetic, ARITH 2011* (E. Antelo, D. Hough, and P. Ienne, eds.), pp. 95–102, 2011, Tübingen, Germany.
- [24] W. Schindler and A. Wiemers, "Efficient side-channel attacks on scalar blinding on elliptic curves with special structure," *NIST Workshop on ECC standards*, 2015.
- [25] K. Järvinen, A. Miele, R. Azarderakhsh, and P. Longa, "FourQ on FPGA: new hardware speed records for elliptic curve cryptography over large prime characteristic fields," in *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference*, pp. 517–537, 2016, Santa Barbara, CA, USA.

Optimized Architectures for Elliptic Curve Cryptography over Curve448

Mojtaba Bisheh Niasar¹, Reza Azarderakhsh^{1,2}, and Mehran Mozaffari Kermani³

¹ Department of Computer and Electrical Engineering and Computer Science,
Florida Atlantic University, FL, USA

`{mbishehniasa2019, razarderakhsh}@fau.edu`

² PQSecure Technologies, LLC, Boca Raton, FL, USA

³ Department of Computer Science and Engineering, University of South Florida,
FL, USA
`mehran2@usf.edu`

Abstract. In this paper, we present different implementations of point multiplication over Curve448. Curve448 has recently been recommended by NIST to provide 224-bit security over elliptic curve cryptography. Although implementing high-security cryptosystems should be considered due to recent improvements in cryptanalysis, hardware implementation of Curve448 has been investigated in a few studies. Hence, in this study, we propose three variable-base-point FPGA-based Curve448 implementations, i.e., lightweight, area-time efficient, and high-performance architectures, which aim to be used for different applications. Synthesized on a Xilinx Zynq 7020 FPGA, our proposed high-performance design increases 12% throughput with executing 1,219 point multiplication per second and increases 40% efficiency in terms of required clock cycles $\times$ utilized area compared to the best previous work. Furthermore, the proposed lightweight architecture works in 250 MHz and saves 96% of resources with the same performance. Additionally, our area-time efficient design considers a trade-off between time and required resources, which shows a 48% efficiency improvement with 52% fewer resources. Finally, effective side-channel countermeasures are added to our proposed designs, which also outperform previous works.

Keywords: Curve448, elliptic curve cryptography, FPGA, hardware security, implementation, point multiplication, side-channel

1 Introduction

Elliptic curve cryptography (ECC) has gained prominent attention among asymmetric cryptographic algorithms due to its short key size. ECC is mostly implemented in Internet-of-Thing (IoT) devices considering their limited power resources and processing units. Recently, to address some backdoor issues in ECC constructions due to advances in the strong cryptanalysis and classical attacks, new NIST [1] and IETF [2] recommendations make Curve25519 and Curve448 suitable for higher-level security requirements.

Although we are confident with the security of ECC over prime fields, there is always the possibility that algorithmic improvements reduce the required computation to break ECC. Therefore, moving to a higher level of security will help to keep a margin against unknown attack improvements. However, higher security levels come with the performance penalty and industry often resists them. Hence, we need to provide a level of security that can be feasible subject to the performance requirement of the target application such as high-end servers of constrained devices.

According to Shor’s algorithm [3], most of the current cryptosystems will be broken by quantum computing. Hence, Post-Quantum Cryptography (PQC) algorithm are going to replace the classic public key cryptography algorithms. PQC based on elliptic curves is available for example in [4,5,6]. However, the transition to PQC includes an emerging field called hybrid systems, which require both classic and PQC [7]. Hence, ECC is going to be used in the hybrid mode for maintaining accordance with industry or government regulations, while PQC updates will be applied completely. Therefore, classical cryptosystems cannot be eliminated even if PQC will significantly be developed, so designing high-security ECC is crucial.

As a part of the Transport Layer Security (TLS) [8], Curve448 provides 224-bit security designed by Hamburg in 2015 [9,10]. Moreover, Curve448 belongs to [11], and Safe-Curve policies are considered in its design procedures. Although Curve25519 is highly investigated in recent years for different applications [12,13,14,15], there are few FPGA-based Curve448 implementations due to its optimal software-based design.

The closest related works that can be directly compared to ours are proposed by Sasdrich and Güneysu in [16], and the protected architecture in [13] by adding re-randomization countermeasure to design the resistant scheme against horizontal attacks. In these works, a field arithmetic unit based on schoolbook multiplication is designed by cascading arithmetic and reduction core employing 28 and 5 DSP blocks, respectively. These architectures heavily rely on the DSP blocks which leads to work in high operating frequency (i.e. 357 and 341 MHz). Furthermore, Shah *et al.* in [17] proposed a LUT-based scheme for high-performance point multiplication employing the most significant digit multiplier. Other FPGA implementations of ECC for Montgomery curves in the literature cannot be directly compared to ours, because they target different curves. Ananyi *et al.* in [18] introduced a flexible hardware ECC processor over several NIST prime fields. Furthermore, Alrimeih *et al.* designed an enhanced architecture over different NIST prime fields up to 521-bit in [19] including several countermeasures.

Based on the aforementioned discussions, implementation gaps are identified in: (i) the need for exploration of the different trade-offs between resource utilization and performance considering different optimization goals, and (ii) the lack of employing the Karatsuba-friendly property of Curve448.

Our contributions: To the best of our knowledge, there appear to be extremely few hardware implementations that focus only on Curve448 and make the best of all its features. The main contributions of this work are as follows:

- We Investigate three design strategies to port Curve448 to various platforms with different design goals (i.e., time-constrained, area-constrained, and area-time trade-off applications) using a precise schedule corresponding to each architecture. Hence, different modular multiplication and addition/subtraction modules are developed particularly tailored on a Xilinx Zynq 7020 FPGA to perform variable-base-point point multiplications. Furthermore, all schemes are extended by side-channel countermeasures.
- The proposed architectures are combined with interleaved multiplication and reduction employing redundant number presentation and refined Karatsuba multiplication to increase efficiency in comparison with those presented in previous.
- Our proposed architectures outperform the counterparts available in the literature.

The rest of this paper is organized as follows: In Sec. 2, some relevant mathematical background and side-channel considerations are reviewed. In Sec. 3, the proposed architectures are investigated. In Sec. 4, our proposed FPGA implementations are detailed. In Sec. 5, the results and comparison with other works are discussed. Eventually, we conclude this paper in Sec. 6.

2 Preliminaries

In this section, the mathematical background of ECC will be covered briefly. Additionally, Curve448 and its specifications will be introduced. Then, the respective side-channel analysis attack protection will be described.

2.1 Field Arithmetic and ECDH Key Exchange

The Galois Field $GF(p)$ is described by finite elements including $\{0, 1, \dots, p-1\}$ to define a finite field. Curve448 over $GF(p)$ is defined by $E : y^2 + x^2 \equiv 1 + dx^2y^2 \pmod{p}$ where $p = 2^{448} - 2^{224} - 1$ and $d = -39081$. This curve is a Montgomery curve and also an untwisted Edwards curve which is called Edwards448 [2].

Curve448 specifications can be employed to speed up the elliptic curve Diffie-Hellman (ECDH) Key-Exchange. Using the advantage of $448 = 7 \times 64 = 14 \times 32 = 28 \times 16 = 56 \times 8$ provides more flexibility to design efficient architecture for different platforms. Additionally, due to its Solinas prime with golden ratio $\phi = 2^{224}$, fast Karatsuba multiplication can be performed as follows:

$$\begin{aligned}
 C = A \cdot B &= (a_1\phi + a_0) \cdot (b_1\phi + b_0) \\
 &= a_1b_1\phi^2 + a_0b_0 + ((a_1 + a_0) \cdot (b_1 + b_0) - a_1b_1 - a_0b_0)\phi \\
 &\equiv (a_1b_1 + a_0b_0) + ((a_1 + a_0) \cdot (b_1 + b_0) - a_0b_0)\phi \pmod{p} \quad (1)
 \end{aligned}$$

Table 1. Comparison of different ECC specifications over prime fields

Curve	Ref.	Prime Shape	Security Level	Public Key Size (bits)	Private Key Size (bits)
Curve25519	[2]	$2^{255} - 19$	127-bit	255	255
FourQ	[20]	$2^{127} - 1$	128-bit	2×127	2×127
NIST P-256	[21]	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	128-bit	256	256
BrianpoolP320r1	[22]	Not Specific	160-bit	320	320
NIST P-384	[21]	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$	192-bit	384	384
BrianpoolP384r1	[22]	Not Specific	192-bit	384	384
Secp384r1	[23]	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$	192-bit	384	384
Curve448	[2]	$2^{448} - 2^{224} - 1$	224-bit	448	448
NIST P-521	[21]	$2^{521} - 1$	260-bit	521	521

where $A = (a_1\phi + a_0)$, $B = (b_1\phi + b_0)$, and $A, B, C \in GF(p)$.

To implement modular inversion over $GF(p)$ using Fermat's Little Theorem (FLT), $a^{-1} \equiv a^{p-2} \pmod{p}$ is computed by consecutive operations including 447 squaring and 15 multiplications.

To generate a shared secret key Q between two parties through an insecure channel, i.e., internet, ECDH Key-Exchange protocol can be implemented using elliptic curve point multiplication (ECPM) $Q = k \cdot P$ over Curve448 where k and P are a secret scalar and a known base point, respectively. Moreover, public keys of Curve448 are reasonably short and do not require validation as long as the resulting shared secret is not zero.

Table 1 lists security level, public key size, private key size, and prime shape for security level in the range of about 128 to 256-bit.

2.2 Group Arithmetic and Montgomery Ladder

Scalar multiplication is broken down into 448 iterations considering bit values of k . To perform efficient scalar multiplication over the Montgomery curve, the Montgomery ladder was introduced [24] to perform one point addition (PA) and one point doubling (PD) in each iteration. Furthermore, using this method in projective coordinate increases efficiency. If $P = (x_p, y_p)$ is a base point in affine coordinate, it can be transmitted to projective coordinate such that $P = (X, Y, Z)$ where $x_p = X/Z$ and $y_p = Y/Z$. Suppose $P_1 = (X_1, Z_1)$ and $P_2 = (X_2, Z_2)$ are two points in projective coordinates. Therefore, $P_1 + P_2$ and $2P_1$ are computed by following equations:

$$X_{PD} = (X_1 - Z_1)^2 \cdot (X_1 + Z_1)^2 \quad (2)$$

$$Z_{PD} = 4X_1Z_1 \cdot (X_1^2 + 39081X_1Z_1 + Z_1^2) \quad (3)$$

$$X_{PA} = 4(X_1X_2 - Z_1Z_2)^2 \quad (4)$$

$$Z_{PA} = 4x_p(X_1Z_2 - Z_1X_2)^2 \quad (5)$$

2.3 Side-Channel Protection

To implement a resistant architecture against side-channel analysis (SCA) attacks, different considerations are taken into account. Hence, several countermeasures should be embedded in the cryptographic implementations to prevent information leakage. Two basic protection including (i) constant-time implementation against timing attack, and (ii) secret-independent implementation against simple power analysis (SPA) can be achieved by performing inherently resistant algorithms. Hence, the proposed architecture is resistant against timing attacks and SPA attacks due to performing a constant number of operations in each iteration of the Montgomery ladder and employing constant-time FLT Inversion. Furthermore, some countermeasures which were introduced by Coron [25] are considered to avoid differential power analysis (DPA) attacks. These countermeasures include point randomization and scalar blinding which change both terms in the scalar multiplication $Q = k \cdot P$.

Point Randomization: Point randomization can be achieved by adding a degree of freedom to represent a base point using projective coordinate representations. In this method, the base point $P = (X, Z)$ is projected from affine coordinates using a random value $\lambda \in \mathbb{Z}_{2^{448}} \setminus \{0\}$ such that $P_r = (\lambda \cdot X, \lambda \cdot Z)$. However, the scalar multiplication output is not changed, as proven in (6).

$$x_p = \frac{X}{Z} = \frac{\lambda X}{\lambda Z} \quad (6)$$

Point randomization provides different point representations corresponding to random value λ to avoid any information extraction employing statistical analysis.

Scalar Blinding: The second term in scalar multiplication is randomized in scalar blinding. In this method, multiple group order $\#E$ is added to k such that $k_r = k + r \times \#E$ where r is a random value. According to the fact that adding group order times of base point results in the point in infinity, the correctness of the scalar blinding approach can be proven as follows:

$$k_r \cdot P = (k + r \times \#E) \cdot P = k \cdot P + r \cdot O = k \cdot P \quad (7)$$

This computation takes away data dependency between the swap function in the Montgomery ladder and the corresponding bit in k .

3 Proposed Algorithm and Architecture

In this section, the proposed algorithms for three different performance levels including lightweight, area-time efficient, and high-performance designs are covered. Moreover, their architectures and modular arithmetic units are investigated.

The top-level architecture used in our schemes for different design strategies is illustrated in Fig. 1 composed of three stages: (i) the dedicated controller stage including an FSM and program ROM, (ii) the field arithmetic units in the middle stage including modular addition/subtraction and modular multiplication, and (iii) a memory stage to store the intermediate results. All stages are customized based on the corresponding design strategy for increasing efficiency.

In lightweight and area-time efficient design, a major determinant of efficiency is matching the compute power to the memory bandwidth, so that most cycles the compute units and the memory units are used to accomplish as much work as possible. However, in high-performance design, more resources are utilized to reduce the required time for computation.

Implementing the Karatsuba multiplication requires a complex memory access pattern. Therefore, in lightweight architecture, the product scanning approach is proposed to perform repetitive operations with shared resources. In Design II and III, Karatsuba-based modular multiplication is employed.

3.1 Design I: Lightweight Architecture

The proposed architecture for lightweight design is shown in Fig. 1. This design includes a controller, modular addition/subtraction, modular multiplication, ROM, and dual-port RAM in the top-level architecture. Program ROM consists of 2183 instruction lines (28 lines for Montgomery ladder, 1233 lines for multiplication, and 922 lines for inversion) for modular arithmetic units and memory addresses. Each line of ROM is 15-bit wide: 1 bit for modular multiplication, 2 bits for modular addition, and three 4-bit operand addresses for the RAM. The controller is connected to the ROM to read the instruction code for each cycle. Additionally, the controller sets required addresses for arithmetic units considering the corresponding bit of chosen scalar k and the instruction code. The intermediate data is stored into a dual-port RAM to decrease the required resources. A 448-bit data is stored by splitting to 28 words in 16-bit chunks due to DSP block input size. Therefore, the RAM bit width is 16-bit. To reduce the utilized resources in the proposed lightweight architecture, all required computations are performed sequentially.

Modular Addition/Subtraction: Two 16-bit data are loaded from RAM in each iteration, and addition/subtraction is performed between them. Then, the result is stored into RAM in the next iteration. The carry/borrow should be propagated to the next digit which is shown in the proposed architecture. Thus, modular addition/subtraction takes 56 cycles of which 28 are required for the computation and the remaining ones for reading/storing data. Moreover, the last carry/borrow is considered for the reduction procedure. To design a constant time computation, addition/subtraction is interleaved by a reduction based on the fact that $2^{448} \equiv 2^{224} + 1 \pmod{p}$. Eventually, this computation requires another 56 iterations to complete for reduction.

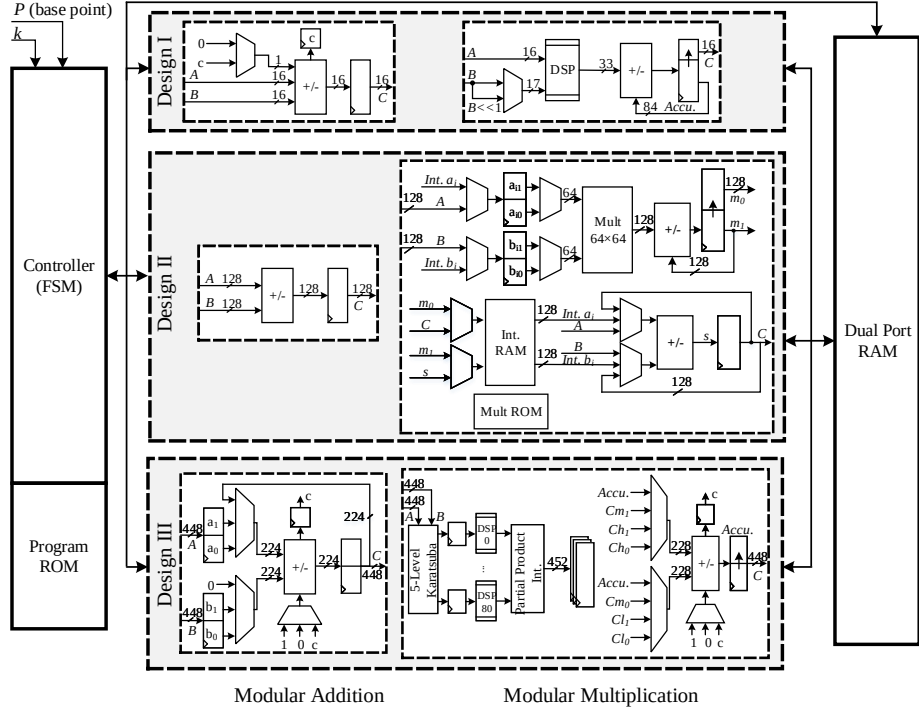


Fig. 1. Proposed hardware architecture for different performance levels over Curve448: (Design I) The arithmetic units architecture in the lightweight design with 16-bit datapath, (Design II) The arithmetic units architecture in the area-time efficient design with 128-bit datapath, and (Design III) The arithmetic units architecture in the high-performance design with 448-bit datapath. A and B are read from the memory unit, and C are stored in the memory unit.

Modular Multiplication: Modular multiplication is designed based on the product scanning approach. To perform this approach, 28 multiplications are required for each 16-bit digit. In [16], the authors introduced a schoolbook multiplication with interleaved reduction performed in parallel using 28 DSPs. In lightweight architecture, only one DSP is utilized to perform multiplication sequentially. Due to the proposed interleaved reduction, different numbers of multiplication are performed corresponding to each product digit. Besides, several multiplications are duplicated for some operands. Thus, considering 1 bit free of DSP input, one bit shifted of the second operand can be selected to accelerate computation. For example, the fourteenth digit of the final product requires 54 16×16 -bit multiplications of which 26 are duplicated. Hence, they can be performed in 28 16×16 -bit and 13 17×16 -bit multiplications. The partial product is accumulated in a register shown by $Accu_{i+1} \times 2^{16} + C_i$ where $Accu_i$ and C_i are carry and product digit of i^{th} element. The C_i is stored in the RAM, and the

$Accu_i$ is shifted 16 bits to propagate to the next product digit. However, $Accu_{28}$ should be considered for the second stage of reduction.

In the proposed lightweight design, each multiplication is followed by modular addition to implement an efficient reduction algorithm. Due to the special form of the prime, the addition between $Accu_{28} \times 2^{14 \times 16} + Accu_{28}$ and $C_{27} \times 2^{27 \times 16} + \dots + C_0$ leads to modular multiplication result.

3.2 Design II: Area-Time Efficient Architecture

The proposed area-time efficient design is shown in Fig. 1. This scheme is derived from our previous work presented in [26]. Program ROM consists of 480 instruction lines (18 lines for Montgomery ladder and 462 lines for inversion) for modular arithmetic units and memory addresses. Each line of ROM is 15-bit wide: 1 bit for modular multiplication, 2 bits for modular addition, and three 4-bit operand addresses for the RAM. For efficient resource utilization, the multiplication is performed based on a dedicated ROM consists of 69 lines of 17-bit width to generate the control commands.

