Curve448 on 32-bit ARM Cortex-M4

Hwajeong Seo[1][0000−0003−0069−9061] and Reza Azarderakhsh[2,3]

[1]Division of IT Convergence Engineering, Hansung University, Seoul 136-792, Korea,
`hwajeong84@gmail.com`
[2]Department of Computer and Electrical Engineering and Computer Science,
Florida Atlantic University, FL, USA,
`razarderakhsh@fau.edu`
[3]PQSecure 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 32×32-bit $\rightarrow$ 64-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 ← A+B
ADC	C, A, B	Addition with Carry	C ← A+B+Carry
SUB	C, A, B	Subtraction without Carry	C ← A-B
MOV	C, A	Move 32-bit word between registers	C ← A
UMAAL	D, C, A, B	Multiplication with Accumulaion	{D|C} ← A×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 $A = (A[13], ..., A[1], A[0])$ and $B = (B[13], ..., B[1], B[0])$. The result $C = A \cdot B$ is represented as $C = (C[27], ...,$

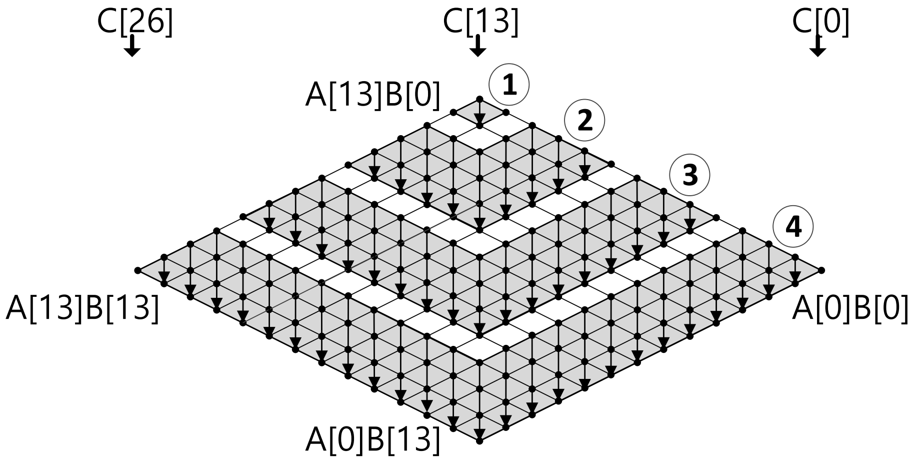

Fig. 1. 448-bit Operand Scanning multiplication at the word-level on ARM Cortex-M4 [4].

$C[1]$, $C[0]$). 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]∼A[0] in 224-bit)
Ensure: 448-bit result C (C[1]||C[0] in 224-bit)

1: $\varepsilon$0||T ← A[2]+A[3]

2: $\varepsilon$1||C[0] ← A[0]+$\varepsilon$0||T
3: $\varepsilon$2||C[1] ← A[1]+A[3]+$\varepsilon$0||T

4: $\varepsilon$3||C[0] ← C[0]+$\varepsilon$2
5: $\varepsilon$4||C[1] ← C[1]+($\varepsilon$1+$\varepsilon$2+$\varepsilon$3)

6: $\varepsilon$5||C[0] ← C[0]+$\varepsilon$4
7: C[1] ← C[1]+($\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:

$$mask \leftarrow 0 - (carry\ or\ borrow)$$

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: $x3 \leftarrow e3 \cdot v2$                    $\{A = (y1 - x1) \cdot (y2 - x2)\}$
 5: $y3 \leftarrow h3 \cdot u2$                    $\{B = (y1 + x1) \cdot (y2 + x2)\}$
 6: $e3 \leftarrow y3 - x3$                        $\{E = B - A\}$
 7: $h3 \leftarrow y3 + x3$                        $\{H = B + A\}$
 8: $x3 \leftarrow t1 \cdot w2$                    $\{C = t1 \cdot w2\}$
 9: $t1 \leftarrow z1 - x3$                        $\{F = z1 - C\}$
10: $x3 \leftarrow z1 + x3$                        $\{G = z1 + C\}$
11: $z3 \leftarrow t1 \cdot x3$                    $\{Z3 = F \cdot G\}$
12: $y3 \leftarrow x3 \cdot h3$                    $\{Y3 = G \cdot H\}$
13: $x3 \leftarrow e3 \cdot t1$                    $\{X3 = E \cdot F\}$

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

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
		NIST P-256	Curve25519	FourQ	Curve448
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) algorithms[1]. Because SIKE is an alternative candidates, this algorithm should be counted for PQC[2] 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	premaster_secret = ECDHE_KEY ‖ PQ_KEY
seed = "master secret"	seed = "hybrid master secret"
‖ ClientHello.random	‖ ClientHello.random
‖ ServerHello.random	‖ ServerHello.random
master_secret=HMAC(premaster_secret,seed)	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 $\mathbb{Z}_{2m}[x]$ 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, “Four$\mathbb{Q}$ 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.