The most important advantage of special form p is the implementation of the fast Karatsuba multiplication based on (1) [9]. In an area-time efficient design, a 448×448 -bit multiplication in the first level can be converted by the Karatsuba method [27] to three 225×255 -bit multiplications considering the addition carry. Furthermore, Bernstein in [28] introduced the refined Karatsuba identity as follows:

$$(a_0 + a_1 t^n) \cdot (b_0 + b_1 t^n) = (1 - t^n) \cdot (a_0 b_0 - t^n a_1 b_1) + t^n (a_0 + a_1) \cdot (b_0 + b_1) \quad (8)$$

A 448-bit data is represented in four integer pieces in radix $2^{448/4} = 2^{112}$, i.e., A is written as $A = a_0 + a_1 \times 2^{112} + a_2 \times 2^{224} + a_3 \times 2^{336}$, and B is decomposed similarly to A . Applying Karatsuba multiplication provides $A_0 B_0$, $A_1 B_1$, and $A_{10} B_{10}$ such that:

$$A \times B = (A_0 B_0 + A_1 B_1) + 2^{224} (A_{10} B_{10} - A_0 B_0) \quad (9)$$

where:

$$A_0 B_0 = (a_0 + a_1 \times 2^{112}) \cdot (b_0 + b_1 \times 2^{112}) \quad (10)$$

$$A_1 B_1 = (a_2 + a_3 \times 2^{112}) \cdot (b_2 + b_3 \times 2^{112}) \quad (11)$$

$$A_{10} B_{10} = (a_0 + a_2 + (a_1 + a_3) \times 2^{112}) \cdot (b_0 + b_2 + (b_1 + b_3) \times 2^{112}) \quad (12)$$

For the second level, refined Karatsuba identity is applied where t^n is equal to 2^{112} . Hence, for example, $A_0 B_0$ is decomposed such that:

$$\begin{aligned} A_0 B_0 &= (1 - 2^{112}) \cdot (a_0 b_0 - 2^{112} a_1 b_1) + 2^{112} (a_0 + a_1) \cdot (b_0 + b_1) \\ &= (1 - 2^{112}) \cdot (a_0 b_0 - 2^{112} a_1 b_1) + 2^{112} (a_{10} b_{10}) \end{aligned} \quad (13)$$

Therefore, three 114×114 -bit multiplications are required (i.e., a_0b_0 , a_1b_1 , and $a_{10}b_{10}$) which are implemented by schoolbook method with 64-bit digit size.

To diminish the cost of the carry propagation, redundant representation is employed in the proposed architecture. Datapath is considered 128-bit to provide a more 16 bits for each digit. Moreover, in [29], the authors used reduced refined Karatsuba technique in top-level to decrease the number of additions. Based on [26], we employ the interleaved reduction technique into the first Karatsuba level. Hence, we reduce the second term in (9), i.e., $2^{224} \times (A_{10}B_{10} - A_0B_0)$, before adding with $(A_0B_0 + A_1B_1)$.

Addition/Subtraction: In the area-time efficient architecture, addition/subtraction is designed non-modular. Based on the Montgomery ladder, addition/subtraction units are followed by multiplication. Due to redundant representation in this architecture, the reduction stage can be postponed to the multiplication stage. Moreover, carry propagation is removed between digits considering redundancy in representation. Addition/subtraction is computed in four cycles between two 128-bit data in parallel with multiplication. Thus, its latency is absorbed by multiplication cycles to increase performance.

Modular Multiplication: Modular multiplication includes a 64×64 -bit multiplication using 16 DSPs. The multiplier is designed pipelined with 8 stages to increase throughput. Two 128-bit data are stored in its input registers. Furthermore, a register is considered in its output to accumulate partial products. The throughput of the proposed architecture is 1 multiplication per 4 cycles employing the pipeline architecture. Hence, in $3 \times 4 = 12$ clock cycles, three required multiplications in (13), i.e., a_1b_1 , a_0b_0 , and $a_{10}b_{10}$, are computed.

The proposed middle-level recombination is shown in Fig. 2-(a). After computing the three lowest-level products, i.e., a_0b_0 , a_1b_1 , and $a_{10}b_{10}$, we need only 5 additions performed in 3 steps to compute A_0B_0 which are pipelined with A_1B_1 computing. We obtain A_0B_0 as follows:

- *Step 1: Compute $s_1 = (a_0b_0 - 2^{112}a_1b_1)$.* We subtract the first digit a_1b_1 from the second digit a_0b_0 . Note that the second digit a_1b_1 is not actually subtracted from 0. Hence, it should be implicitly negated during the recombination.
- *Step 2: Compute $s_2 = (1 - 2^{112})s_1$.* Multiplying s_1 by $(1 - 2^{112})$ is equivalent to subtracting the shifted s_1 from itself. Therefore, we align the first digit of s_1 with the second digit and subtract.
- *Step 3: Compute $s_3 = s_2 + 2^{112}(a_{10}b_{10})$.* We align the first digit of $a_{10}b_{10}$ with the second digit of s_2 . Hence, this level of computation is finished by performing two additions between the first and second digits of $a_{10}b_{10}$, and the second and third digits of s_2 , respectively.

Moreover, A_1B_1 and $A_{10}B_{10}$ can be computed in the same way as A_0B_0 . The total cost for these three steps is 15 128-bit additions.

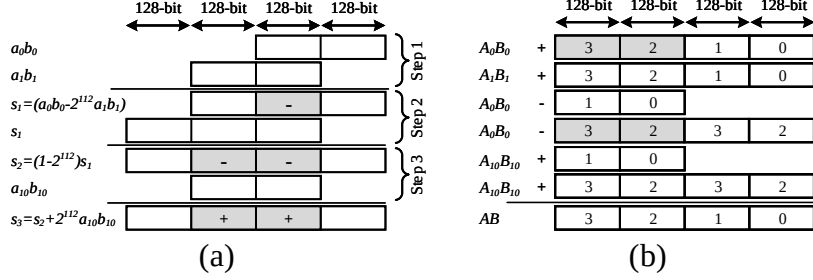


Fig. 2. Proposed recombination in efficient modular multiplication. (a) middle-level recombination: only five highlighted operations are performed. (b) top-level recombination: applying interleaved reduction cancels the highlighted digits.

Fig. 2-(b) illustrates the top-level recombination. According to this figure, the two most significant digits of A_0B_0 , i.e., the third and fourth digits, cancel themselves considering the reduction algorithm. Then, several consecutive additions between partial products for the carry propagation and reduction are performed.

For efficient control of the operation sequences, a dedicated controller and an internal RAM are considered in this unit. Eventually, modular multiplication latency is 69 clock cycles employing the pipelined architecture.

3.3 Design III: High-Performance Architecture

The proposed high-performance design is shown in Fig. 1. Program ROM consists of 1554 instruction lines (168 lines for Montgomery ladder and 1386 lines for inversion) for modular arithmetic units and memory addresses. Each line of ROM is 12-bit wide: 1 bit for modular multiplication, 2 bits for modular addition, 1 bit for storing data on RAM, and two 4-bit memory addresses for the RAM

There are different multiplication methods for designing high-performance architecture, e.g., Schoolbook or Toom-3 approach. For Curve448, Karatsuba multiplication has a special advantage due to its golden ratio. Different levels of Karatsuba multiplication can be applied in 448×448 -bit multiplication. In this design, 5-level Karatsuba multiplication is considered to decrease multiplier latency considering DPS block input size. Although increasing the number of Karatsuba level drops operating frequency due to expanding addition tree, parallelization in the lowest level of computation reduces considerably the required cycles to speed up the computation. According to this approach, high-performance architecture is designed which is illustrated in Fig. 1. This architecture includes a controller, modular addition/subtraction, modular multiplication, ROM, and dual-port RAM. To decrease loading/storing latency from/to RAM, datapath is implemented with 448-bit wide.

Modular Addition/Subtraction: Addition/subtraction is implemented between two 224-bit to decrease critical path delay (CPD). The data is loaded from RAM and stored in its input registers. Then, $a_0 + b_0$ and $a_1 + b_1$ are performed and the results are stored in a register, while the first carry/borrow is propagated and the second one is considered for reduction. Then, an interleaved reduction follows the addition/subtraction operation. Therefore, multiplexers are designed for selecting operands, carry/borrow bit, and inputs. Finally, although modular addition/subtraction requires 4 clock cycles for operation and 3 cycles for data transmission, its latency is entirely absorbed by modular multiplication.

Modular Multiplication: Modular multiplication splits a 448×448 -bit multiplication into $3^5 = 243$ partial multiplications employing 5 levels of Karatsuba. Due to the number of available DSP blocks in the proposed FPGA (i.e., 220 DSPs), the first level is designed pipeline, while the rest of the multiplications are performed parallel using 81 DSP blocks. Then, an addition tree is implemented, and its achieved results are stored into three registers including $Ch = A_1B_1$, $Cl = A_0B_0$, and $Cm = A_{10}B_{10}$. There are registers between different stages of multiplier and addition tree to design pipeline architecture. Therefore, a next multiplication (i.e., A_0B_0) is exactly started after that the first stage of previous multiplication (i.e., A_1B_1) is completed. Thus, all three registers are filled in 5 clock cycles with 452-bit wide.

An addition/subtraction circuit is build to compute the top-level addition and interleaved reduction. Therefore, computing $C_0 = Ch_0 + Cl_0 + Cm_1 - Cl_1$ and $C_1 = Ch_1 + Cl_1 + Cm_0 - Cl_0$ leads to the proposed result. Multiplexers determine inputs, carry/borrow propagation, and operands in each iteration. Eventually, the last reduction stage is computed considering C_1 carry. This procedure takes 9 cycles. However, its first stages are arranged with the last stages of multiplication. Thus, modular multiplication is completed in 15 cycles considering RAM interfacing.

4 FPGA Implementations

In this section, we present details of the proposed ECPM, in particular the Montgomery ladder, its scheduling, and latency. Second, SCA considerations for implementing enhanced architecture are discussed.

4.1 ECPM implementation

Point multiplication is implemented employing field arithmetic units including modular addition, subtraction, multiplication, and inversion. Modular addition, modular subtraction, and modular multiplication are entirely explained corresponding to each optimization goals. Furthermore, modular inversion is implemented based on the FLT constant-time algorithm using consecutive modular multiplications performed after point multiplication.

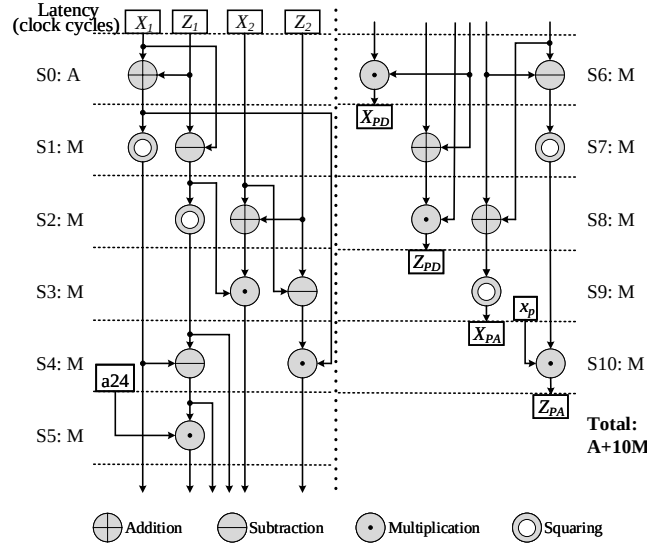


Fig. 3. Design II and III data dependency diagram for one step Montgomery ladder execution over Curve448. The total latency is $A + 10M$ where A and M are modular addition and multiplication latency, respectively. Furthermore, a_{24} is equal to 39081 for Curve448.

Montgomery ladder is implemented using precise scheduling of arithmetic units. It consists of 8 additions (or subtraction) and 10 multiplications (or squaring). Fig. 3 shows the data dependency diagram for executing a step Montgomery ladder in area-time efficient and high-performance architectures. As one can see, the total latency is $A + 10M$ where A and M are modular addition and multiplication latency, respectively. In other words, addition latency is absorbed by multiplication employing parallel computing. However, in our proposed lightweight architecture each multiplication is followed by an additional addition to perform reduction which leads to an increase in its latency.

4.2 Side-Channel Protected Implementation

Different SCA countermeasures are embedded for the proposed designs to provide enhanced architecture against DPA. As it is explained earlier, three different approaches are considered including scalar blinding, base point randomization, and continuous point randomization. Designing a random number generator (RNG) is not in the scope of this study, so we assume the randomized numbers are provided externally.

Scalar Blinding: According to [30], at least half of the field size is recommended to be used for scalar blinding. Therefore, r with 224-bit length builds k_r with 672-bit which makes longer execution time compared to unprotected design.

However, it does not change resource utilization considering externally provided k_r assumption.

Base Point Randomization: Base point randomization is achieved by using randomized base point $P_r = (\lambda \cdot x_p, \lambda)$. We assume P_r is externally delivered to the ECPM core. Due to implementing variable-base-point architecture, base point randomization is achieved without any cost.

Continues Point Randomization: Although exploiting of randomized base point provides more resistance against DPA, point randomization can also be applied consecutively to prevent horizontal attacks. In this approach, two more modular multiplications (i.e., $\lambda \cdot x_{PA}$ and $\lambda \cdot z_{PA}$) are performed to re-randomized Montgomery ladder outputs. Hence, continuous point randomization increases Montgomery ladder latency, and consequently total latency.

5 Implementation Results and Comparison

Our three different performance level architectures are synthesized and implemented with Xilinx Vivado 2019.2 on a Xilinx Zynq-7020 FPGA device. All given results are obtained after place-and-route (PAR). Furthermore, several test vectors have been tested to verify our implementations, and the results for suggested vectors by [2] are reported in Appendix A.

5.1 Performance Results

Table 2 summarizes the number of clock cycles and latency requirements for all proposed architectures in our unprotected schemes broken down into the group and field arithmetic operations. According to this table, the operating frequency for high-performance design compared to other schemes is dropped as expected to 95 MHz due to the increasing level of Karatsuba multiplication. As described before, the most time-consuming operation in Design III is modular multiplication with 15 clock cycles which absorb addition/subtraction latency. Furthermore, modular inversion is implemented in the last stage of ECPM for back transformation from projective coordinates to affine coordinates at the cost of about 10% of the clock cycles. In total, the unprotected core can execute 35, 363, and 1,219 ECDH operations per second for lightweight, area-efficient, and high-performance architectures, respectively.

5.2 Comparison

Recently, there are considerable efforts to implement ECC using instruction set processors [31]. It makes the design smaller, slower, and more controllable/programmable at cost of implementing a software-based processor. However, pure hardware implementations are better as it is faster and could be any time integrated with hardware/software co-design techniques.

Table 2. FPGA implementation results for different performance level architectures in terms of clock cycles and latency requirements

Proposed Architecture		Design I	Design II	Design III
Clock Freq. [MHz]		250	127	95
$\mathbb{F}_p$ Operation	Addition	112	8	7
	Subtraction	112	8	7
	Multiplication	1,233	69	15
	Inversion	620,047	32,272	6,917
Unprotected	Mont. step	14,346	708	158
	Point Multiplication	7,047,055	349,546	77,702
	Latency [ms]	28.19	2.75	0.82

Additionally, our target is optimization based on the area-time trade-offs point of view, while other trade-offs from a time-memory perspective can be investigated. However, the latter method relies on the pre-computation over a fixed-base-point, while we focus on variable-base-point architectures.

To the best of our knowledge, there are only a few FPGA implementations for security level more than 128-bit. For comparison, we have selected point multiplications presented in [16] and [13] with the same platform, and [17] mapped to a Xilinx Virtex FPGA. Additionally, we have opted [19] and [18] for NIST P-384 and P-521, since they are in the same security level as Curve448. Moreover, to estimate area equivalent for efficiency computing, we force the Vivado synthesis tool to implement DSP block employing only LUTs, so we investigate each DSP is equivalent to almost 100 occupied Slices.

The area and time specifications for our proposed work and comparison to the counterparts are reported in Table 3. Three different levels of protection are considered including no protection, protection with scalar blinding and point randomization, and high protection with scalar blinding and point re-randomization countermeasures.

Design I requires only a few resources on our target device, i.e., 137 (1%) of the Slices, 1 (0.5%) of DSP block, and 2 (1.4%) BlockRAM. It occupies at least 96% fewer resources compared to other works while decreasing utilized resources increases latency by more than 7 million cycles.

Design II utilizes the available resources more than Design I and less than Design III, i.e., 760 of the Slices, 16 of DSPs, and 9 of BlockRAM. Our proposed Design II saves 52% of Slice utilization compared to the unprotected scheme in [16]. Furthermore, the critical resource in FPGA is the DSP component which has a significant effect on performance. So, the proposed architecture for an area-time efficient scheme saves 52% of DSP utilization at the cost of longer execution time compared to [16] and [13], respectively. Thus, this design can be considered for some applications which need a trade-off between area and time. Moreover, the protected scheme of Design II saves 60%, 52%, and 36% Slices, DSPs, and BRAM compared to the high-protected scheme presented in [13], while the total time is $3\times$ slower.

Table 3. FPGA implementation results for different ECPM cores over prime fields above 128-bit security

Work	SCA ⁴	Area					Time			
		LUTs	FFs	Slices	DSPs	BRAMs	Latency [CCs]	Freq [MHz]	Total time [ms]	OP/s
NIST P-384 (192-bit security)										
[19] ¹	(+)	32900	~	11,200	289	128	~	100	1.18	847
[18] ²	(-)	31946	~	20,793	32	1	~	60	17.50	57
NIST P-521 (260-bit security)										
[19] ¹	(+)	32900	~	11,200	289	128	~	100	1.60	625
[18] ²	(-)	31946	~	20,793	32	1	~	60	39.90	25
Curve448 (224-bit security)										
[16] ³	(-)	2,555	7,049	1,580	33	14	328,286	357	0.92	1,087
	(+)	3,583	7,423	1,648	35	14	473,926	335	1.41	708
[13] ³	(+)	4,624	8,209	1,985	33	14	499,344	341	1.46	685
	(++)	~	~	2,056	33	14	547,728	341	1.61	622
[17] ⁴	(-)	50,143	~	~	0	~	372,742	325	1.15	869
Design I ³	(-)	321	174	137	1	2	7,047,055	250	28.19	35
	(++)	509	438	203	1	2	12,068,239	250	48.27	21
Design II ³	(-)	2,233	1,152	760	16	9	349,546	127	2.75	363
	(++)	2,587	1,629	842	16	9	602,801	127	4.75	211
Design III ³	(-)	13,132	4,035	4,354	81	0	77,702	95	0.82	1,219
	(++)	13,415	4,610	4,424	81	0	133,254	95	1.40	713

¹ Platform: XC6VLX760 ² Platform: XC4VFX100 ³ Platform: XC7Z7020 ⁴ Platform: Virtex-7

⁴(-): no protection, (+): scalar blinding and point randomization countermeasures, (++): scalar blinding and point re-randomization countermeasures

Design III is implemented employing 4,354 of the Slices, 81 DSPs, and no BlockRAM. It is designed based on 5-level Karatsuba multiplication employing the distributed memory to reduce total latency. Design III generates 1,219 ECPM per second which leads to 76% and 12% improvement in terms of the required clock cycles and throughput compared to [16], respectively. Besides, its protected scheme improves 76% and 15% of the required clock cycles and throughput compared to the high-protected scheme in [13] at the cost of utilizing $2.1\times$ and $2.5\times$ more Slices and DSPs, respectively.

We also report NIST P-384 and NIST P-521 hardware implementations. Most of these results are hard to compare since the implementations differ in various ways. According to Table 3, [18] in both field size 384 and 521-bit utilizes almost 20,800 Slices and 32 DSPs on a Virtex4 FPGA (2LUTs per Slice) to generate 57 and 25 ECPM per second. Moreover, [19] uses 11,200 Slices and 289 DSPs to

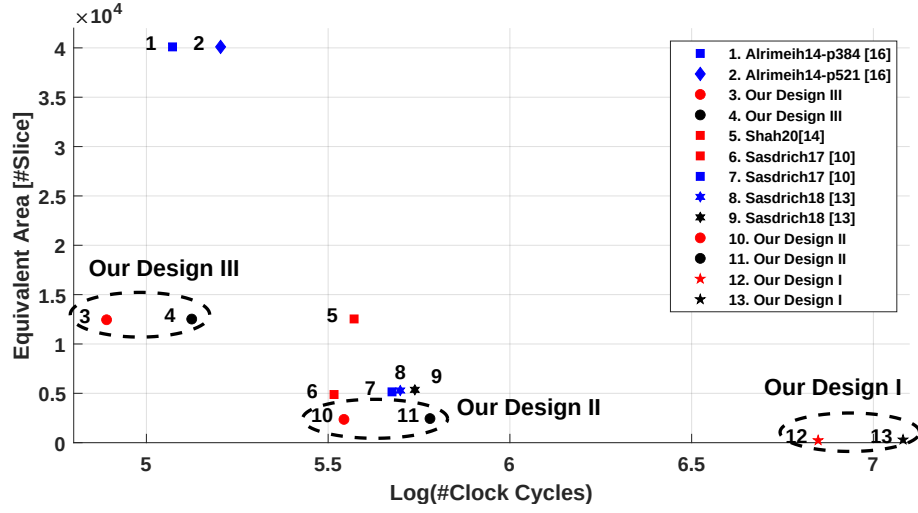


Fig. 4. Efficiency comparison between FPGA-based ECPM architectures in terms of $A \cdot T$ (Area $\times$ Time). Area is considered as *Slices + DSPs equivalent*, while each DSP is assumed equivalent to 100 Slices based on the Vivado synthesis tool. A fixed low-frequency is considered for all architectures, hence, the required *clock cycle* is used rather than time. It should be notified that a low-frequency is employed for IoT devices to reduce power consumption. (red: unprotected, blue: protected, black: high-protected)

compute 847 and 625 ECPM per second over 384 and 521-bit NIST prime on a Virtex-6 FPGA, respectively.

It is worth mentioning that the aforementioned references (i.e., [16] and [13]) take advantage of the high operating frequency, thanks to designing a novel architecture with the optimized critical path. They implement an arithmetic core and a reduction core using pipelined DSPs with different configurations corresponding to the required operations.

One of the most important challenges in developing IoT devices is reducing power consumption due to their resource constraints. Although area utilization has an undeniable role to decrease power consumption, increasing operating frequency grows the required dynamic power. Hence, considering a fixed low-frequency to reduce the required power, we compare the efficiency of our proposed architectures to other works in terms of $A \cdot T$ (Area $\times$ Time). Fig. 4 illustrates the efficiency of different FPGA-based point multiplication architectures. According to this figure, our proposed Design I utilizes minimum resources. Our proposed Design III achieves the lowest latency, while more resources are employed compared to [16,13]. Eventually, the proposed Design II stands between these two designs with better performance in comparison with the previous works.

According to the target platform each scheme can be implemented which addresses the restrictions of the system. The proposed architecture in [16] and

[13] result in $3\times$ speedup compared to our Design II, when they work in 357 and 341 MHz, respectively. However, a more technology-independent measurement is the required cycle. An operating frequency in a limited range is mostly considered to reduce the required power. Thus, efficiency can be computed by the required clock cycles $\times$ area. In this case, the efficiency of the proposed lightweight design is almost equal to the best previous work, while unprotected and high-protected Design II improve 48% and 50% efficiency compared to [16] and [13], respectively. Furthermore, Design III increases efficiency by 40% and 43% compared to [16] and [13], respectively.

6 Conclusion

In this work, three hardware implementations for security level 224-bit are presented for area-constrained, time-constrained, and area and time trade-off applications. All proposed architectures are implemented on a mid-range Xilinx FPGA XC7Z7020, while side-channel countermeasures are simultaneously investigated. We choose efficient techniques for each application, which include interleaved reduction, consecutive Karatsuba multiplication, refined Karatsuba identity, and pipelined architecture. These architectures can compute 1219, 363, and 35 ECDH operations per second for high-performance, area-time efficient, and lightweight designs, respectively.

Acknowledgment

The authors would like to thank the reviewers for their comments. Also, we thank Mike Hamburg for his constructive comments. This work is supported by a grant from ARO W911NF-17-1-0311.

References

1. Chen, L., Moody, D., Regenscheid, A., Randall, K.: Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. Computer Security, Draft NIST Special Publication, National Institute of Standards and Technology 800-186, (2019)
2. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security (2016)
3. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. 124–134 (1994)
4. Elkhatib, R., Azarderakhsh, R., Mozaffari Kermani, M.: Highly optimized montgomery multiplier for SIKE primes on FPGA. In: 27th IEEE Symposium on Computer Arithmetic, ARITH 2020, Portland, OR, USA, June 7-10, 2020. 64–71 (2020)

5. Seo, H., Sanal, P., Jalali, A., Azarderakhsh, R.: Optimized implementation of SIKE round 2 on 64-bit ARM cortex-a processors. *IEEE Trans. Circuits Syst. I Regul. Pap.* 67-I(8), 2659–2671 (2020)
6. Seo, H., Anastasova, M., Jalali, A., Azarderakhsh, R.: Supersingular isogeny key encapsulation (SIKE) round 2 on ARM cortex-m4. *IACR Cryptol. ePrint Arch.* 2020, 410 (2020)
7. Bindel, N., Herath, U., McKague, M., Stebila, D.: Transitioning to a quantum-resistant public key infrastructure. *IACR Cryptology ePrint Archive* 2017, 460
8. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. *RFC* 8446 (2018)
9. Hamburg, M.: Ed448-goldilocks, a new elliptic curve. *IACR Cryptology ePrint Archive* 2015, 625 (2015)
10. Mike Hamburg: Ed448-goldilocks, a new high-strength curve and implementation. URL: <https://csrc.nist.gov/csrc/media/events/workshop-on-elliptic-curve-cryptography-standards/documents/presentations/session7-hamburg-michael.pdf> (June 2015)
11. Bernstein, D.J., Lange, T.: Safecurves: choosing safe curves for elliptic-curve cryptography. URL: <https://safecurves.cr.yp.to/>. (2016)
12. Sasdrich, P., Güneysu, T.: Efficient elliptic-curve cryptography using curve25519 on reconfigurable devices. In Goehringer, D., Santambrogio, M.D., Cardoso, J.M.P., Bertels, K., eds.: *Reconfigurable Computing: Architectures, Tools, and Applications - 10th International Symposium, ARC 2014, Proceedings.* 25–36 (2014, Vilamoura, Portugal)
13. Sasdrich, P., Güneysu, T.: Exploring RFC 7748 for hardware implementation: Curve25519 and curve448 with side-channel protection. *J. Hardware and Systems Security* 2(4), 297–313 (2018)
14. Bisheh Niasar, M., Elkhatab, R., Azarderakhsh, R., Mozaffari Kermani, M.: Fast, small, and area-time efficient architectures for key-exchange on curve25519. In: *27th IEEE Symposium on Computer Arithmetic, ARITH 2020, Portland, OR, USA, June 7–10, 2020.* 72–79 (2020)
15. Salarifard, R., Sarmadi, S.B.: An efficient low-latency point-multiplication over curve25519. *IEEE Trans. on Circuits and Systems* 66-I(10), 3854–3862 (2019)
16. Sasdrich, P., Güneysu, T.: Cryptography for next generation TLS: implementing the RFC 7748 elliptic curve448 cryptosystem in hardware. In: *Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, June 18–22, 2017.* 16:1–16:6 (2017)
17. Shah, Y.A., Javeed, K., Shehzad, M.I., Azmat, S.: Lut-based high-speed point multiplier for goldilocks-curve448. *IET Computers Digital Techniques* 14(4), 149–157 (2020)
18. Ananyi, K., Alrimeih, H., Rakhmatov, D.: Flexible hardware processor for elliptic curve cryptography over nist prime fields. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17(8), 1099–1112 (2009)
19. Alrimeih, H., Rakhmatov, D.: Fast and flexible hardware support for ecc over multiple standard prime fields. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22(12), 2661–2674 (2014)
20. Järvinen, K., Miele, A., Azarderakhsh, R., Longa, P.: FourQ on FPGA: new hardware speed records for elliptic curve cryptography over large prime characteristic fields. In Gierlichs, B., Poschmann, A.Y., eds.: *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Proceedings.* 517–537 (2016, Santa Barbara, CA, USA)

21. U.S. Department of Commerce/National Institute of Standards and Technology: Digital signature standard (DSS) (2013)
22. Lochter, M., Merkle, J.: Elliptic curve cryptography (ecc) brainpool standard curves and curve generation (2010)
23. Brown, D.R.L.: Sec 2: Recommended elliptic curve domain parameters (2010)
24. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation* 48, 243–264 (1987)
25. Coron, J.: Resistance against differential power analysis for elliptic curve cryptosystems. In Koç, Ç.K., Paar, C., eds.: *Cryptographic Hardware and Embedded Systems, CHES'99*. 292–302 (1999, Worcester, MA, USA)
26. Bisheh Niasar, M., Azarderakhsh, R., Mozaffari Kermani, M.: EdDSA for hardware implementation: Highly optimized Ed25519 and Ed448 signatures. Accepted in *IEEE Transactions on Circuits and Systems I: Regular Papers*, (2020)
27. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. *Soviet physics doklady* 7, 595 (1963)
28. Bernstein, D.J.: Batch binary edwards. In: *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*. 317–336 (2009)
29. Bernstein, D.J., Chuengsatiansup, C., Lange, T.: Curve41417: Karatsuba revisited. In: *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*. 316–334 (2014)
30. Schindler, W., Wiemers, A.: Efficient side-channel attacks on scalar blinding on elliptic curves with special structure. *NIST Workshop on ECC standards*, (2015)
31. Ogawa, H.S., Luther, T.E., Ricardini, J.E., Cunha, H., Jr., M.S., Aranha, D.F., Derwig, R., Kupwade-Patil, H.: Accelerated v2x provisioning with extensible processor platform. *Cryptology ePrint Archive, Report 2019/1039* (2019)

Curve448 on 32-bit ARM Cortex-M4

Hwajeong Seo¹[0000–0003–0069–9061] and Reza Azarderakhsh^{2,3}

¹Division of IT Convergence Engineering, Hansung University, Seoul 136-792, Korea,
hwajeong84@gmail.com

²Department of Computer and Electrical Engineering and Computer Science,
Florida Atlantic University, FL, USA,
razarderakhsh@fau.edu

³PQSecure Technologies, LLC

Abstract. Public key cryptography is widely used in key exchange and digital signature protocols. Public key cryptography requires expensive primitive operations, such as finite-field and group operations. These finite-field and group operations require a number of clock cycles to execute. By carefully optimizing these primitive operations, public key cryptography can be performed with reasonably fast execution timing. In this paper, we present the new implementation result of Curve448 on 32-bit ARM Cortex-M4 microcontrollers. We adopted state-of-art implementation methods, and some previous methods were re-designed to fully utilize the features of the target microcontrollers. The implementation was also performed with constant timing by utilizing the features of microcontrollers and algorithms. Finally, the scalar multiplication of Curve448 on 32-bit ARM Cortex-M4@168MHz microcontrollers requires 6,285,904 clock cycles. To the best of our knowledge, this is the first optimized implementation of Curve448 on 32-bit ARM Cortex-M4 microcontrollers. The result is also compared with other ECC and post-quantum cryptography (PQC) implementations. The proposed ECC and the-state-of-art PQC results show the practical usage of hybrid post-quantum TLS on the target processor.

Keywords: ARM Cortex-M4 · Curve448 · Public Key Cryptography · Hybrid Post-Quantum TLS

1 Introduction

Public key cryptography is widely used in key exchange and digital signature protocols. For public key cryptography, implementation is a challenge with low-end microcontrollers, which have the disadvantages of low energy, performance, and memory. In particular, the efficiency of elliptic curve cryptography (ECC) depends on the compact implementation of finite-field arithmetic and group operation. For this reason, the optimized implementation of finite-field arithmetic and group operation should be considered. In this paper, we present the first Curve448 implementation result on 32-bit ARM Cortex-M4 microcontrollers. The motivations of this work may be summarized as follows:

- Curve448 offers 224-bit security and is designed for use with the elliptic curve Diffie-Hellman (ECDH) key agreement scheme [1]. The curve was favored by the Internet Research Task Force Crypto Forum Research Group (IRTF CFRG) for inclusion in transport layer security (TLS) standards along with Curve25519. The curve is an approved elliptic curve for use by the US federal government, which is confirmed in FIPS 186-5. However, the implementation of algorithms has not been actively conducted. This work fills this gap.
- The target microcontroller, namely the 32-bit ARM Cortex-M4, is the most widely used in practice because it has relatively powerful computation abilities in terms of the arithmetic logic unit (ALU), frequency of the CPU, RAM, and ROM in comparison to legacy embedded processors, such as 8-bit AVR ATmega and 16-bit MSP430(X) microcontrollers. Furthermore, the NIST recommended this board for evaluation of post-quantum cryptography (PQC). For this reason, a number of cryptographic implementations have been recently done over 32-bit ARM Cortex-M4 microcontrollers [2–5]. However, Curve448 had not been implemented on this target microcontroller. This work evaluated Curve448 on ARM Cortex-M4 microcontrollers for the first time.

For high performance, we adopted state-of-art implementation methods and some previous methods were re-designed to fully utilize the features of the target microcontrollers. This was the first implementation of Curve448 on this target processor. The result was compared with those of other 128-bit security ECC implementations. The scalar multiplication of Curve448 on 32-bit ARM Cortex-M4@168MHz microcontrollers requires 6,285,904 clock cycles. The result shows that Curve448 is reasonably fast enough on the target microcontroller. The result was also compared with other PQC implementations. This shows the practical usage of hybrid post-quantum TLS on the target processor is available.

1.1 Contribution

Detailed contributions are as follows:

First implementation of Curve448 on 32-bit ARM Cortex-M4 This paper presents the first implementation of Curve448 on 32-bit ARM Cortex-M4 processors. State-of-art techniques were applied to improve the performance. The result shows that the implementation is practically fast enough.

Secure and efficient implementation of primitive operations All primitive operations such as finite-field arithmetic and group operation were implemented in a secure and efficient way. By using constant and regular implementation, the timing attack was efficiently prevented. Furthermore, cache attack were prevented by avoiding the pre-computed value access. All requirements for constant timing on ARM Cortex-M4 specifically are also presented for interested cryptographic researchers.

In-depth comparison of pre-quantum and post-quantum cryptography

We compared pre-quantum and post quantum cryptography on the target processors. The performance report shows the availability of hybrid post-quantum TLS. Furthermore, we discuss the trade-off between performance and security in detail.

First Curve448 on ARM Cortex-M4 as an open source The implementation will be public domain after publication. The source code will be a helpful resource for researchers.

The remainder of this paper is organized as follows. In Section 2, we introduce the target curve (Curve448), the target microcontroller (32-bit ARM Cortex-M4), and previous implementations. The optimized implementation techniques for Curve448 on 32-bit ARM Cortex-M4 are presented in Section 3. In Sections 4, we evaluate and compare implementation results. Finally, we conclude the paper in Section 6.

2 Related Works

In this section, we introduce the target curve (Curve448), target microcontroller (32-bit ARM Cortex-M4), and previous implementations.

2.1 Target Curve: Curve448

Edwards curves, which were suggested in [6] provide complete addition formulas, which does not have a case (division by zero). The one proper Edwards curve for cryptography is Curve448–Goldilocks, which is faster and simpler than traditional NIST curves [1]. Curve448–Goldilocks provides high-security level (224-bit) and the related equation is as follows:

$$E : y^2 + x^2 = 1 + dx^2y^2$$

defined over the field $\mathbb{F}_{2^{448}-2^{224}-1}$ with curve parameter $d = -39081$. Curve448 satisfies the requirement of SafeCurves [7] and is one of ECC standards for TLS 1.3 [8].

2.2 Target Microcontroller: 32-bit ARM Cortex–M4

The ARM Cortex–M4 microcontroller is a small and energy-efficient ARM processor. The microcontroller supports the ARMv7E-M instruction set, which comprises Thumb-2 instructions and additional DSP extensions. The Cortex-M4 architecture has a 3-stage pipeline with branch speculation. It includes 16 32-bit registers (R0:R15), and supports a mix of 16 and 32-bit operations corresponding to Thumb-2.

Instructions that are relevant for the proposed implementation include 32-bit arithmetic and logical instructions, such as addition (ADD) and addition with

carry (ADC), as well as memory instructions that perform multiple-data loading/storing (LDM/STM).

The microcontroller is equipped with powerful single-cycle multiply and multiply-and-accumulate instructions from DSP extensions, including UMUL, UMLAL, and UMAAL. These instructions compute the product $32 \times 32\text{-bit} \rightarrow 64\text{-bit}$ (UMUL), plus a 64-bit accumulation with a single 64-bit value (UMLAL) or plus a 64-bit accumulation with two 32-bit values (UMAAL). The core instruction set is presented in detail in Table 1.

Table 1. Instruction set summary for ARM Cortex-M4.

Inst.	Operands	Description	Operation
ADD	C, A, B	Addition without Carry	$C \leftarrow A+B$
ADC	C, A, B	Addition with Carry	$C \leftarrow A+B+\text{Carry}$
SUB	C, A, B	Subtraction without Carry	$C \leftarrow A-B$
MOV	C, A	Move 32-bit word between registers	$C \leftarrow A$
UMAAL	D, C, A, B	Multiplication with Accumulation	$\{D C\} \leftarrow A \times B + C + D$
LDM	A!, {B-C}	Loading data from memory to registers	—
STM	A!, {B-C}	Storing data from registers to memory	—

2.3 Previous Implementations

Since Curve448 was recently presented, it has become a new ECC standard as a TLS 1.3. For this reason, only few implementations of Curve448 on low-end microcontrollers are available. In [9], the first Curve448 implementations on both 8-bit AVR ATmega and 16-bit MSP430 microcontrollers were presented. These works achieved 103,228,541 and 73,477,660 clock cycles for scalar multiplication of Curve448 on 8-bit AVR ATmega and 16-bit MSP430 microcontrollers, respectively. To improve the performance, the Karatsuba algorithm is utilized for multi-precision multiplication. On the 32-bit ARM Cortex-M4 microcontroller, several studies have investigated optimized implementations of the well-known Curve25519 [10–13]. Curve25519 provides a 128-bit security level (i.e. short-term security), while Curve448 provides a 224-bit security level (i.e. long-term security). For long-term security, implementation of Curve448 should be considered rather than Curve25519. In this paper, we present an optimized implementation of Curve448 on the 32-bit ARM Cortex-M4 microcontroller for the first time.

3 Optimization Techniques for Curve448 on 32-bit ARM Cortex-M4

ECC implementations consist of finite-field arithmetic and group operation. For finite-field arithmetic, modular addition, subtraction, multiplication, squaring, and inversion operations are required. For group operations, point addition, point doubling, and scalar multiplication operations are required.

3.1 Finite-Field Operations

Finite-Field Addition / Subtraction The 448-bit addition and subtraction operations are performed together with modular reduction for finite-field addition and subtraction. First, addition or subtraction is performed. Then, modular reduction is performed when the addition or subtraction generates a carry bit or borrow bit as follows:

Integer Addition/Subtraction $\rightarrow$ Modular Reduction

According to the school-book approach, modular reduction is performed whenever a carry bit or borrow bit is captured. This approach generates leakage information from branch statements. For this reason, modular reduction is always performed regardless of the carry or borrow bit, which removes the relation between secret information and modular reduction. When the carry bit or borrow bit is set, the mask value is generated from it. For example, when the borrow bit is set, the value is 0xFFFFFFFF, which is used to mask the modulus. When the carry bit is set, the zero value is subtracted by the carry bit, which also generates 0xFFFFFFFF mask. Afterward, the masked modulus is added/subtracted to/from the intermediate results for modular subtraction and modular addition, respectively.

For efficient memory access, the usage of registers is also optimized further because the register access is much faster than the memory access. The 32-bit ARM Cortex-M4 microcontroller provides 14 general purpose registers. These registers cannot maintain all operands and intermediate results throughout the computation to reduce the number of memory accesses. For this reason, only some of the intermediate results are maintained in registers, while the others are stored in memory. For this purpose, 9, 2, and 3 registers are used for intermediate results, temporal storage, and memory pointers, respectively.

Finite-Field Multiplication Multiplication is the most expensive operation of ECC implementation. The multiplication consists of integer multiplication and modular reduction. The proposed implementation performs each operation separately.

Integer Multiplication $\rightarrow$ Modular Reduction

To improve the multiplication performance on the 32-bit ARM Cortex-M4 microcontroller, the operand caching (OC) method is utilized [14]. The OC method caches many operands in registers, which reduces the number of memory accesses efficiently. In a previous work, the OC method with a width of 4 was adopted utilizing general purpose registers [4]. The order of instructions was also optimized to reduce the number of pipeline stalls.

Figure 1 illustrates strategies for implementing 448-bit multiplication on 32-bit ARM Cortex-M4 microcontroller. Let A and B be operands of length 448 bits each. Each operand is written as $A = (A[13], \dots, A[1], A[0])$ and $B = (B[13], \dots, B[1], B[0])$. The result $C = A \cdot B$ is represented as $C = (C[27], \dots,$

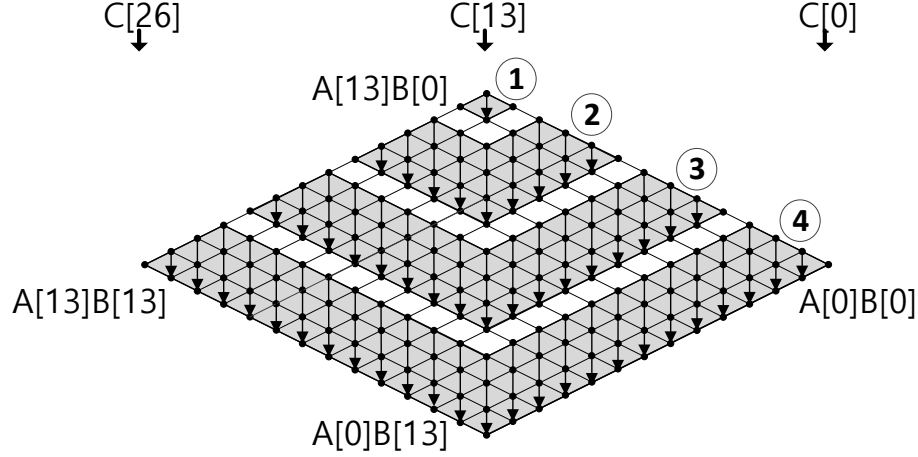


Fig. 1. 448-bit Operand Scanning multiplication at the word-level on ARM Cortex-M4 [4].

$C[1]$, $C[0]$). In the rhombus form, the lowest indices ($i, j = 0$) of the product appear at the rightmost corner, whereas the highest indices ($i, j = 13$) appear at the leftmost corner. A black arrow over a point indicates the processing of a partial product. The lowermost points represent the results $C[i]$ from the rightmost corner ($i = 0$) to the leftmost corner ($i = 27$). Computation is performed from ① to ④. Because the length of the operand caching is set to 4, the process is divided into 4 sections.

Finally, the implementation achieved 566 clock cycles for 448-bit multiplication. Because this approach achieves the best performance, we adopted our implementation. For better performance, the Karatsuba algorithm was also considered but performance improvement was not observed during the experiment due to the high efficiency of UMAAL instructions. For the modular reduction, the fast reduction method introduced in [9] was adopted. Detailed descriptions of Curve448 are given in Algorithm 1. All general purpose registers are utilized to maintain the intermediate results. In Step 1, both operands $A[2]$ and $A[3]$ are added and output the intermediate result ($\varepsilon 0 \| T$). The intermediate result (T) is maintained in registers, and the carry bit ($\varepsilon 0 \| T$) is stored in **STACK**.

In Step 2, both operands $A[0]$ and $\varepsilon 0 \| T$ are added and output intermediate result ($\varepsilon 1 \| C[0]$). The intermediate result ($\varepsilon 1 \| C[0]$) is stored in **STACK**, while the intermediate result (T) is maintained in registers.

In Step 3, the operand ($A[1]$) is loaded and the intermediate result ($\varepsilon 0 \| T$) is added. Then, the operand ($A[3]$) is added to the intermediate result and output the intermediate result ($\varepsilon 2 \| C[1]$).

From Step 4 to Step 7, carry bits are added to the intermediate result. Both intermediate results ($C[0]$ and $C[1]$) are maintained in registers. Two registers are utilized to handle carry bits, while part of registers are stored in **STACK**.

Algorithm 1 Fast reduction Curve448 [9].

Require: 896-bit intermediate result A ($A[3] \sim A[0]$ in 224-bit)

Ensure: 448-bit result C ($C[1] \parallel C[0]$ in 224-bit)

```

1:  $\varepsilon_0 \parallel T \leftarrow A[2] + A[3]$ 

2:  $\varepsilon_1 \parallel C[0] \leftarrow A[0] + \varepsilon_0 \parallel T$ 
3:  $\varepsilon_2 \parallel C[1] \leftarrow A[1] + A[3] + \varepsilon_0 \parallel T$ 

4:  $\varepsilon_3 \parallel C[0] \leftarrow C[0] + \varepsilon_2$ 
5:  $\varepsilon_4 \parallel C[1] \leftarrow C[1] + (\varepsilon_1 + \varepsilon_2 + \varepsilon_3)$ 

6:  $\varepsilon_5 \parallel C[0] \leftarrow C[0] + \varepsilon_4$ 
7:  $C[1] \leftarrow C[1] + (\varepsilon_4 + \varepsilon_5)$ 

8: return  $C$ 
```

Finite-Field Inversion The finite-field inversion can be performed by following Fermat's Theorem. The prime of Curve448 is $p = 2^{448} - 2^{224} - 1$ and the computation of inversion is $a = z^{-1} \equiv z^{2^{448} - 2^{224} - 3} \bmod p$. The inversion operation can be performed with 447 modular squaring and 13 modular multiplication operations. Detailed descriptions are given in Algorithm 2.

3.2 Group Operations

The scalar multiplication of Curve448 requires a number of point addition and point doubling operations. The school-book approach to perform the scalar multiplication executes the addition operation depending on the secret value (i.e. branch statement). In order to ensure the constant execution timing for scalar multiplication, Montgomery ladder algorithm is utilized [15]. The Montgomery ladder algorithm performs point addition and point doubling in a regular pattern. The inner routine of the point addition performs addition of two points, including $P1(x1, y1, z1, e1, h1)$ in extended projective coordinates and $P2(u2, v2, w2)$ in extended affine coordinates. This point addition outputs the point $P3(x3, y3, z3, e3, h3)$ in extended projective coordinates. The detailed procedure of point addition is given in Algorithm 3.

The inner routine of the point doubling performs doubling of one point, including $P1(x1, y1, z1, e1, h1)$ in extended projective coordinates. This point doubling outputs the point $P3(x3, y3, z3, e3, h3)$ in extended projective coordinates. The detailed procedure of point doubling is given in Algorithm 4.

3.3 Side-Channel Attack Protection

The cryptography implementation may include the conditional branch depending on the secret. The proposed implementation of finite-field operation is per-

Algorithm 2 Fermat-based inversion for Curve448 ($p = 2^{448} - 2^{224} - 1$).

Require: Integer z satisfying $1 \leq z \leq p - 1$.**Ensure:** Inverse $t_7 = z^{p-2} \bmod p = z^{-1} \bmod p$.

```

1:  $z_3 \leftarrow z^{2^1} \cdot z$  { cost: 1S+1M}
2:  $t_0 \leftarrow z_3^{2^2} \cdot z_3$  { cost: 2S+1M}
3:  $t_1 \leftarrow t_0^{2^1} \cdot z$  { cost: 1S+1M}
4:  $t_2 \leftarrow t_1^{2^4} \cdot t_0$  { cost: 4S+1M}
5:  $t_3 \leftarrow t_2^{2^9} \cdot t_2$  { cost: 9S+1M}
6:  $t_4 \leftarrow (t_3^{2^{18}} \cdot t_3)^2 \cdot z$  { cost: 19S+2M}
7:  $t_5 \leftarrow (t_4^{2^{37}} \cdot t_4)^{2^{37}} \cdot t_4$  { cost: 74S+2M}
8:  $t_6 \leftarrow t_5^{2^{111}} \cdot t_5$  { cost: 111S+1M}
9:  $t_7 \leftarrow (t_6^{2^1} \cdot z^{2^{223}} \cdot t_6)^{2^2} \cdot z$  { cost: 226S+3M}

10: return  $t_7$ 

```

Table 2. Evaluation of finite-field operation and group operation on the 32-bit ARM Cortex-M4 microcontrollers in speed (in clock cycles).

Frequency	finite-field Operation				Group Operation		
	Addition	Subtraction	Multiplication	Inversion	Addition	Doubling	Scalar Multiplication
24MHz	164	161	821	363,485	6,566	6,567	6,218,135
168MHz	181	172	838	363,626	6,686	6,674	6,285,904

formed with constant timing by replacing the conditional branch with masked operation. The mask generation is as follows:

$$mask \leftarrow 0 - (carry \text{ or } borrow)$$

Furthermore, the legacy ARM Cortex-M3 has early termination issues depending on the input values [23]. Because the ARM Cortex-M4 is the successor of the ARM Cortex-M3, all arithmetic and logical operations are performed in one clock cycle. This satisfies one requirement for constant timing.

For the case of group operation, the Montgomery ladder algorithm always performs point doubling and point addition in regular fashion [24]. When the target processor equips the cache, the implementation must prevent a cache attack. The cache is activated when memory accesses happen frequently depending on a certain regular pattern of input (i.e. pre-computed result). The proposed implementation does not utilize the pre-computed result to avoid a cache attack.

With the above approaches, the implementation achieved constant timing and this is a basic requirement for cryptographic implementation (i.e. timing attack resistant). The checklist for constant timing is presented in Table 4. The proposed implementation satisfies all requirements for constant timing.

Algorithm 3 Point Addition for Curve448.

Require: Point $P1 = (x1, y1, z1, e1, h1)$ in extended projective coordinates, Point $P2 = (u2, v2, w2)$ in extended affine coordinates

Ensure: $P3 = (x3, y3, z3, e3, h3)$ in extended projective coordinates

```

1:  $t1 \leftarrow e1 \cdot h1$ 
2:  $e3 \leftarrow y1 - x1$ 
3:  $h3 \leftarrow y1 + x1$ 
4:  $x3 \leftarrow e3 \cdot v2$ 
5:  $y3 \leftarrow h3 \cdot u2$ 
6:  $e3 \leftarrow y3 - x3$ 
7:  $h3 \leftarrow y3 + x3$ 
8:  $x3 \leftarrow t1 \cdot w2$ 
9:  $t1 \leftarrow z1 - x3$ 
10:  $x3 \leftarrow z1 + x3$ 
11:  $z3 \leftarrow t1 \cdot x3$ 
12:  $y3 \leftarrow x3 \cdot h3$ 
13:  $x3 \leftarrow e3 \cdot t1$ 
14: return  $P3(x3, y3, z3, e3, h3)$ 

```

$\{ A = (y1 - x1) \cdot (y2 - x2) \}$
 $\{ B = (y1 + x1) \cdot (y2 + x2) \}$
 $\{ E = B - A \}$
 $\{ H = B + A \}$
 $\{ C = t1 \cdot w2 \}$
 $\{ F = z1 - C \}$
 $\{ G = z1 + C \}$
 $\{ Z3 = F \cdot G \}$
 $\{ Y3 = G \cdot H \}$
 $\{ X3 = E \cdot F \}$

4 Evaluation

In this section, we first evaluate the proposed implementations of finite-field operation and group operation for 448-bit wise on the 32-bit ARM Cortex-M4 microcontroller. Then, a comparison of scalar multiplication on low-end processors will be presented.

A benchmark result was obtained on an STM32F4 Discovery board equipped with 32-bit ARM Cortex-M4 microcontrollers. The execution timing in clock cycles was obtained at two frequencies (24MHz and 168MHz). The high frequency (i.e. 168MHz) was for the real-world application, and it showed the highest performance. The low frequency (i.e. 24MHz) is to avoid wait cycles due to the speed of the memory controller, which ensures the correct clock cycles. All implementations of arithmetic were implemented in the ARM assembly, and the libraries were compiled with GCC with optimization flags set to `-O3`.

The results of finite-field operation and group operation on the 32-bit ARM Cortex-M4 microcontroller is presented in Table 2. Finite-field addition, subtraction, multiplication, and inversion operations require 164/181, 161/172, 821/838, and 363,485/363,626 clock cycles for 24MHz/168MHz, respectively. Clock cycles at 24MHz show better performance than the 168MHz case because the frequency does not have a wait delay. Group addition, doubling, and scalar multiplication operations require 6,566/6,686, 6,567/6,674, and 6,218,135/6,285,904 clock cycles for 24MHz/168MHz, respectively.

In Table 3, a comparison of scalar multiplication on 8-bit AVR, 16-bit MSP, and 32-bit ARM processors is presented. For other low-end microcontrollers, NIST P-256 shows the worst performance among 128-bit security ECCs. The

Algorithm 4 Point Doubling for Curve448.**Require:** Point $P1 = (x1, y1, z1, e1, h1)$ in extended projective coordinates**Ensure:** $P3 = (x3, y3, z3, e3, h3)$ in extended projective coordinates

1: $e3 \leftarrow x1 \cdot x1$	$\{ A = x1 \cdot x1 \}$
2: $h3 \leftarrow y1 \cdot y1$	$\{ B = y1 \cdot y1 \}$
3: $t1 \leftarrow e3 - h3$	$\{ G = A - B \}$
4: $h3 \leftarrow e3 + h3$	$\{ H = A + B \}$
5: $x3 \leftarrow x1 + y1$	
6: $e3 \leftarrow x3 \cdot x3$	
7: $e3 \leftarrow h3 - e3$	$\{ E = H - (x1 + y1) \cdot (x1 + y1) \}$
8: $y3 \leftarrow z1 \cdot z1$	
9: $y3 \leftarrow 2 \cdot y3$	$\{ C := 2 \cdot z1 \cdot z1 \}$
10: $y3 \leftarrow t1 + y3$	$\{ F := G + C \}$
11: $x3 \leftarrow e3 \cdot y3$	$\{ X3 := E \cdot F \}$
12: $z3 \leftarrow y3 \cdot t1$	$\{ Z3 := F \cdot G \}$
13: $y3 \leftarrow t1 \cdot h3$	$\{ Y3 := G \cdot H \}$

14: **return** $P3(x3, y3, z3, e3, h3)$

fastest performance is achieved in the implementation of Four $\mathbb{Q}$. Implementations of Curve25519 show middle performance. The 224-bit security ECC (i.e. Curve448) on 8-bit AVR ATmega and 16-bit MSP430 requires 103M and 73M clock cycles, respectively. The performance of Curve448 is relatively slower than that of 128-bit security ECC implementations because of its parameters. On the 32-bit ARM Cortex-M4 microcontroller, the fastest implementation of Curve25519 requires 847,048 clock cycles [13], while the Four $\mathbb{Q}$ requires 542,900 clock cycles [20]. The proposed implementation of Curve448 requires 6,218,135 clock cycles. Compared with other ECC implementations, the implementation of Curve448 is 86% and 91% slower than Curve25519 and Four $\mathbb{Q}$ because these curves are defined over small finite-fields, which ensure compact finite-field implementations on the target processor. ROM and RAM sizes are 3,828 bytes and 2,128 bytes, respectively.

4.1 Trade-off between Performance and Security

Performance and security have trade-off relations between them. In the implementation, we focused on security first. The recommended security level by 2030 is 128-bit (i.e. Curve25519 and Four $\mathbb{Q}$) [25, 26]. Even though the performance of 128-bit security ECCs (i.e. Curve25519 and Four $\mathbb{Q}$) is better than that of 224-bit security ECC (i.e. Curve448), security-sensitive services should ensure high security levels. This is even more secure against quantum attacks. The quantum resources for the 224-bit security ECC are significantly more than those for 128-bit security ECCs [27].

Table 3. Comparison of scalar multiplication on 8-bit AVR ATmega, 16-bit MSP430(X), and 32-bit ARM Cortex-M4 processors in speed (in clock cycles).

Target	Implementation	128-bit security			224-bit security Curve448
		NIST P-256	Curve25519	FourQ	
8-bit AVR ATmega	Wenger et al. [16]	34,930 000	–	–	–
	Hutter and Schwabe [17]	–	22,791,580	–	–
	Nascimento et al. [18]	–	20,153,658	–	–
	Düll et al. [19]	–	13,900,397	–	–
	Liu et al. [20]	–	–	7,296,000	–
	Seo [9]	–	–	–	103,228,541
16-bit MSP430	Wenger et al. [16]	22,170 000	–	–	–
	Gouvêa and López [21]	20,476,234	–	–	–
	Seo [9]	–	–	–	73,477,660
16-bit MSP430X	Hinterwälder et al. [22]	–	6,513,011	–	–
	Düll et al. [19]	–	5,301,792	–	–
	Liu et al. [20]	–	–	4,826,100	–
32-bit ARM Cortex-M4	Groot [10]	–	1,816,351	–	–
	Santis and Sigl [11]	–	1,563,852	–	–
	Fujii and Aranha [12]	–	907,240	–	–
	Haase and Labrique [13]	–	847,048	–	–
	Liu et al. [20]	–	–	542,900	–
	This work	–	–	–	6,218,135

Table 4. Checklist for ECC implementations in constant timing.

Masked implementation	Early termination prevention	Montgomery ladder	w/o look-up table
✓	✓	✓	✓

5 Hybrid Post-Quantum TLS

During the transition from pre-quantum cryptography to post-quantum cryptography, both algorithms should be supported in real-world applications. Recently, AWS cryptography proposed supersingular isogeny key encapsulation (SIKE) based hybrid post-quantum transport layer security (TLS) algorithms¹. Because SIKE is an alternative candidates, this algorithm should be counted for PQC². Classical TLS 1.2 and hybrid post-quantum TLS 1.2 are compared in detail in Table 5. The protocol performs two independent key exchanges (one classical and one post-quantum). Then, both keys are combined into a single TLS master secret. The hybrid post-quantum TLS allows network connections to be secure when one of the key exchanges (i.e. classical or post-quantum) for TLS is compromised by hackers. One of the potential scenarios is quantum computers. If a large-scale quantum computer is developed in the near future, the current discrete logarithm problem (DLP) and integer factorization (IF)-based public key cryptography will be vulnerable. Under this difficult condition, the hybrid post-quantum TLS still keeps the connection in secret. Similarly, PQC is not completely proven to be secure against the quantum computer and quantum algorithm. When PQC has a backdoor, the legacy PKC still ensures security.

¹ <https://aws.amazon.com/ko/blogs/security/round-2-hybrid-post-quantum-tls-benchmarks/>

² <https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement>

In Table 6, the performance of isogeny based post-quantum cryptography (i.e. SIKE) is presented. The execution timing for SIKEp434, SIKEp503, SIKEp610, and SIKEp751 require 184, 257, 493, and 770 million clock cycles, respectively. Implementations on the 168MHz Cortex-M4 take 1.09, 1.53, 2.94, and 4.58 s for SIKEp434, SIKEp503, SIKEp610, and SIKEp751, respectively. The performance is not as fast as pre-quantum PKC but it is still practically fast enough for real-world applications, considering that PKC is not frequently performed. When ECC and SIKE cryptography systems are adopted for hybrid post-quantum TLS, the multiplication part can be shared. This optimizes the code size. It is also possible to adopt other PQC candidates for protocols. This is our future work.

Table 5. Comparison result between classical TLS 1.2 and hybrid post-quantum TLS 1.2 [28].

Classical TLS 1.2	Hybrid Post-Quantum TLS 1.2
premaster_secret = ECDHE.KEY seed = “master secret” ClientHello.random ServerHello.random master_secret=HMAC(premaster_secret,seed)	premaster_secret = ECDHE.KEY PQ_KEY seed = “hybrid master secret” ClientHello.random ServerHello.random master_secret=HMAC(premaster_secret,seed)

6 Conclusion

In this paper, we presented the first optimized implementation of Curve448 on the 32-bit ARM Cortex-M4 microcontroller. State-of-art implementation techniques are used to achieve the optimal performance. The proposed implementation achieved 6,218,135 clock cycles. This is practically fast enough considering that the target microcontroller supports a 168 MHz operating frequency. Furthermore, the implementation is secure against timing attacks by avoiding conditional branch and cache access.

Our future work is practical implementation of a hybrid post-quantum TLS protocol for pre-quantum and post-quantum cryptography algorithms. We will investigate the secure and efficient implementation of both protocols to achieve the highest performance.

7 Acknowledgement

This work of Hwajeong Seo was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (<Q|Crypton>, No.2019-0-00033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity). This work of Reza Azarderakhsh was supported by ARO grant W911NF2010328.

Table 6. SIKE implementations on the ARM Cortex-M4 microcontrollers.

Implementation	Timings [$cc \times 10^6$]				Timings [second]			
	KeyGen	Encaps	Decaps	Total	KeyGen	Encaps	Decaps	Total
SIKEp434 (AES-128)								
Seo et al. [4]	74	122	130	252	0.44	0.73	0.77	1.50
Seo et al. [29]	54	89	95	184	0.32	0.53	0.56	1.09
SIKEp503 (SHA-256)								
Seo et al. [4]	104	172	183	355	0.62	1.02	1.09	2.11
Seo et al. [29]	76	125	133	257	0.45	0.74	0.79	1.53
SIKEp610 (AES-192)								
Seo et al. [29]	134	246	248	493	0.80	1.46	1.47	2.94
SIKEp751 (AES-256)								
Seo et al. [4]	282	455	491	946	1.68	2.71	2.92	5.63
Seo et al. [29]	229	371	399	770	1.36	2.21	2.37	4.58

References

1. M. Hamburg, “Ed448-Goldilocks, a new elliptic curve,” *IACR Cryptology ePrint Archive*, vol. 2015, p. 625, 2015.
2. M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, “pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4,” 2019.
3. M. J. Kannwischer, J. Rijneveld, and P. Schwabe, “Faster multiplication in $\mathbb{Z}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates,” in *International Conference on Applied Cryptography and Network Security*, pp. 281–301, Springer, 2019.
4. H. Seo, A. Jalali, and R. Azarderakhsh, “SIKE round 2 speed record on ARM Cortex-M4,” in *International Conference on Cryptology and Network Security*, pp. 39–60, Springer, 2019.
5. L. Botros, M. J. Kannwischer, and P. Schwabe, “Memory-efficient high-speed implementation of Kyber on Cortex-M4,” in *International Conference on Cryptology in Africa*, pp. 209–228, Springer, 2019.
6. H. Edwards, “A normal form for elliptic curves,” *Bulletin of the American Mathematical Society*, vol. 44, no. 3, pp. 393–422, 2007.
7. D. J. Bernstein, T. Lange, et al., “SafeCurves: choosing safe curves for elliptic-curve cryptography,” URL: <http://safecurves.cr.yp.to>, 2013.
8. E. Rescorla et al., “The transport layer security (TLS) protocol version 1.3,” URL: <https://tools.ietf.org/html/draft-ietf-tls-tls13-21>, 2017.
9. H. Seo, “Compact implementations of Curve Ed448 on low-end IoT platforms,” *ETRI Journal*, vol. 41, no. 6, pp. 863–872, 2019.
10. W. de Groot, *A Performance Study of X25519 on Cortex-M3 and M4*. PhD thesis, Ph. D. thesis, Eindhoven University of Technology, 2015.
11. F. De Santis and G. Sigl, “Towards side-channel protected X25519 on ARM Cortex-M4 processors,” *Proceedings of Software performance enhancement for encryption and decryption, and benchmarking, Utrecht, The Netherlands*, pp. 19–21, 2016.
12. H. Fujii and D. F. Aranha, “Curve25519 for the Cortex-M4 and beyond,” in *International Conference on Cryptology and Information Security in Latin America*, pp. 109–127, Springer, 2017.

13. B. Haase and B. Labrique, “AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–48, 2019.
14. M. Hutter and E. Wenger, “Fast multi-precision multiplication for public-key cryptography on embedded microprocessors,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 459–474, Springer, 2011.
15. P. L. Montgomery, “Speeding the pollard and elliptic curve methods of factorization,” *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.
16. E. Wenger, T. Unterluggauer, and M. Werner, “8/16/32 shades of elliptic curve cryptography on embedded processors,” in *International Conference on Cryptology in India*, pp. 244–261, Springer, 2013.
17. M. Hutter and P. Schwabe, “NaCl on 8-bit AVR microcontrollers,” in *International Conference on Cryptology in Africa*, pp. 156–172, Springer, 2013.
18. E. Nascimento, J. López, and R. Dahab, “Efficient and secure elliptic curve cryptography for 8-bit AVR microcontrollers,” in *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 289–309, Springer, 2015.
19. M. Düll, B. Haase, G. Hinterwälder, M. Hutter, C. Paar, A. H. Sánchez, and P. Schwabe, “High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers,” *Designs, Codes and Cryptography*, vol. 77, no. 2-3, pp. 493–514, 2015.
20. Z. Liu, P. Longa, G. Pereira, O. Reparaz, and H. Seo, “FourQ on embedded devices with strong countermeasures against side-channel attacks,” *IEEE Transactions on Dependable and Secure Computing*, 2018.
21. C. P. L. Gouvêa and J. López, “Software implementation of pairing-based cryptography on sensor networks using the MSP430 microcontroller,” in *International Conference on Cryptology in India*, pp. 248–262, Springer, 2009.
22. G. Hinterwälder, A. Moradi, M. Hutter, P. Schwabe, and C. Paar, “Full-size high-security ECC implementation on MSP430 microcontrollers,” in *International conference on cryptology and information security in Latin America*, pp. 31–47, Springer, 2014.
23. C. Franck, J. Großschädl, Y. Le Corre, and C. L. Tago, “Energy-scalable montgomery-curve ECDH key exchange for ARM cortex-M3 microcontrollers,” in *2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 231–236, IEEE, 2018.
24. M. Joye and S.-M. Yen, “The Montgomery powering ladder,” in *International workshop on cryptographic hardware and embedded systems*, pp. 291–302, Springer, 2002.
25. E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, *et al.*, *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2006.
26. H. Orman and P. Hoffman, “Determining strengths for public keys used for exchanging symmetric keys,” tech. rep., BCP 86, RFC 3766, April, 2004.
27. M. Roetteler, M. Naehrig, K. M. Svore, and K. Lauter, “Quantum resource estimates for computing elliptic curve discrete logarithms,” in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 241–270, Springer, 2017.
28. M. Campagna and E. Crockett, “Hybrid post-quantum key encapsulation methods (PQ KEM) for transport layer security 1.2 (TLS),” *Internet Engineering Task Force, Internet-Draft draft-campagna-tls-bike-sike-hybrid-01*, 2019.
29. H. Seo, M. Anastasova, A. Jalali, and R. Azarderakhsh, “Supersingular isogeny key encapsulation (SIKE) round 2 on ARM Cortex-M4,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 410, 2020.

Optimized Implementation of SIKE Round 2 on 64-bit ARM Cortex-A Processors

Hwajeong Seo¹, Pakize Sanal, Amir Jalali², and Reza Azarderakhsh³, *Member, IEEE*

Abstract— In this work, we present the first highly-optimized implementation of Supersingular Isogeny Key Encapsulation (SIKE) submitted to NIST's second round of post quantum standardization process, on 64-bit ARMv8 processors. To the best of our knowledge, this work is the first optimized implementation of SIKE round 2 on 64-bit ARM over SIKEp434 and SIKEp610. The proposed library is explicitly optimized for these two security levels and provides constant-time implementation of the SIKE mechanism on ARMv8-powered embedded devices. We adapt different optimization techniques to reduce the total number of underlying arithmetic operations on the field level. In particular, benchmark results on embedded processors equipped with ARM Cortex-A55@1.766GHz and ARM Cortex-A75@2.803GHz show that the entire SIKE round 2 Key Encapsulation Mechanism (KEM) takes only 98.6 ms and 85.3ms at NIST's security level 1, respectively. We also evaluated the compressed version of NIST's security level 1, which requires 134.7 ms and 113.7 ms for Cortex-A55 and Cortex-A75, respectively. Considering SIKE's extremely small key size in comparison to other post-quantum cryptography candidates, our result implies that SIKE is one of the promising candidates for key encapsulation mechanism on embedded devices in the quantum era.

Index Terms—Post-quantum cryptography, isogeny-based cryptography, ARM processors, assembly, key encapsulation mechanism.

I. INTRODUCTION

INITIATED by the National Institute of Standards and Technology (NIST), Post-Quantum Cryptography (PQC) has been elevated to a standardization process to solicit, evaluate, and standardize one or more quantum-resistant public-key

cryptographic algorithms [25]. To prepare for security concerns caused by quantum computers, in 2016, NIST called for the cryptographic algorithms which were assumed to be resistance against high-scale quantum computers. These proposals provided Key Encapsulation Mechanism (KEM) or digital signature algorithms from different arithmetic structures, resulting in different characteristics and parameters. Recently, NIST announced approved candidates for round 2 which are the most promising candidates in terms of security, performance, and compatibility with current cryptography technology. For the key encapsulation mechanism, only 17 candidates made it through to the second round for being evaluated and analyzed from different perspectives.

Different PQC candidates are constructed on hard mathematical problems which are assumed to be impossible to solve even for large-scale quantum computers. These problems can be categorized into five main categories: code-based cryptography, lattice-based cryptography, hash-based cryptography, multivariate cryptography, and supersingular isogeny-based cryptography, see, for instance [9].

Supersingular Isogeny Key Encapsulation (SIKE) mechanism is one of the PQC candidates which is constructed on the hardness of solving isogeny maps between supersingular elliptic curves. In fact, SIKE is the only candidate that offers the quantum-resistance cryptographic construction over elliptic curves, resulting in well-known structures in implementation perspective. The proposed key encapsulation mechanism is derived from the original Jao-De Feo's Diffie-Hellman key-exchange and public-key encryption algorithms [20]. However, constructing cryptographic structures from hardness of supersingular isogeny graphs was introduced by Charels-Lauter-Goren [8].

The first round SIKE submission offered three different security levels known as SIKEp503, SIKEp751, and SIKEp964. According to the best known quantum attacks on solving supersingular isogeny problem by that time, the proposed security levels met NIST's level 1, 3, and 5 requirements, respectively.

However, recent studies on the cost of solving isogeny problem on quantum computers by Adj et al. [2] revealed that the security assumptions for SIKE was too conservative. In fact, a set of realistic models of quantum computation on solving Computational Supersingular Isogeny (CSSI) problem in [2] suggests that the Oorschot-Wiener golden collision search is the most powerful attack on the CSSI problem, resulting in significant improvement on the SIKE's classical and quantum security levels.

Manuscript received November 24, 2019; revised February 14, 2020; accepted March 2, 2020. Date of publication March 16, 2020; date of current version July 31, 2020. This work of Hwajeong Seo was supported in part by the Institute for Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korean Government (MSIT) (Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity) under Grant 2019-0-00033, and in part by the Institute for Information & Communications Technology Promotion (IITP) Grant funded by the Korean Government (MSIT) (Research on Blockchain Security Technology for IoT Services) under Grant 2018-0-00264. This work of Reza Azarderakhsh was supported in part by NSF under Grant CNS-1801341, Grant NIST-60NANB17D184, Grant NIST-60NANB16D246, and Grant ARO W911NF-17-1-0311. This article was recommended by Associate Editor M. Martina. (Corresponding author: Reza Azarderakhsh.)

Hwajeong Seo is with the College of IT Engineering, Hansung University, Seoul 13557, South Korea (e-mail: hwajeong84@gmail.com).

Pakize Sanal and Reza Azarderakhsh are with the Department of Computer, Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: psanal2018@fau.edu; razarderakhsh@fau.edu).

Amir Jalali is with the Information Security Group, LinkedIn Corporation, Sunnyvale, CA 94085 USA (e-mail: ajalali@linkedin.com).

Color versions of one or more of the figures in this article are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCSI.2020.2979410

1549-8328 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.

See <https://www.ieee.org/publications/rights/index.html> for more information.

Accordingly, the second round SIKE [4] offers a new set of security levels which are more realistic and provide significant improvement on the key encapsulation performance. In particular, decreasing the bit-length of SIKE's primes translates to notable performance improvement, making this scheme suitable for many potential applications on low-end embedded devices.

In this work, we provide a full report on the highly-optimized implementation of SIKE on 64-bit ARM embedded processors over all proposed security levels. In particular, the reference optimized implementation of SIKE [4] on 64-bit ARM embedded processor only targets two security levels, i.e., SIKEp503 and SIKEp751. Therefore, in this work, we address this shortcoming by providing the KEM full benchmarks on different security levels which provide a reference for the performance analysis of this scheme for the second round.

Our proposed library takes advantage of state-of-the-art engineering techniques as well as low level assembly optimizations. We studied different approaches for finite field arithmetic implementation over SIKE's new primes. Our benchmark results offer significant improvement in performance compared to portable implementation, suggesting the possible integration of this scheme on mobile devices in the future.

II. BACKGROUND

In this section, we briefly review the SIDH protocol and the required steps for Alice and Bob to generate a shared secret. Furthermore, we describe the SIKE, a post-quantum key encapsulation mechanism from isogenies of supersingular elliptic curves which was submitted to NIST's PQC standardization competition. We refer the readers to [5], [20] for further details.

Let E_1 and E_2 be elliptic curves over a finite field $\mathbb{F}_q$. An isogeny $\phi : E_1 \rightarrow E_2$ is a non-constant rational map defined over $\mathbb{F}_q$ which is also a group homomorphism from $E_1(\mathbb{F}_q)$ to $E_2(\mathbb{F}_q)$. If such a map exists we say E_1 is isogenous to E_2 , and two curves E_1 and E_2 over $\mathbb{F}_q$ are isogenous if and only if $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$.

An isogeny ϕ can be expressed in terms of two rational maps f and g over $\mathbb{F}_q$ such that $\phi((x, y)) = (f(x), y \cdot g(x))$. We can write $f(x) = p(x)/q(x)$ with polynomials $p(x)$ and $q(x)$ over $\mathbb{F}_q$ that do not have a common factor, and similarly for $g(x)$. The degree $\deg(\phi)$ of the isogeny is defined as $\max\{\deg(p(x)), \deg(q(x))\}$.

Given an isogeny $\phi : E_1 \rightarrow E_2$ we define the kernel of ϕ as follows:

$$\ker(\phi) = \{P \in E_1 : \phi(P) = \mathcal{O}\}.$$

For any finite subgroup H of $E(\mathbb{F}_q)$, there is a unique isogeny $\phi : E \rightarrow E'$ such that $\ker(\phi) = H$ and $\deg(\phi) = |H|$, where $|H|$ denotes the cardinality of H . In this case, we denote by E/H the curve E' . Given a subgroup $H \subseteq E(\mathbb{F}_q)$, Velu formula can be used to find the isogeny ϕ and isogenous curve E/H . An example of isogeny map is given in Figure 1.

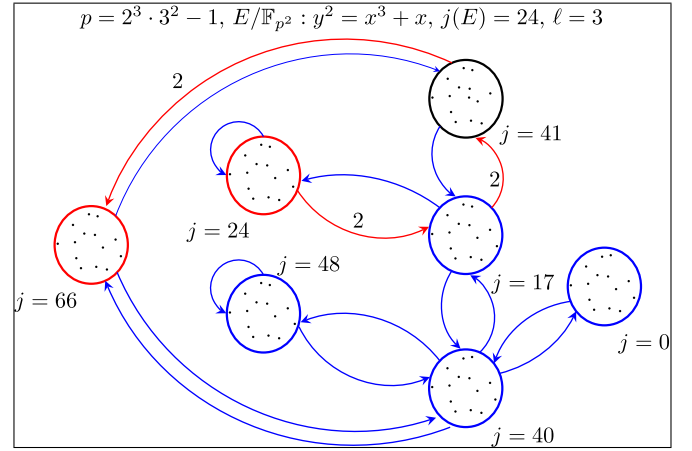


Fig. 1. An 9-degree isogeny map from elliptic curve with $j = 24$ to elliptic curve with $j = 17$ (each circle represents the isomorphism class, the numbers inside of the each circle are the j -invariants of that isomorphism class, the numbers next to the arrows denote the degree of the isogeny).

A. SIDH Key Exchange

In 2011, Jao and De Feo [20] proposed the SIDH, a quantum resistant key exchange protocol from isogenies of supersingular elliptic curves. Similar to classical Diffie-Hellman key exchange, SIDH protocol is constructed over some public parameters which are agreed upon by communication parties prior to key exchange.

1) *Public Parameters*: Fix a prime p of the form $p = \ell_A^{e_A} \cdot \ell_B^{e_B} \cdot f \pm 1$ where ℓ_A and ℓ_B are small primes, e_A and e_B are positive integers, and f is a very small cofactor. We define a based supersingular elliptic curve E over $\mathbb{F}_{p^2}$ with cardinality $\#E = (\ell_A^{e_A} \cdot \ell_B^{e_B} \cdot f \mp 1)^2$, and base points $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ from the torsion subgroups $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ respectively, such that $\langle P_A, Q_A \rangle = E[\ell_A^{e_A}]$ and $\langle P_B, Q_B \rangle = E[\ell_B^{e_B}]$.

2) *Key Exchange Protocol*: Alice randomly chooses two integers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$, not both divisible by ℓ_A as her secret key and computes an isogeny $\phi_A : E \rightarrow E_A$ using kernel $R_A := \langle [m_A]P_A + [n_A]Q_A \rangle$. Alice also computes the image points $\{\phi_A(P_B), \phi_A(Q_B)\} \subset E_A$ by applying her secret isogeny ϕ_A to the public basis P_B and Q_B . She sends $\phi_A(P_B), \phi_A(Q_B)$ and E_A to Bob as her public key. Bob also selects random elements $m_B, n_B \in \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$, not both divisible by ℓ_B and computes a secret isogeny $\phi_B : E \rightarrow E_B$ from kernel $R_B := \langle [m_B]P_B + [n_B]Q_B \rangle$, along with image points $\{\phi_B(P_A), \phi_B(Q_A)\} \subset E_B$. He sends his public key, i.e., $\phi_B(P_A), \phi_B(Q_A)$ and E_B to Alice.

In the second round of key exchange, Alice uses Bob's public key $(\phi_B(P_A), \phi_B(Q_A), E_B)$ and computes an isogeny $\phi'_A : E_B \rightarrow E_{AB}$ from kernel equal to $\langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A) \rangle$; Similarly, Bob computes an isogeny $\phi'_B : E_A \rightarrow E_{BA}$ having kernel $\langle [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B) \rangle$ using Alice's public key. Since the common j -invariant of E_{AB} and E_{BA} are equal, they use this value to form a secret shared key. The entire SIDH key exchange protocol is illustrated in Figure 2.

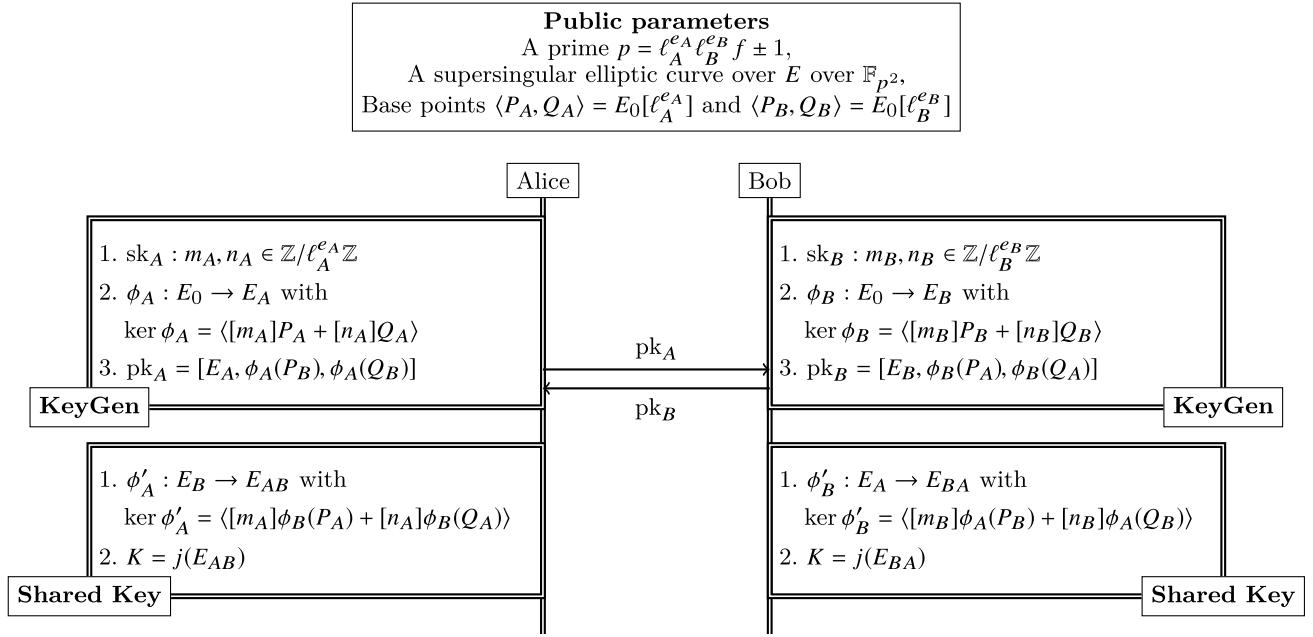


Fig. 2. SIDH key exchange protocol.

B. SIKE Mechanism

SIKE mechanism is constructed by applying a transformation of Hofheinz, Hövelmanns, and Kiltz [16] to the supersingular isogeny Public Key Encryption (PKE) scheme described in [20]. It is an actively secure key encapsulation mechanism (IND-CCA KEM) which addresses the static key vulnerability of SIDH due to active attacks in [14].

1) *Public Parameters:* Similar to SIDH, SIKE can be defined over a prime of the form $p = \ell_A^{e_A} \cdot \ell_B^{e_B} \cdot f \pm 1$. However, for efficiency reasons, $\ell_A = 2$, $\ell_B = 3$, and $f = 1$ are fixed, thus the SIKE prime has the form of $p = 2^{e_A} \cdot 3^{e_B} - 1$. The starting supersingular elliptic curve $E_0/\mathbb{F}_{p^2} : y^2 = x^3 + x$ with cardinality equal to $(2^{e_A} \cdot 3^{e_B})^2$, along with base points $\langle P_A, Q_A \rangle = E_0[2^{e_A}]$ and $\langle P_B, Q_B \rangle = E_0[3^{e_B}]$ are defined as public parameters.

2) *Key Encapsulation Mechanism:* The key encapsulation mechanism can be divided into three main operations: Alice's key generation, Bob's key encapsulation, and Alice's key decapsulation. We describe each operation in the following. Figure 3 presents the entire key encapsulation mechanism in a nutshell.

a) *Key generation:* Alice randomly chooses an integer $sk_A \in \mathbb{Z}/2^{e_A}\mathbb{Z}$ and by applying an isogeny $\phi_A : E_0 \rightarrow E_A$ with kernel $R_A := \langle P_A + [sk_A]Q_A \rangle$ to the base points $\{P_B, Q_B\}$, computes her public key $pk_A = [E_A, \phi_A(P_B), \phi_A(Q_B)]$. Moreover, she generates a t -bit¹ random sequence $s \in_R \{0, 1\}^t$.

b) *Encapsulation:* Bob generates an t -bit random message $m \in_R \{0, 1\}^t$, concatenates it with Alice's public key pk_A and computes an $(e_B \log_2 3)$ -bit hash value r using cSHAKE256 hash function H_1 , taking $m \parallel pk_A$ as the input. Using r , he applies a secret isogeny $\phi_B : E_0 \rightarrow E_B$

to the base points $\{P_A, Q_A\}$ and forms his public key $pk_B(r) = [E_B, \phi_B(P_A), \phi_B(Q_A)]$. Bob also computes the common j -invariant of curve E_{BA} by applying another isogeny $\phi'_B : E_A \rightarrow E_{BA}$ using Alice's public key. Bob forms a ciphertext $c = (c_0, c_1)$, such that:

$$c = (c_0, c_1) = (pk_B(r), H_2(j(E_{BA})) \oplus m),$$

where H_2 is a cSHAKE256 hash with a custom length output and a defined initialization parameter. Finally, Bob computes the shared secret as $K = H_3(m \parallel c)$ and sends c to Alice.

c) *Decapsulation:* Upon receipt of c , Alice computes the common j -invariant of E_{AB} by applying her secret isogeny to E_B . She computes $m' = c_1 \oplus H_2(j(E_{AB}))$ and $r' = H_1(m \parallel pk_A)$. Finally, she validates Bob's public key by computing $pk_B(r')$ and comparing it with c_0 . She generates the same shared secret $K = H_3(m' \parallel c)$ if the public key is valid, otherwise she outputs a random value $K = H_3(s \parallel c)$ to be resistant against active attacks.

C. Key Compression

Compared to other PQC candidates, SIKE provides the smallest public key size which makes this scheme a suitable candidate for the applications with limited bandwidth/memory. Furthermore, the standard representation of SIKE's public keys can be further compressed using the key-compression technique proposed in the SIKE round 2 proposal with a slight but not negligible overhead in overall performance. First key compression is proposed by Azarderakhsh et al. in [6] and then it is improved in [11] for SIDH. Recently a compressed key for SIKE is appeared in SIKE round 2 in [4]. The key compression efficiency on SIKE has been improved by dual isogenies method. In particular, recent optimization in pairing computations and basis generation, in addition to fast and compact x -only formulas for dual isogenies of

¹The value of t is defined by the implementation parameters.

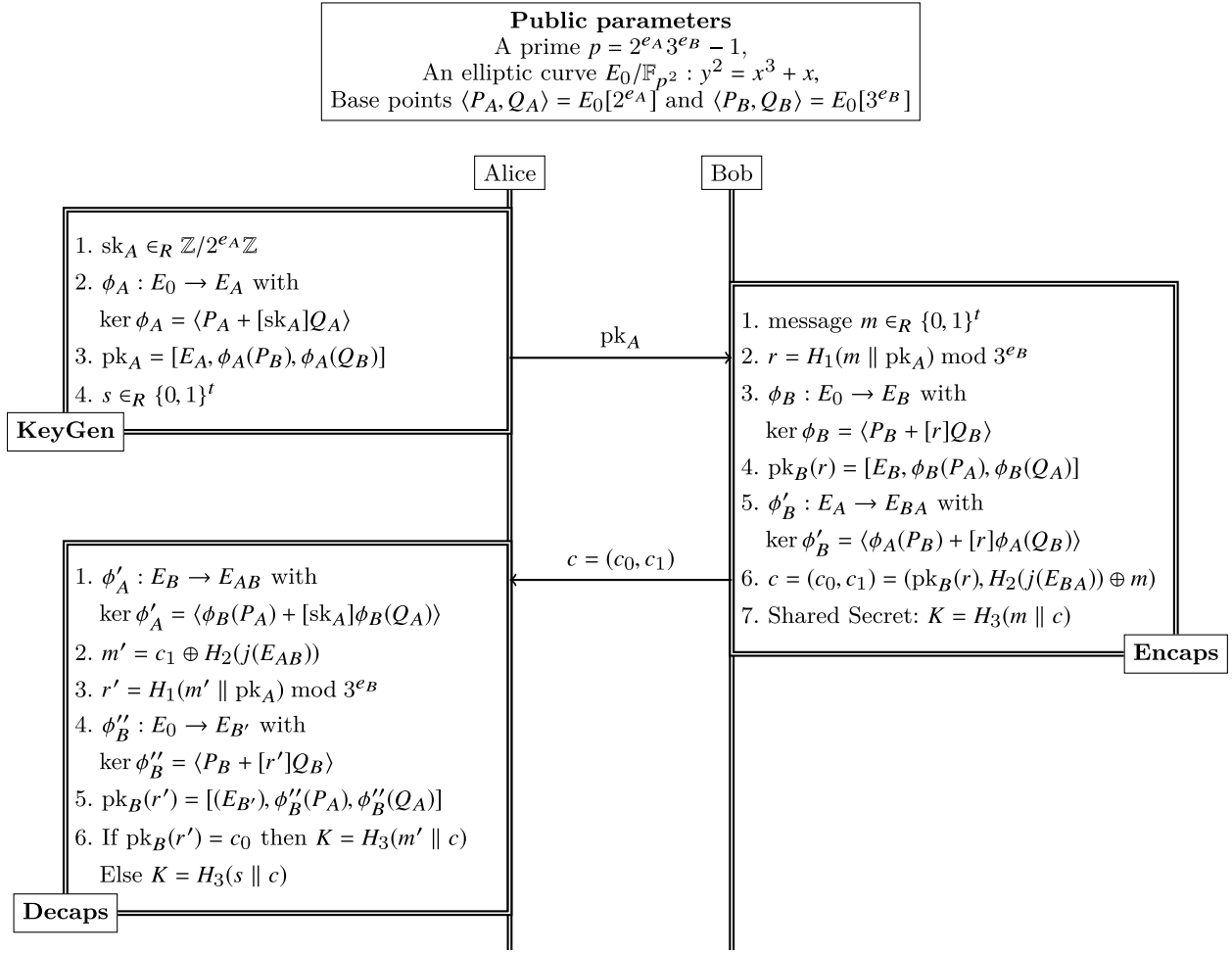


Fig. 3. SIKE mechanism.

degree 2 and 3 in [22], improved the SIKE's key compression overhead significantly compared to the previous works, while the public-key and ciphertext size are reduced by 59%.

SIKE's key compression feature offers a flexibility in protocol design for key encapsulation mechanism in different environments: the required bandwidth can be shrunk down almost in half if the performance is not a bottleneck.

Recall from SIDH key exchange that Alice's public key is $\{E_A, \phi_A(P_B), \phi_A(Q_B)\}$ and similarly Bob's public key is $\{E_B, \phi_B(P_A), \phi_B(Q_A)\}$. We can shortly represent those public keys $\{E, \phi(P), \phi(Q)\}$. That is, the public key consists of an elliptic curve over $\mathbb{F}_{p^2}$ and two points over $\mathbb{F}_{p^2}$ on this curve. However, as the public keys are similar in SIKE mechanism, they are encoded by x -coordinates of three points (see [4] for details):

$$\{X_{\phi(P)}, X_{\phi(Q)}, X_{\phi(P-Q)}\}$$

of size $6 \log p$ in total. Note that this encoding can be deterministically converted to original public key.

It is further improved by Azarderakhsh *et al.* [6], by sending in the form

$$\{j(E) \in \mathbb{F}_{p^2} \text{ and } a_1, a_2, b_1, b_2 \in \mathbb{Z}_{3^n}\},$$

so that $\phi(P) = a_1 R_1 + a_2 R_2$ and $\phi(Q) = b_1 R_1 + b_2 R_2$ for some pre-shared canonical basis $\langle R_1, R_2 \rangle = 3^n$. In that case, it reduces $4 \log p$ total key size bits but 10 times slower compression/decompression. Moreover, Costello *et al.* [11] reduces the total key size $3.5 \log p$ bits but 2.4 times slower compression/decompression by sending in the form

$$\{j(E) \in \mathbb{F}_{p^2} \text{ and } \alpha, \beta, \gamma \in \mathbb{Z}_{3^n}\},$$

so that $\alpha = b_1 a_1^{-1}, \beta = a_2 a_1^{-1}, \gamma = b_2 a_1^{-1}$. This compression methods are also given in Figure 4. Together with the works of Zanon *et al.* [26] and Naehrig and Renes [22], compression/decompression running time is reduced significantly. The running times for actual operation required for key compression can be seen in Table 2 of [22]. In this paper, we used the latest library SIDHv3.2 which includes the latest improvement by [22].

III. TARGET ARCHITECTURE

ARMv8 Cortex-A, or simply ARMv8, is the latest generation of ARM architectures targeted at the "application" profile. It includes the typical 32-bit architecture, called "AArch32", and advanced 64-bit architecture named "AArch64" with its

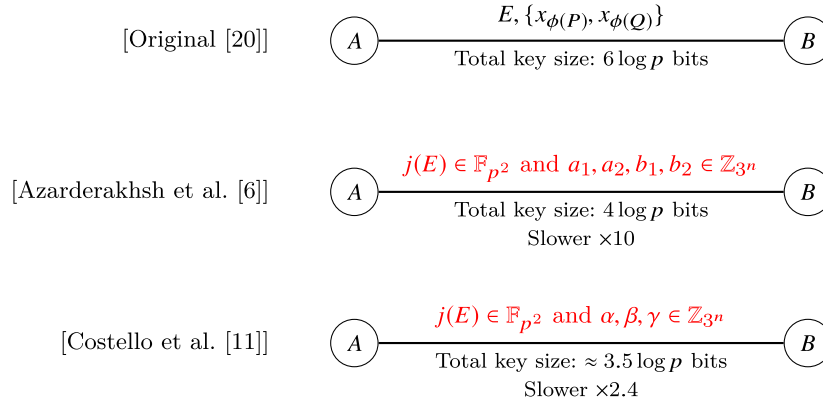


Fig. 4. Key compression.

TABLE I
COMPARISON OF PUBLIC KEY SIZE BETWEEN STANDARD
AND COMPRESSED VERSIONS

NIST Security Level	Prime	Public Key Size (in bytes)	
		Standard	Compressed
Level I	SIKEp434	330	196
Level II	SIKEp503	378	224
Level III	SIKEp610	462	273
Level V	SIKEp751	564	331

TABLE II
COMPARISON OF MULTIPLICATION METHODS, IN TERMS OF THE NUMBER
OF ADDITION OPERATIONS DEPENDING ON THE NUMBER OF WORD

Method	3	4	5	6	7
Operand Scanning	24	40	60	84	112
Product Scanning	27	48	75	108	147

associated instruction set “A64” [3]. AArch32 preserves backwards compatibility with ARMv7 and supports the so-called “A32” and “T2” instructions sets, which correspond to the traditional 32-bit and Thumb instruction sets, respectively. AArch64 comes equipped with 31 general purpose 64-bit registers (i.e. X0~X30) and one zero register (i.e. XZR), and an instruction set supporting 32-bit and 64-bit operations. The significant register expansion means that with AArch64 the maximum register capacity is expanded to 1,984 bits (i.e. 31×64 , a 4x increase with respect to ARMv7.).

ARMv8 processors started to dominate the smartphone market soon after their first release in 2011, and nowadays they are widely used in various high-end smartphones (e.g. Apple iPhone, Huawei Mate and Samsung Galaxy series). Since this architecture is used primarily in embedded systems and smartphones, efficient and compact implementations are of special interest.

ARMv8 processor supports powerful 64-bit wise unsigned integer multiplication instructions. Our implementation of modular multiplication uses the AArch64 architecture and makes extensive use of the following multiply instructions:

- MUL (unsigned multiplication, low part):
MUL X0, X1, X2 computes $X0 \leftarrow (X1 \times X2) \bmod 2^{64}$.

- UMULH (unsigned multiplication, high part):
UMULH X0, X1, X2 computes $X0 \leftarrow (X1 \times X2)/2^{64}$.

The two instructions above are required to compute a full 64-bit multiplication of the form $128\text{-bit} \leftarrow 64 \times 64\text{-bit}$, namely, the MUL instruction computes the lower 64-bit half of the product, while UMULH computes the higher 64-bit half.

For addition and subtraction operations, ADDS and SUBS instructions ensure 64-bit wise results, respectively. Detailed descriptions are as follows:

- ADDS (unsigned addition):
ADDS X0, X1, X2 computes {CARRY, X0} $\leftarrow (X1 + X2)$.
- SUB (unsigned subtraction):
SUBS X0, X1, X2 computes {BORROW, X0} $\leftarrow (X1 - X2)$.

IV. OPTIMIZED FIELD ARITHMETIC IMPLEMENTATION

There are a number of works in the literature that study the ARMv8 instructions to implement multi-precision multiplication or the full Montgomery multiplication for “SIDH friendly” modulus [18], [19], [24]. In [18], Jalali et al. implemented 751-bit and 964-bit finite field multiplication. They utilized the Comba method (i.e. column-wise multiplication) for both cases [10]. In particular, they used 2-level Karatsuba for 964-bit finite field multiplication, which shows 23.9% performance enhancements than conventional Comba method. In [24], Seo et al. optimized the 503-bit finite field multiplication for SIKEp503. They also used the Comba method with 2-level Karatsuba method to enhance the performance of multiplication. Furthermore, they optimized the MAC (Multiplication Accumulation) routines to avoid pipeline stalls. In [17], Jalali et al. presented the optimized Montgomery multiplication by mixing AArch64 and ASIMD instructions. This approach shows the better performance than Comba method, when the operand size is long enough, such as SIKEp751 and SIKEp964.

Recently, two novel SIKE protocols (i.e. SIKEp434 and SIKEp610) for NIST Post Quantum Cryptography competition round 2 were suggested, which meet NIST security level 1 and 3, respectively [4]. However, previous works do not show the optimized results for both protocols. In this paper,

we show the first practical implementations of SIKEp434 and SIKEp610 protocols on 64-bit ARMv8-A processors. In order to achieve high performance, the arithmetic for SIKEp434 and SIKEp610 is optimized to utilize the ARMv8-A ability fully. Furthermore, we also include the performance of compressed SIKEp434 and compressed SIKEp610. To describe the multi-precision arithmetic, we used following notations. Let A and B be operands of length m bits each. Each operand is written as $A = (A[n-1], \dots, A[1], A[0])$ and $B = (B[n-1], \dots, B[1], B[0])$, where $n = \lceil m/w \rceil$ is the number of words to represent operands, m is operand length, and w is the computer word size (i.e. 64-bit). The addition result ($C = A + B$) is represented as $C = (C[n-1], \dots, C[1], C[0])$. For the multiplication ($C = A \times B$), the result is represented as $C = (C[2n-1], \dots, C[1], C[0])$.

A. Finite Field Addition and Subtraction

Finite field addition and subtraction operations firstly need to perform addition and subtraction operations, respectively (See Section 2.2.1 of [15] for details). Afterward, intermediate results are reduced through the reduction routine, when the carry or borrow bit is detected. In order to avoid the timing attack, the reduction routine is performed without conditional statements (i.e. constant timing). To achieve this property, we used the masked modular reduction approach, which always perform regular routines, regardless of the carry or borrow bit (See Section 4.4 of [23] for details). When the carry or borrow bit is detected, the mask value in word is set to $2^{64} - 1$. Otherwise, the mask value is set to 0. With the mask value, the modulus is determined, whether it is modulus value or 0.

For the 434-bit addition or subtraction operation, we utilized 14 general purpose registers to store the operands (i.e. $2 \times \lceil 434/64 \rceil$) since each operand requires 7 registers. In particular, two words of 434-bit modulus are $2^{64} - 1$ (i.e. $0xFFFFFFFFFFFFFFFF$). We only set one word to $2^{64} - 1$ and use this twice for computations, which reduces one operand setting overheads. For 610-bit addition or subtraction operation, we utilized 20 general purpose registers to retain all operands (i.e. $2 \times \lceil 610/64 \rceil$) since each operand requires 10 registers. Similarly, three words of 610-bit modulus are set to $2^{64} - 1$ (i.e. $0xFFFFFFFFFFFFFFFF$). This word is used three times with only one memory access, which reduces two operand setting overheads.

B. Multiplication

In previous works, they used the Comba method (i.e. column-wise method) to improve the multi-precision multiplication. The Comba method performs the partial products in column-wise, which ensures small number of registers for maintaining the intermediate results. In Figure 5, the part of Multiplication ACculmuation (MAC) routine in column-wise method for 64-bit ARMv8 processors is described. The example performs the three partial products ($A[i] \times B[j]$, $A[i+1] \times B[j-1]$, and $A[i+2] \times B[j-2]$) and accumulates them to intermediate results. In each MAC routine, two multiplication (MUL_LOW and MUL_HIGH) and three addition

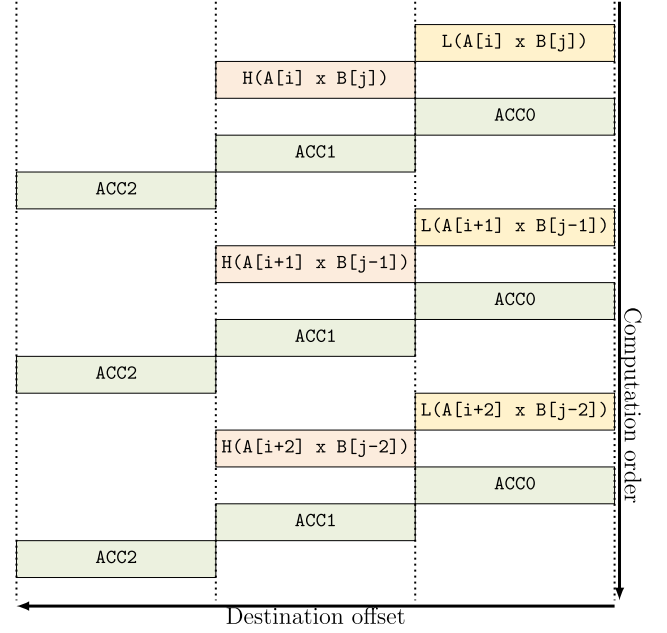


Fig. 5. Part of column-wise multiplication for ARMv8, where L , H , and ACC represent lower multiplication, higher multiplication, and accumulation, respectively.

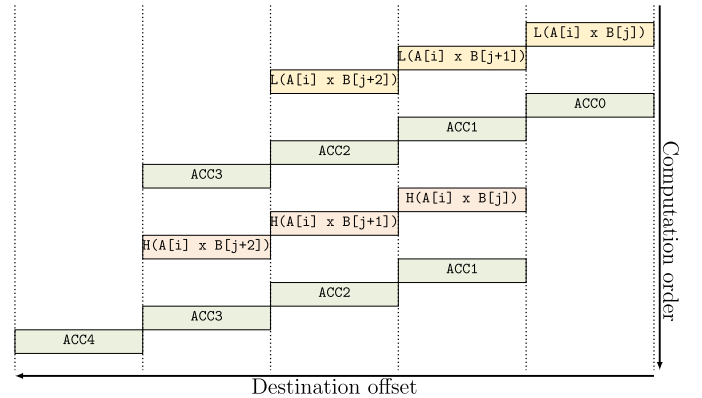


Fig. 6. Part of row-wise multiplication for ARMv8, where L , H , and ACC represent lower multiplication, higher multiplication, and accumulation, respectively.

operations (ACC0, ACC1, and ACC2) are required. For one word multiplication, we need three addition operations. For that reason, n -word multiplication requires $3 \times n^2$ addition operations.

In this work, we target the relatively shorter modulus (i.e. 434-bit) than previous works (i.e. 751-bit or 964-bit). We decide to use the row-wise multiplication, which requires $2n + 2$ registers ($n + 1$ for operands and $n + 1$ for intermediate results), where n , m , and w are $\lceil m/w \rceil$, operand length, and word size, respectively. Under the 64-bit processor setting, the n is set to 7 for 434-bit ($\lceil 434/64 \rceil$). Considering that ARMv8 supports 31 64-bit registers, the required number of registers for 434-bit can be retained in registers. In Figure 6, the part of MAC routine in row-wise method for 64-bit ARMv8 processors is described. The example performs three partial products ($A[i] \times B[j]$, $A[i] \times B[j+1]$, and

$A[i] \times B[j+2]$) and accumulates them to intermediate results. The number of addition for three partial products in Figure 6 are 8 (i.e. $2 \times (n+1)$ where n is 3.). For the n -word multiplication, it requires $2 \times n \times (n+1)$ addition operations. The comparison of multiplication methods in terms of the number of addition operations depending on the number of word are given in Table II. Compared with the column-wise method (i.e. product-scanning), the row-wise method (i.e. operand-scanning) requires less number of addition operations for accumulation routines. For the 7-word case (i.e. 434-bit), the row-wise method reduces the number of addition operations by 35 times than the column-wise method. The multiplication is performed in original row-wise multiplication rather than row-wise multiplication with Karatsuba method. The Karatsuba method is also working for 7-word case but it generates a number of sub-routines to perform the computation and store the intermediate results, which requires additional operations and memory accesses [21].

For the 610-bit multiplication, the operands $A = (A[9], \dots, A[0])$ and $B = (B[9], \dots, B[0])$ need 20 64-bit registers. Except operands, we also need registers for intermediate results and temporal storage. Due to limited number of registers, we only maintain half number of operands in registers and load remaining operands on demand.

Karatsuba's method reduces a multiplication of two m -bit operands to three $\frac{m}{2}$ -bit multiplications with some addition or subtraction operations. There are two approaches to perform Karatsuba multiplication, including additive Karatsuba and subtractive Karatsuba. Taking the multiplication of n -word operand A and B as an example, operands are represented as $A = A_H \cdot 2^{\frac{n}{2}} + A_L$ and $B = B_H \cdot 2^{\frac{n}{2}} + B_L$. The multiplication $P = A \cdot B$ can be computed according to the following equation by using additive Karatsuba's method:

$$A_H \cdot B_H \cdot 2^n + [(A_H + A_L)(B_H + B_L) - A_H \cdot B_H - A_L \cdot B_L] \cdot 2^{\frac{n}{2}} + A_L \cdot B_L \quad (1)$$

or subtractive Karatsuba's method:

$$A_H \cdot B_H \cdot 2^n + [A_H \cdot B_H + A_L \cdot B_L - |A_H - A_L| \cdot |B_H - B_L|] \cdot 2^{\frac{n}{2}} + A_L \cdot B_L \quad (2)$$

For the 610-bit multiplication, 1-level subtractive Karatsuba multiplication is used, which consists of 3 320-bit multiplication operations with some addition or subtraction operations.

In the beginning, we compute the lower 320-bit multiplication $R_L \leftarrow A[4 \sim 0] \cdot B[4 \sim 0]$ using the row-wise method that requires 25 MUL, 25 UMULH and 52 addition instructions for accumulating the partial products. Second, we compute the higher 310-bit multiplication $R_H \leftarrow A[9 \sim 5] \cdot B[9 \sim 5]$, similarly. Third, we compute the subtractions and absolute values $|A[4 \sim 0] - A[9 \sim 5]|$ and $|B[4 \sim 0] - B[9 \sim 5]|$ and proceed to the last 310-bit multiplication $R_M \leftarrow |A[4 \sim 0] - A[9 \sim 5]| \cdot |B[4 \sim 0] - B[9 \sim 5]|$. Finally, we obtain the result by performing the accumulation step $R_H \cdot 2^{610} + (R_L + R_H - R_M) \cdot 2^{310} + R_L$. Since the multiplication uses all available registers, 12 callee-saved registers ($X19 \sim X30$) are stored into the stack. The multiplication is also designed to reduce the pipeline stalls. The multiplication and addition/subtraction

operations use different instruction group. They can hide each others costs. Based on the above observation, we engineer a multi-precision multiplication to hide the addition costs into the multiplication. At the lowest level, we implement multi-precision multiplication using the row-wise method based on the following multiplication/addition instruction sequence:

```

:
MUL   X7, X6, X2
ADCS  X18, X18, X13
MUL   X8, X6, X3
ADCS  X19, X19, X14
MUL   X9, X6, X4
ADCS  X20, X20, X15
MUL   X10, X6, X5
ADCS  X21, X21, X16
:

```

We ensure that the destination of MUL instruction is not used for the source of following ADCS instructions. This approach avoids the pipeline stalls. Second, MUL and ADCS instructions are performed one by one to hide the each cost.

C. Reduction

In this section, we adapt techniques described in previous sections to implement modular multiplication for SIKE. Specifically, we target parameter sets based on SIKEp434 and SIKEp610 [4].

Multi-precision modular multiplication is the most expensive operation for the implementation of SIKE [12], [20]. In particular, Montgomery multiplication for SIKE can be efficiently exploited and further simplified by taking advantage of so-called "Montgomery-friendly" modulus. The advantage of using Montgomery multiplication for "SIDH-friendly" primes was recently confirmed by Bos and Friedberger [7], who studied and compared different approaches, including Barrett reduction. Recent works by Seo et al also utilized the Montgomery multiplication for SIKEp503 protocols [24].

Based on the observation above, we choose the Montgomery multiplication to implement SIDH-friendly modular arithmetic for SIKEp434 and SIKEp610 protocols. The approach reduces almost half of partial products since the lower part of modulus is set to 0. To reduce memory accesses, we keep as many results as possible in registers. Since Montgomery multiplication performs partial products with modulus and quotient (m'), we maintained all quotients in registers and used them, directly. The technique reduces the $2 \times (n+1)$ number of memory accesses for $n+1$ load and $n+1$ store operations, where n -word computation. For instance, 16 and 22 memory accesses for SIKEp434 and SIKEp610 are optimized.

1) *Interleaved Montgomery Multiplication:* The Algorithm 1 describes Montgomery multiplication. The Montgomery multiplication firstly performs the operand multiplication. Second, the quotient (Q) is generated by multiplying the

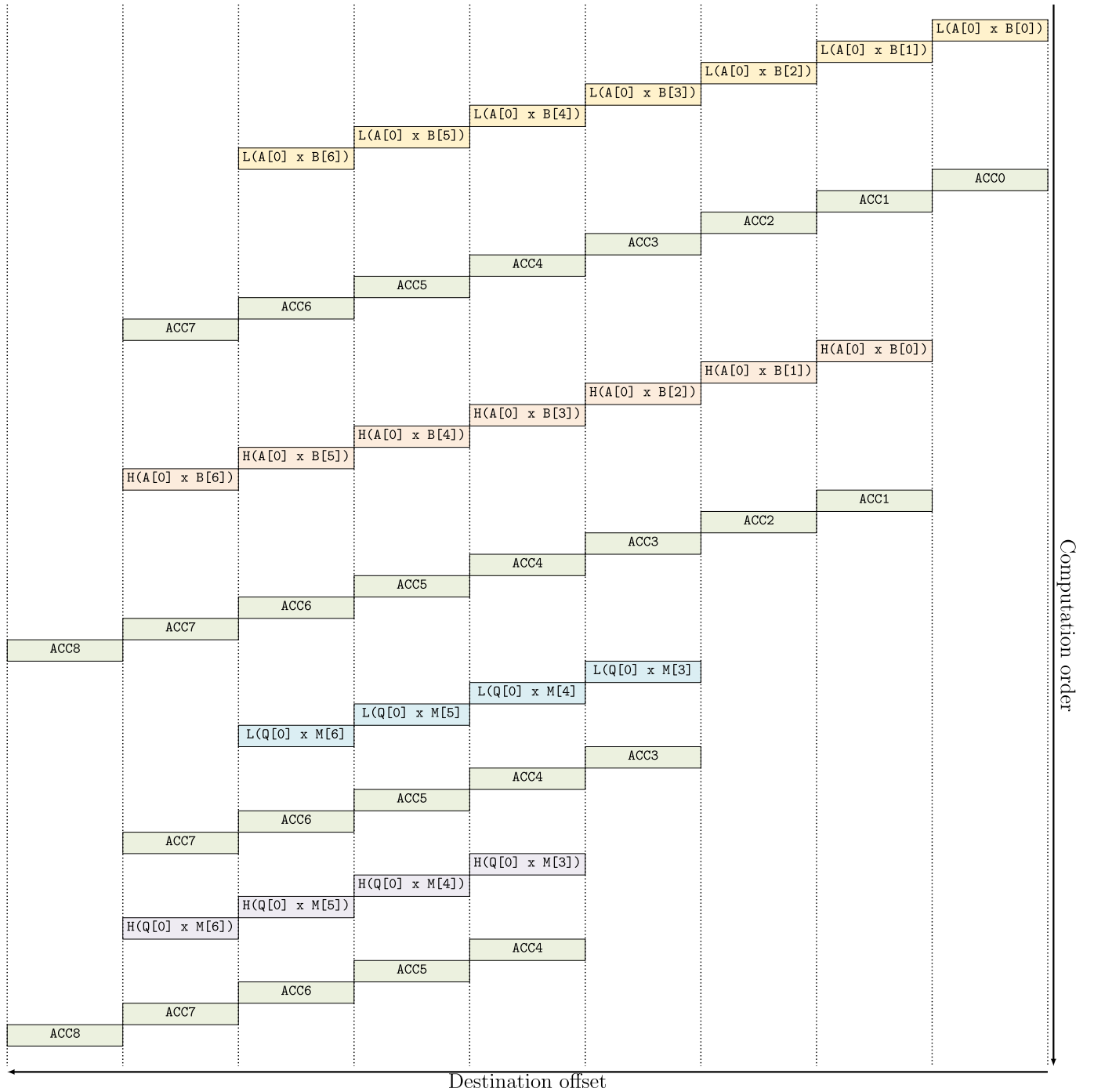


Fig. 7. Part of interleaved Montgomery multiplication for SIKEp434, where L , H , and ACC represent lower multiplication, higher multiplication, and accumulation, respectively.

intermediate result (T) and modulus inverse constant (M'). Third, the quotient (Q) is multiplied by the modulus (M) and the result is added to the intermediate result (T).

The separated Montgomery multiplication performs the multiplication (Step 1) and reduction (Step 2 and Step 3) in separated way. The alternative way to implement the Montgomery multiplication is interleaved way. The part of multiplication is performed and the intermediate result is directly reduced. Compared with the separated approach, this method optimizes the number of memory accesses for intermediate

result loading and storing. Total $2n$ memory accesses are optimized (n for memory loading and n for memory storing). We utilized the interleaved Montgomery multiplication for SIKEp434, since all operands and intermediate results are well retained in general purpose registers. ARMv8 processors support 31 general purpose registers and we utilized 30 registers. In particular, 4, 1, 7, 10, and 8 registers are allocated for modulus, operand A , operand B , intermediate result, and temporal storage, respectively. The part of interleaved Montgomery multiplication for SIKEp434 is given in Figure 7.

TABLE III

COMPARISON OF IMPLEMENTATIONS OF THE SIKEp434 AND SIKEp610 ARITHMETIC ON ARMv8 RMv8 CORTEX-A55@1.766GHz AND CORTEX-A75@2.803GHz BASED PROCESSOR. TIMINGS ARE REPORTED IN TERMS OF CLOCK CYCLES

Implementation	Language	Processor	Protocol	Timings [cc] (Performance Improvement)			
				$\mathbb{F}_p$ add	$\mathbb{F}_p$ sub	$\mathbb{F}_p$ mul	$\mathbb{F}_p$ inv
SIKE (SIDH-v3.2) [13]	C	Cortex-A55	SIKEp434	171	127	2976	1572510
This work	ASM			74 (56%)	67 (47%)	602 (79%)	340425 (78%)
SIKE (SIDH-v3.2) [13]	C	Cortex-A75		272	202	4723	2495892
This work	ASM			34 (87%)	28 (86%)	561 (88%)	309359 (87%)
SIKE (SIDH-v3.2) [13]	C	Cortex-A55	SIKEp610	249	182	6093	4469981
This work	ASM			122 (51%)	111 (39%)	1287 (78%)	955305 (78%)
SIKE (SIDH-v3.2) [13]	C	Cortex-A75		224	154	5673	4156698
This work	ASM			50 (77%)	42 (72%)	1046 (81%)	777992 (81%)

Algorithm 1 Calculation of the Montgomery Multiplication

Require: An odd m -bit modulus M , Montgomery radix $R = 2^m$, an operand T where operands (A and B in the range $[0, m - 1]$), and pre-computed constant $M' = -M^{-1} \bmod R$

Ensure: Montgomery product $Z = \text{MonMul}(A, B, R) = (A \times B) \cdot R^{-1} \bmod M$

- 1: $T \leftarrow A \cdot B$
- 2: $Q \leftarrow T \cdot M' \bmod R$
- 3: $Z \leftarrow (T + Q \cdot M)/R$
- 4: **return** Z

The first word of operand A is multiplied by all operand B ($B[0] \sim B[6]$) and the intermediate result is accumulated. Second, the first word of quotient Q is multiplied by modulus and the intermediate result is accumulated. With this approach, we can optimize the memory access to the intermediate result.

For SIKEp610 implementation, we used the separated Montgomery multiplication since the required number of registers is over available registers, which introduces a number of memory accesses.

2) *Shifted Modulus*: The execution timing of modular reduction is relied on the number of partial products. For this reason, reducing the number of partial products can lead to the performance enhancements. The general form of modulus for SIKEp610 is as follows.

Normal representation

```

M[4] 0x6E02000000000000
M[5] 0xB1784DE8AA5AB02E
M[6] 0x9AE7BF45048FF9AB
M[7] 0xB255B2FA10C4252A
M[8] 0x819010C251E7D88C
M[9] 0x000000027BF6A768

```

This requires 6 registers to retain all modulus. The number of required registers can be optimized with the shifted representation. The least significant word of modulus ($M[4]$) only utilize the 15-bit out of 64-bit and the most significant word of modulus ($M[9]$) has 30-bit empty space. We shifted the modulus by 16-bit to the left and the representation is

re-written as follows.

Shifted representation

```

M[4] 0x4DE8AA5AB02E6E02
M[5] 0xBF45048FF9ABB178
M[6] 0xB2FA10C4252A9AE7
M[7] 0x10C251E7D88CB255
M[8] 0x00027BF6A7688190

```

The shifted representation only requires 5 registers for SIKEp610 modulus. This optimized the number of partial products by 10 ($60 \rightarrow 50$). The shifted computation utilizes the shifted representation. The result is obtained in two steps. First, the ordinary multiplication is performed. Second, the intermediate result is shifted. For SIKEp610, we shifted 16-bit to the left and the result should be reordered after computations. The reorder computation is described as follows.

```

:
LSL X7, X22, #48
ORR X7, X7, X21, LSR#16
:

```

The higher word ($X22$) is shifted to the left by 48-bit and the lower word ($X21$) is shifted to the right by 16-bit. To optimize the shift and addition operation, we utilized the barrel-shifter module, which can perform the shift operation on the second operand without additional costs.

For the SIKEp434 case, the modulus $(p + 1)$ is divided into 4-word. The shifted modulus needs to ensure that the summation of remaining bits of the least significant word and the most significant word is over 64-bit. However, the SIKEp434 case only has 39-bit (14-bit from the most significant word and 25-bit from the least significant word). The following is the modulus of SIKEp434.

```

M[3] 0xFDC1767AE3000000
M[4] 0x7BC65C783158AEA3
M[5] 0x6CFC5FD681C52056
M[6] 0x0002341F27177344

```

TABLE IV

COMPARISON OF IMPLEMENTATIONS OF THE SIKE PROTOCOLS ON ARMv8 CORTEX-A55@1.766GHz AND CORTEX-A75@2.803GHz BASED PROCESSORS. TIMINGS ARE REPORTED IN TERMS OF MILLISECONDS AND CLOCK CYCLES. ^m: MIXED APPROACH

Processor	Protocol		Implementation	Language	Timings [milliseconds]				Timings [cc × 10 ⁶]			
					KeyGen	Encaps	Decaps	Total	KeyGen	Encaps	Decaps	Total
Cortex-A55	Standard	SIKEp434	SIKE (SIDH-v3.2) [13]	C	62.2	101.7	108.5	210.2	109.8	179.5	191.6	371.2
			This work	ASM	15.4	25.3	27.0	52.4	27.2	44.8	47.7	92.5
		SIKEp503	SIKE (SIDH-v3.2) [13]	C	95.5	157.2	167.4	324.6	168.6	277.6	295.7	573.2
			SIKE (SIDH-v3.2) [13]	ASM	21.9	35.3	38.0	73.4	38.6	62.4	67.2	129.6
			Jalali et al. [17]	ASM	27.6	45.1	48.2	93.4	48.7	79.7	85.2	164.9
			Jalali et al. [17] ^m	ASM	39.6	64.8	69.2	134.0	69.9	114.4	122.3	236.7
		SIKEp610	SIKE (SIDH-v3.2) [13]	C	181.4	333.7	335.6	669.3	320.4	589.2	592.7	1181.9
			This work	ASM	43.5	79.9	80.7	160.6	76.8	141.2	142.4	283.6
		SIKEp751	SIKE (SIDH-v3.2) [13]	C	328.3	532.2	571.7	1103.9	579.8	939.9	1009.7	1949.6
			SIKE (SIDH-v3.2) [13]	ASM	74.4	118.3	128.5	246.8	131.3	208.9	226.9	435.8
			Jalali et al. [17]	ASM	85.3	137.9	148.5	286.3	150.6	243.5	262.2	505.6
			Jalali et al. [17] ^m	ASM	102.9	165.7	178.8	344.5	181.7	292.7	315.7	608.3
	Compressed	SIKEp434	SIKE (SIDH-v3.2) [13], [22]	C	102.6	159.0	152.0	311.0	181.2	280.8	268.5	549.3
			This work [22]	ASM	25.2	39.3	37.0	76.3	44.5	69.3	65.3	134.7
		SIKEp503	SIKE (SIDH-v3.2) [13], [22]	C	156.6	242.3	228.1	470.4	276.5	427.9	402.8	830.8
			SIKE (SIDH-v3.2) [13], [22]	ASM	35.9	56.3	52.8	109.1	63.4	99.4	93.3	192.7
		SIKEp610	SIKE (SIDH-v3.2) [13], [22]	C	314.4	461.8	451.5	913.3	555.2	815.6	797.3	1612.9
			This work [22]	ASM	74.0	110.1	108.2	218.4	130.7	194.5	191.1	385.6
		SIKEp751	SIKE (SIDH-v3.2) [13], [22]	C	528.9	834.2	780.9	1615.1	934.0	1473.1	1379.1	2852.2
			SIKE (SIDH-v3.2) [13], [22]	ASM	117.2	190.6	178.2	368.8	207.0	336.6	314.7	651.3
Cortex-A75	Standard	SIKEp434	SIKE (SIDH-v3.2) [13]	C	36.4	59.6	63.6	123.2	102.1	167.1	178.3	345.4
			This work	ASM	8.1	13.4	14.3	27.6	22.8	37.5	40.0	77.5
		SIKEp503	SIKE (SIDH-v3.2) [13]	C	56.4	92.9	98.7	191.6	158.0	260.3	276.8	537.1
			SIKE (SIDH-v3.2) [13]	ASM	9.6	15.8	16.9	32.7	27.0	44.3	47.3	91.6
			Jalali et al. [17]	ASM	11.6	19.1	20.3	39.4	32.5	53.5	57.0	110.5
			Jalali et al. [17] ^m	ASM	14.2	23.2	24.8	48.1	39.8	65.2	69.6	134.8
		SIKEp610	SIKE (SIDH-v3.2) [13]	C	106.6	196.3	197.4	393.7	298.8	550.3	553.2	1103.5
			This work	ASM	21.0	38.8	39.0	77.8	58.9	108.7	109.3	218.0
		SIKEp751	SIKE (SIDH-v3.2) [13]	C	193.3	313.6	336.7	650.4	541.7	879.1	943.9	1823.0
			SIKE (SIDH-v3.2) [13]	ASM	32.3	52.3	56.3	108.6	90.6	146.5	157.8	304.3
			Jalali et al. [17]	ASM	38.6	62.6	67.3	130.0	108.3	175.6	188.8	364.4
			Jalali et al. [17] ^m	ASM	38.7	62.5	67.3	129.9	108.4	175.2	188.8	364.0
	Compressed	SIKEp434	SIKE (SIDH-v3.2) [13], [22]	C	59.5	93.2	87.6	180.8	166.7	261.2	245.5	506.8
			This work [22]	ASM	13.3	20.7	19.8	40.6	37.3	58.1	55.6	113.7
		SIKEp503	SIKE (SIDH-v3.2) [13], [22]	C	91.9	142.7	137.3	280.0	257.7	400.1	384.7	784.8
			SIKE (SIDH-v3.2) [13], [22]	ASM	16.2	24.8	23.4	48.2	45.4	69.6	65.6	135.2
		SIKEp610	SIKE (SIDH-v3.2) [13], [22]	C	182.2	269.8	265.2	535.0	510.7	756.2	743.4	1499.5
			This work [22]	ASM	36.1	53.1	52.1	105.2	101.1	148.8	146.1	294.9
		SIKEp751	SIKE (SIDH-v3.2) [13], [22]	C	309.1	489.9	468.1	957.9	866.4	1373.1	1312.0	2685.1
			SIKE (SIDH-v3.2) [13], [22]	ASM	51.6	82.9	77.4	160.3	144.6	232.4	216.9	449.3

V. PERFORMANCE RESULT

In this section, we evaluate the performance of proposed implementations for 64-bit ARMv8-A processors. All our finite field arithmetic implementations were written in assembly language and compiled with optimization level -O3.

We implemented the multi-precision multiplication algorithm described in Section IV-B and Montgomery reduction in Section IV-C. We integrated our implementation of the Montgomery multiplication for ARMv8-A into the SIKE round 2 library [4].

Table III summarizes results of different software implementations of the SIKEp434 and SIKEp610 arithmetic on

ARMv8-A processor: a 1.536GHz ARM Cortex-A53 processor. Since this is first work for SIKEp434 and SIKEp610 on ARMv8-A based processors, we compare results with the SIKE round 2 reference code [13]. The *unoptimized* reference implementation is written in C using the SIKE round 2 library [4]. In this case, proposed arithmetic implementations show much higher performance than reference work. Finite field addition for SIKEp434 and SIKEp610 shows performance enhancements by 2.67x/8.77x and 2.39x/4.66x on Cortex-A55/Cortex-A75, respectively. Finite field subtraction for SIKEp434 and SIKEp610 shows performance enhancements by 2.28x/8.08x and 1.93x/3.94x

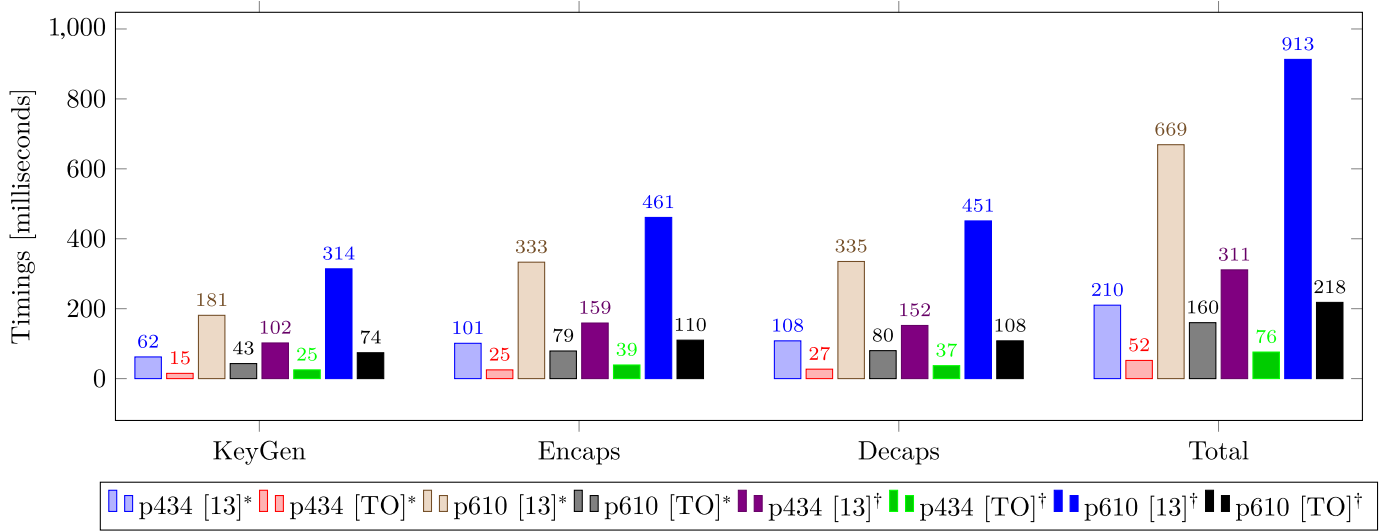


Fig. 8. Comparison of implementations of the SIKE protocols on ARMv8 Cortex-A55@1.766GHz based processors. Timings are reported in terms of milliseconds. *: standard version, †: compressed version, TW: this work.

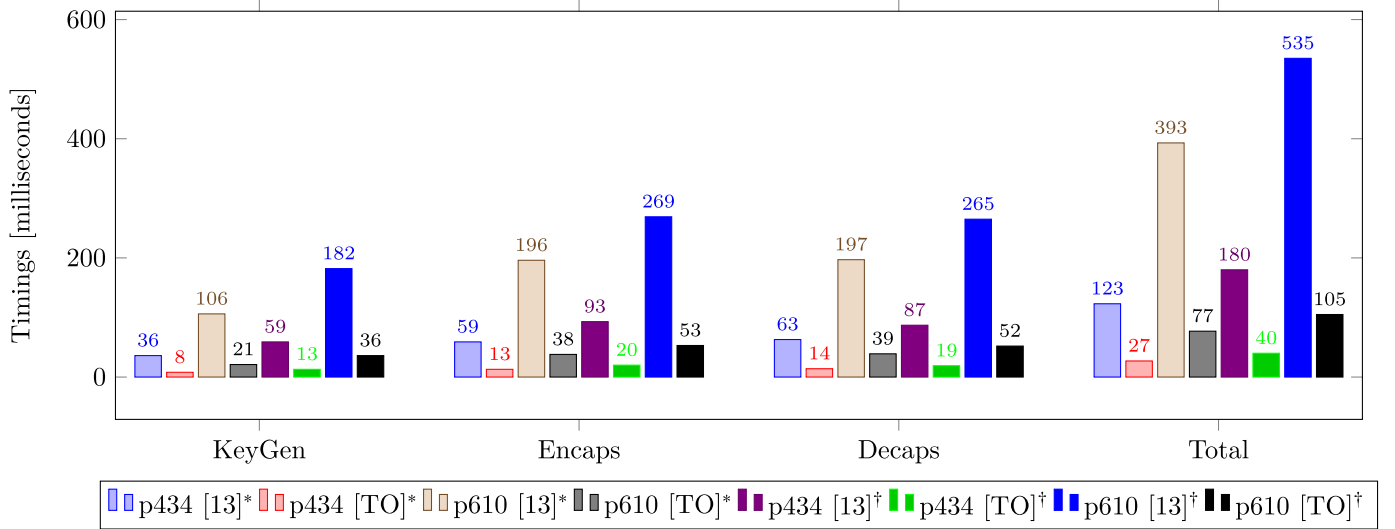


Fig. 9. Comparison of implementations of the SIKE protocols on ARMv8 Cortex-A75@2.803GHz based processors. Timings are reported in terms of milliseconds. *: standard version, †: compressed version, TW: this work.

on Cortex-A55/Cortex-A75, respectively. Finite field multiplication for SIKEp434 and SIKEp610 shows performance enhancements by 5.01x/8.41x and 4.76x/5.45x on Cortex-A55/Cortex-A75, respectively. Finite field inversion for SIKEp434 and SIKEp610 shows performance enhancements by 4.66x/8.07x and 4.70x/5.36x on Cortex-A55/Cortex-A75, respectively.

As described above, performance enhancements are observed in both ARMv8 processors but Cortex-A75 shows the better performance improvements than Cortex-A55. This difference comes from their different computer architectures. Cortex-A55 supports 2-wide decode in-order superscalar pipeline while Cortex-A75 supports 3-wide decode out-of-order superscalar pipeline. The assembly code is optimized further through out-of-order superscalar pipeline of Cortex-A75. Between SIKEp434 and SIKEp610, SIKEp434 is

optimized further than SIKEp610 since the implementation of SIKEp434 can take advantage of the optimal register usage due to the short operand.

Table IV summarizes results of different software implementations of the SIKEp434 and SIKEp610 protocols on ARMv8 Cortex-A55 and Cortex-A75 processors. For standard SIKE protocols, proposed implementations of SIKEp434 and SIKEp610 outperform previous works by 3.76x/4.04x and 3.98x/4.57x for Cortex-A55/Cortex-A75 processors, respectively. Considering that target processors are working at 1.766GHz and 2.803GHz, SIKEp434 and SIKEp610 require only 0.055/0.030 and 0.168/0.086 seconds for Cortex-A55/Cortex-A75 processors, respectively. Compared with other SIKE protocols, the SIKEp434 shows the highest performance and the SIKEp751 shows the lowest performance. We also evaluated the key compressed SIKE protocols.

For compressed SIKE protocols, proposed implementations of SIKEp434 and SIKEp610 outperform previous works by 4.07x/4.45x and 4.18x/5.08x for Cortex-A55/Cortex-A75 processors, respectively. Key compressed SIKEp434 and SIKEp610 require only 0.076/0.040 and 0.218/0.105 seconds for Cortex-A55@1.766GHz/Cortex-A75@2.803GHz processors, respectively. Compressed versions require more timing than standard SIKE protocols but the execution timing is reasonably fast enough for real world applications.

The performance comparison is given in Figure 8 and 9 for Cortex-A55 and Cortex-A75, respectively. The fastest performance of standard version is obtained from proposed implementation of SIKEp434. For the compressed version, proposed SIKEp434 implementation also achieved the fastest performance.

Overall, the proposed assembly implementation achieved significant performance improvements. The assembly implementation has several advantages over C based implementation. First, the customized register utilization is available. By carefully designing variable assignments, many variables are kept in registers, which reduces the number of memory accesses. Second, the assembly implementation can handle status registers. This avoids a number of carry handling routines in C language. Third, C implementation is mainly relied on compiler's capability. If the compiler selects inefficient instructions, this leads to slow performance.

VI. CONCLUSION

This paper presented high-speed implementation of SIKE round 2 on high-end 64-bit ARMv8 Cortex-A55 and Cortex-A75 processors. A combination of several optimization methods yields very efficient modular multiplications for SIKEp434 and SIKEp610 protocols that are shown, for example, to be approximately 5.01x/8.41x and 4.76x/5.45x faster than the normal modular multiplication implementations for "SIDH-friendly" modulus on a 64-bit ARMv8 Cortex-A55/Cortex-A75 processors. The optimized implementations which push further the performance of post-quantum supersingular isogeny-based protocols, are 3.98x/4.57 faster than the previous implementations of SIKEp610, targeting the Cortex-A55/Cortex-A75 processors. Furthermore, we integrated our fast modular arithmetic implementations, compact prime SIKEp434, and optimal strategy for isogeny computations into Microsoft's SIDH library. A 128-bit full key-exchange execution over optimal prime SIKEp434 is performed in about 0.055/0.030 seconds on a ARMv8 Cortex-A55@1.766GHz/Cortex-A75@2.803GHz processors, which show the practicality of isogeny based post-quantum cryptography over mobile devices. The key compressed versions are also evaluated and SIKEp434 is performed in about 0.076/0.040 seconds on a ARMv8 Cortex-A55@1.766GHz/Cortex-A75@2.803GHz processors.

Inspired by recent IETF draft [1] on SIKE integration into Amazon's AWS services, supporting hybrid post-quantum KEM on one of the largest cloud providers in industry, this work improves the overall performance of the key encapsulation mechanism significantly and provides guidelines for

industry practitioners to enhance SIKE's performance on the high-performance ARM Cortex-A processors which are used vastly in the new generation of cellphones.

REFERENCES

- [1] (2019). *BIKE and SIKE Hybrid Key Exchange Cipher Suites for Transport Layer Security (TLS)*. Accessed: Nov. 2019. [Online]. Available: <https://tools.ietf.org/html/draft-campagna-tls-bike-sike-hybrid-00>
- [2] G. Adj, D. Cervantes-Vázquez, J.-J. Chi-Domínguez, A. Menezes, and F. Rodríguez-Henríquez, "On the cost of computing isogenies between supersingular elliptic curves," in *Proc. 25th Int. Conf. Sel. Areas Cryptogr. (SAC)*, 2018, pp. 322–343.
- [3] ARM Limited. (2017). *ARM Architecture Reference Manual ARMv8, for ARMv8-A Architecture Profile*. [Online]. Available: https://static.docs.arm.com/ddi0487/ca/DDI0487C_a_armv8_arm.pdf
- [4] R. Azarderakhsh et al. (Feb. 2019). *Supersingular Isogeny Key Encapsulation—Submission to the NIST's Post-Quantum Cryptography Standardization Process*. [Online]. Available: <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions/SIKE.zip>
- [5] R. Azarderakhsh et al. (2017). *Supersingular Isogeny Key Encapsulation—Submission to the NIST's Post-Quantum Cryptography Standardization Process*. [Online]. Available: <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/SIKE.zip>
- [6] R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel, and C. Leonardi, "Key compression for isogeny-based cryptosystems," in *Proc. 3rd ACM Int. Workshop ASIA Public-Key Cryptogr. (AsiaPKC)*, 2016, pp. 1–10.
- [7] J. W. Bos and S. Friedberger, "Fast arithmetic modulo $2^p \pm 1$," in *Proc. IEEE Symp. Comput. Arithmetic (ARITH)*, Jul. 2017, pp. 148–155.
- [8] D. X. Charles, K. E. Lauter, and E. Z. Goren, "Cryptographic hash functions from expander graphs," *J. Cryptol.*, vol. 22, no. 1, pp. 93–113, Jan. 2009.
- [9] L. Chen et al., "Report post-quantum cryptography," U.S. Dept. Commerce, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NISTIR 8105, 2016.
- [10] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM Syst. J.*, vol. 29, no. 4, pp. 526–538, 1990.
- [11] C. Costello, D. Jao, P. Longa, M. Naehrig, J. Renes, and D. Urbanik, "Efficient compression of SIDH public keys," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Paris, France: Springer, 2017, pp. 679–706.
- [12] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny Diffie-Hellman," in *Advances in Cryptology (Lecture Notes in Computer Science)*, vol. 9814, M. Robshaw and J. Katz, Eds. Santa Barbara, CA, USA: Springer, 2016, pp. 572–601.
- [13] C. Costello, P. Longa, and M. Naehrig. (2019). *SIDH Library*. [Online]. Available: <https://github.com/Microsoft/PQCrypto-SIDH>
- [14] S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti, "On the security of supersingular isogeny cryptosystems," in *Proc. 22nd Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2016, pp. 63–91.
- [15] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2006.
- [16] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the Fujisaki-Okamoto transformation," in *Proc. 15th Int. Theory Cryptogr. Conf. (TCC)* 2017, pp. 341–371.
- [17] A. Jalali, R. Azarderakhsh, M. M. Kermani, M. Campagna, and D. Jao, "ARMv8 SIKE: Optimized supersingular isogeny key encapsulation on ARMv8 processors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4209–4218, Nov. 2019.
- [18] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Supersingular isogeny Diffie-Hellman key exchange on 64-bit ARM," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 5, pp. 902–912, Sep. 2019.
- [19] A. Jalali, R. Azarderakhsh, and M. Mozaffari-Kermani, "Efficient post-quantum undeniable signature on 64-bit ARM," in *Proc. Int. Conf. Sel. Areas Cryptogr.* Ottawa, ON, Canada: Springer, 2017, pp. 281–298.
- [20] D. Jao and L. D. Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *Post-Quantum Cryptography (Lecture Notes in Computer Science)*, vol. 7071, B. Yang, Ed. Taipei, Taiwan: Springer, 2011, pp. 19–34.
- [21] P. L. Montgomery, "Five, six, and seven-term Karatsuba-like formulae," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 362–369, Mar. 2005.
- [22] M. Naehrig and J. Renes, "Dual isogenies and their application to public-key compression for isogeny-based cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 499, Dec. 2019.

- [23] H. Seo, A. Jalali, and R. Azarderakhsh, "SIKE round 2 speed record on ARM Cortex-M4," in *Proc. Int. Conf. Cryptol. Netw. Secur.* Fuzhou, China: Springer, 2019, pp. 39–60.
- [24] H. Seo, Z. Liu, P. Longa, and Z. Hu, "SIDH on ARM: Faster modular multiplications for faster post-quantum supersingular isogeny key exchange," in *Proc. IACR Trans. Cryptograph. Hardw. Embedded Syst.*, 2018, pp. 1–20.
- [25] The National Institute of Standards and Technology. (2018). *Post-Quantum Cryptography Standardization*. [Online]. Available: <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization>
- [26] G. H. M. Zanon, M. A. Simplicio, G. C. C. F. Pereira, J. Doliskani, and P. S. L. M. Barreto, "Faster key compression for isogeny-based cryptosystems," *IEEE Trans. Comput.*, vol. 68, no. 5, pp. 688–701, May 2019.



Amir Jalali received the Ph.D. degree in computer engineering from the Department of Computer, Electrical Engineering and Computer Science, Florida Atlantic University, USA, in 2018. He is currently with the Information Security Group, LinkedIn Corporation. His current research interests include applied cryptography, post-quantum cryptography, and homomorphic encryption.



Hwajeong Seo received the B.S.E.E., M.S., and Ph.D. degrees in computer engineering from Pusan National University. He is currently an Assistant Professor with Hansung University. His research interests include the Internet of Things and information security.



Pakize Sanal received the B.E. and M.E. degrees in computer engineering from Yasar University, Turkey, in 2016 and 2019, respectively. She is currently pursuing the Ph.D. degree in computer engineering with Florida Atlantic University under the supervision of Dr. Azarderakhsh. She is also a Research Assistant with I-SENSE Lab. Her research interests include efficient implementation of cryptographic algorithms and post-quantum cryptography.



Reza Azarderakhsh (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Western University, in 2011. He was a recipient of the NSERC Post-Doctoral Research Fellowship working with the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Florida Atlantic University. His current research interests include finite field and its applications, elliptic curve cryptography, pairing-based cryptography, and post-quantum cryptography. He is serving as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS.