Time-Efficient Finite Field Microarchitecture Design for Curve448 and Ed448 on Cortex-M4

Mila Anastasova[1], Reza Azarderakhsh[1], Mehran Mozaffari Kermani[2], and Lubjana Beshaj[3]

[1] Computer and Electrical Engineering and Computer Science Department and I-SENSE at Florida Atlantic University, Boca Raton, FL, USA
(manastasova2017, razarderakhsh)@fau.edu
[2] Computer Engineering and Science Department at University of South Florida, Tampa, FL, USA,
mehran2@usf.edu
[3] United States Military Academy West Point, West Point, NY, USA,
lubjana.beshaj@westpoint.edu

Abstract. The elliptic curve family of schemes has the lowest computational latency, memory use, energy consumption, and bandwidth requirements, making it the most preferred public key method for adoption into network protocols. Being suitable for embedded devices and applicable for key exchange and authentication, ECC is assuming a prominent position in the field of IoT cryptography. The attractive properties of the relatively new curve Curve448 contribute to its inclusion in the TLS1.3 protocol and pique the interest of academics and engineers aiming at studying and optimizing the schemes. When addressing low-end IoT devices, however, the literature indicates little work on these curves. In this paper, we present an efficient design for both protocols based on Montgomery curve Curve448 and its birationally equivalent Edwards curve Ed448 used for key agreement and digital signature algorithm, specifically the X448 function and the Ed448 DSA, relying on efficient low-level arithmetic operations targeting the ARM-based Cortex-M4 platform. Our design performs point multiplication, the base of the Elliptic Curve Diffie-Hellman (ECDH), in 3,2KCCs, resulting in more than 48% improvement compared to the best previous work based on Curve448, and performs sign and verify, the main operations of the Edwards-curves Digital Signature Algorithm (EdDSA), in 6,038KCCs and 7,404KCCs, showing a speedup of around 11% compared to the counterparts. We present novel modular multiplication and squaring architectures reaching $\sim$ 25% and $\sim$ 35% faster runtime than the previous best-reported results, respectively, based on Curve448 key exchange counterparts, and $\sim$ 13% and $\sim$ 25% better latency results than the Ed448-based digital signature counterparts targeting Cortex-M4 platform.

Keywords: Elliptic Curve Cryptography, Curve448, Elliptic Curve Diffie-Hellman (ECDH), Edwards-Curve Digital Signature Algorithm (EdDSA), Cortex-M4.

1 Introduction

The use of Public Key Cryptography, often known as PKC, protects the confidentiality of data by ensuring its secure transmission through an unsecured channel, such as the internet, relying on hard mathematical problems to protect the privacy of the information. The NP-hard factorization and (Elliptic Curve) Discrete Logarithm Problems (DLP) are the foundations upon which traditional cryptosystems are constructed. Elliptic Curve Cryptography (ECC), which is entrenched in the complexity of solving ECDLP, delivers high-security levels while demonstrating minimal computational latency and small key sizes in comparison to other classical cryptosystems. As a consequence, ECC-based cryptoschemes are essential for network protocols, as they are frequently used for key agreement and digital signature algorithms. Despite the minimal resource requirements of ECC schemes, public key cryptography remains challenging to implement and deploy on low-end real-time devices which feature scarce memory, limited battery life, and restricted bandwidth.

The continuous advancement of technology leads to its integration into daily life activities, producing the vast universe of the Internet of Things (IoT), which implies an improved standard of living. The widespread usage of real-time embedded systems over the last several decades has created a demand for efficient cryptographic scheme implementation on resource-constrained devices. The specifications of the ARMv7-based Cortex-M4 processor, suitable for cost-conscious and power-constrained development, position the platform among the most widely used embedded devices in the IoT market. This is the reason why the National Institute of Standards and Technology (NIST) [1] selected it as a target platform for evaluating the performance of the cryptographic primitives.

The classical cryptosystems, believed to be robust against today's computers, are, however, shown to be vulnerable to quantum attacks as presented by *Shor* in [2]. The hard mathematical problems underpinning classical schemes could be broken in polynomial time, rather than exponential, when a large-scale quantum computer is developed. Although the availability of such a class of quantum computers cannot be predicted, the need for quantum-robust encryption prompted NIST to initialize a Post-Quantum (PQ) standardization process in 2016. The newly proposed PQ primitives are being evaluated and optimized during the standardization effort. The use of stand-alone post-quantum primitives in network protocols, however, is not in accordance with industry and government standards; hence, hybrid systems based on classical and PQ algorithms are the primary focus of cryptography researchers for transitioning from classical- to PQ-robust environment, thus, the optimal implementation of classical schemes such as ECDH and EdDSA, focus of this work, remains critical for the performance of cryptographic network protocols.

ECC is a critical component of the majority of cryptographic libraries. Yet, in recent years, certain NIST curves have been a subject of further investment and analysis, rising concerns about their security. Due to the resolved backdoor issues associated with existing NIST curves introduced in [3], the recently proposed Montgomery curves Curve25519 and Curve448 and their birationally equivalent

(un)twisted Edwards curves Ed25519 and Ed448 have been widely used and recommended by NIST. Providing 128- and 224-bit security, the curves are suitable for implementing key agreement and digital signature protocols. As a result, they have been included in the TLS1.3 version of the widely used Transport Layer Security protocol from 2018. The interest in the curves leads to several target-specific optimizations, resulting in better performance and energy outcomes. Due to its low calculation latency and reduced resource requirements, several research teams are concentrating on Curve25519 and Ed25519 on different systems, according to the literature. To the best of our knowledge, Curve448 and Ed448 have not yet been explored in such depth, particularly on low-end devices, due to the challenging implementation of long-integer finite field arithmetic on such resource-constrained targets. In this work, we present a new performance record of the key exchange protocol based on Curve448 and the digital signature algorithm based on Ed448 targeting low-end embedded devices based on ARM Cortex-M4 platform.

1.1 Related Work

The implementation of cryptographic primitives on low-end IoT devices is a challenge, especially when designing public key cryptography, due to the enormous resource needs of such schemes. This is why academics and engineers are focusing on the ideal development of asymmetric schemes for embedded devices, where ECC has been the dominant choice when addressing resource-constrained devices due to its efficiency and low bandwidth needs.

Bernstein introduced the new-generation elliptic curve Curve25519 and its birationally equivalent twisted Edwards curve Ed25519 in [4] and [5], respectively, to achieve a high level of security and optimal performance outcomes for the Elliptic Curve Diffie-Hellman (ECDH) key agreement and the Edwards-curves Digital Signature Algorithm (EdDSA). Some of the most recent work on Curve25519 and Ed25519 is presented in [6,7,8] aimed at optimizing the finite field and group computations on high-end platforms. Time-efficient implementation of Curve25519 arithmetic targeting embedded devices is presented in [9,10,11] based on optimal register utilization and careful instruction scheduling. Another research focus is the optimization on hardware presented in [12,13,14]. Extensive study on side-channel protection of the scheme is also present in the literature [15,16,17].

Curve448 along with its birationally equivalent untwisted Edwards curve Ed448, proposed by *Hamburg* in [18], offers higher security level than the discussed Curve25519. The optimizations for Curve448, however, to the best of our knowledge, have not been as exhaustive, specifically when targeting low-ended devices with scarce resources due to the higher security level and thus more computationally intensive arithmetic operations.

Recent enhancements to the 224-bit secure ECDH/EdDSA over Curve448 targeting Haswell and the Skylake microarchitectures are presented in [19] where *Oliveira et al.* present an optimal fixed-point multiplication strategy based on precomputation of constant values derived from the fixed point and its multiples.

Later, in [20], *Seo* presents an optimized implementation of Curve448 arithmetic targeting low-end 8-bit AVR and 16-bit MSP processors, where the main contribution consists of the adoption of two- and three-level subtractive Karatsuba method for the execution of the multi-precision multiplication and squaring subroutines. *Faz-Hernandez et al.* present speed optimizations of the Curve448 in [6] targeting the Intel AVX2 vector instruction set, reaching 10-30% of performance improvements for key agreement and digital signature cryptographic schemes. Finally *Seo et al.* present the first implementation of Curve448 ECDH targeting the low-end embedded platform ARM Cortex-M4 [21] and *Anastasova et al.* show optimal target-specific implementation of EdDSA algorithm based on Ed448 [22].

The Elliptic Curve Cryptography has a pyramid-like layered structure, where the computation of high-layer group operations is based on low-layer finite field arithmetic. The pyramidal structure enables high-level improvements aimed at breaking speed records for group operations. The low-level arithmetic computations, the topic of this study, result in an overall acceleration of the ECC primitives, with the platform characteristics dictating the optimization tactics used to create the architecture. The primary challenge when implementing Curve448 and Ed448 field arithmetic is that the operands frequently exceed the available CPU resources, particularly on low-end platforms. Due to the unavoidable necessity for safe and efficient cryptographic protocols, the ECC architectural design undergoes continual research and optimization efforts on both high- and low-level enhancements.

The high-level improvements include representing the curve elements in projective coordinates rather than affine coordinates to minimize the number of costly arithmetic operations required for scalar-point multiplication. Montgomery ladder [23], significantly lowers the latency of ECC-based protocols by combining Montgomery's doubling formulae with Montgomery's differential-addition formulas and enables the use of Y-only coordinate calculations.

Several efforts to optimize low-level field arithmetic operations are documented in the literature, with optimal execution strategies resulting in record performance of the cryptographic protocols. Due to the complexity of the long-integer operations' execution flow, there is no implementation option that can be considered ideal. Indeed, different platforms offer a diversity of capabilities, each of which facilitates a specific set of instructions, hence favors the deployment of a specific multi-precision approach. The adaptation of Product Scanning (PS) or Operand Scanning (OS) methodologies, with a focus on long-integer optimal solutions, for low-end devices, has been rigorously investigated and tested. The literature also proposes combination of multi-precision strategies to provide further performance improvements.

In [24] *Hutter et al.* present the first implementation of Operand Caching (OC) multiplication technique which outperforms the pervious best hybrid implementation architectures. Based on the OC strategy, *Seo et al.* present in [25] and [26] optimized variant of the multi-precision arithmetic where the execution flow of the inner loop is re-arranged to optimally re-use common operands be-

tween previous and new partial products reporting significant speedup results. Optimization of long-integer multiplication and squaring techniques based on the Cortex-M4 platform is presented by *Seo et al.* in [27], where the authors propose Refined-OC multiplication technique based on increased number of cached limbs in the register bank of the processor. The article offers optimal field operations independently of the post-quantum nature of the target protocol since classical ECC techniques, as well as the post-quantum Supersingular Isogeny Key Encapsulation (SIKE) mechanism, are defined over large finite fields and so operate on big integers, thus, implement the same lowest layer arithmetic for the performance of the high layer group operations. Further work on the finite field operations of the elliptic curve-based PQ protocol is presented in [28,27,29,30] targeting specifically the ARM Cortex-M4 platform.

Literature indicates that little effort has been spent optimizing 448-bit integer finite field arithmetic for the entry-level ARM-based Cortex-M4 architecture. Multiple research teams are implementing and enhancing the lowest layer of ECC primitives. Using a novel architecture for finite field arithmetic, we extend this study by demonstrating a new performance benchmark for Curve448 and Ed448. [21] and [22] show the most relevant research based on Curve448 that focuses on low-end RISC devices. In this paper, we extend this area of research by introducing a novel method for long-integer operations and compare its performance to that of prior work. In addition, the long-integer implementation subroutines are an excellent fit for the PQ protocol SIKE.

1.2 Contributions

In this work, we demonstrate novel implementation techniques for accelerating the execution of the Curve448- and Ed448-based key derivation and digital signature protocols. Our contributions include the following:

1. We present a novel design for the underlying finite field operations multi-precision multiplication and squaring targeting the ARM Cortex-M4 platform. We observe a speedup of 25% and 35%, respectively for modular multiplication and squaring functions when compared to the Curve448-based key exchange protocol counterparts in [21] and 13% and 24% when compared to the previously best-reported results for the Ed448-based digital signature algorithm presented in [22].
2. We present the first handcrafted assembly implementation of multi-precision squaring procedure with the goal of improving Curve448 and Ed448 for the ARMv7-M architecture. Both multi-precision multiplication and squaring are implemented using a novel architecture in which we combine multiplication techniques. We allocate a fixed number of registers for storing words from A and lower the number of registers for storing operand B's limbs, where we compute current and successive column-wise partial results. Thus, we present the first multi-precision multiplication architecture, combining product- and operand-scanning techniques in the inner multiplication loop execution flow.

Algorithm 1 Montgomery ladder

Input: $P = (X_P : Z_P)$, $k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_{l-1} = 1$
Output: $R = k \cdot P$
1: $R \leftarrow (X_R, Z_R) = (1, 0)$
2: $Q \leftarrow (X_Q, Z_Q) = (X_P, 1)$
3: **for** $(i = 447; i >= 0; i - -)$ **do**
4: **if** $k_i = 0$ **then**
5: $(R, Q) = ladderstep(X_P, R, Q)$
6: **else**
7: $(Q, R) = ladderstep(X_P, Q, R)$
8: **end if**
9: **end for**
10: **return** $x_R = X_R / Z_R$

3. We present a speedup of around 48% and 11% for the X448 and Ed448 DSA protocols, compared to the best previously reported results in [21] and [22], respectively on the target platform when running on STM32F407VG discovery board @24MHz to avoid zero wait state and to disregard memory controller stalls.
4. We evaluate and analyze the proposed design's performance by conducting benchmarking experiments at 24MHz, which presents the exact number of clock cycles on the target platform regardless of the microcontroller's specifications, and 168MHz, which boosts the performance of the STM32F407VG board to obtain real-world values.

The rest of the paper is organized as follows. In Section 2 we present an overview of the mathematical concepts underlying X448 and Ed448 DSA protocols and summarize the main features of the target architecture. Section 3 presents the proposed finite field arithmetic architecture and overviews the performance results of the newly implemented functions. In Section 4 we perform latency evaluation of the entire protocols after integrating our new function implementations. Finally, we conclude our work in Section 5.

2 Preliminaries

This section provides an overview of the mathematical ideas underpinning the Curve448 and Ed448 key exchange and digital signature protocols. We discuss both protocols and illustrate their execution flow, as well as the primary properties of the target platform.

2.1 ECC Mathematical Background

A Montgomery Elliptic Curve Curve448 over a finite field $\mathbb{F}_p$ is defined by the solutions of the equation:

$$E_M/\mathbb{F}_p : v^2 \equiv u^3 + Au^2 + u$$

Algorithm 2 Montgomery ladder step

Input: x_P, $R = (X_R, Z_R)$, $Q = (X_Q, Z_Q)$
Output: $P_{PD} = 2 \cdot R$, $P_{PA} = R + Q$
1: $X_{PD} = (X_R - Z_R)^2 \cdot (X_R + Z_R)^2$
2: $Z_{PD} = 4X_R Z_R \cdot (X_R^2 + 39081 X_R Z_R + Z_R^2)$
3: $X_{PA} = 4(X_R X_Q - Z_R Z_Q)^2$
4: $Z_{PA} = 4x_p(X_R Z_Q - Z_R X_Q)^2$
5: **return** $P_{PD} = (X_{PD}, Z_{PD})$, $P_{PA} = (X_{PA}, Z_{PA})$

where the value of A is defined as 156326 and $p = 2^{448} - 2^{224} - 1$. Montgomery curves have their birationally analogue Edwards curves, where Curve448 can be represented by the solutions to the equation:

$$E_{Ed}/\mathbb{F}_p : ax^2 + y^2 = 1 + dx^2y^2$$

with $d = -39081$ and $a = 1$ since the value of the prime number is congruent to $3 mod 4$ and thus the curve is untwisted Edwards curve called Ed448. The elements of Curve448 are represented by two coordinated $(u, v) \in \mathbb{F}_p \times \mathbb{F}_p$. The birational map to project a point from Montgomery to Edwards curve is as follow:

$$(u, v) = ((y - 1)/(y + 1), sqrt(156324) * u/x)$$

where to map the point back to Montgomery curve the next formula is applied:

$$(x, y) = (sqrt(156324) * u/v, (1 + u)/(1 - u))$$

Elliptic Curve Cryptography's nature is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). Executing scalar-point multiplications with the point $P = [k] \cdot Q$ results in the addition of point Q with itself k times, where the value of k is difficult to resolve given P and Q.

Point multiplication requires several point additions and point doublings, where various techniques can be applied to obtain the resulting coordinates such as Double-And-Add (and its constant time variants) or Montgomery ladder Algorithm 1, where the latter requires p steps of combined point addition (PA) and point doubling (PD) function (referred to as Montgomery ladder step) offering better performance results.

To further increase the speed of the scalar-point multiplication, the point is transformed from affine (x, y) to projective representation (X, Y, Z) with $x, y = (X \cdot Z^{-1}, Y \cdot Z^{-1})$, which relaxes group operations by reducing the number of costly operations such as modular inversions. Algorithm 2 illustrates the Montgomery ladder step, a more thorough illustration of the execution steps for point addition and point doubling unified formula.

The usage of Montgomery ladder enables the computing of time-efficient $X-$only formulae in which the Y coordinate is not required for point multiplication computations and is restored once the method is done, which results

Algorithm 3 X448 algorithm. G represents the value of the base point

Alice		**Bob**
$sk_A \in_R \mathbb{Z}/\mathbb{F}_p$		$sk_B \in_R \mathbb{Z}/\mathbb{F}_p$
$pk_A = [sk_A] \cdot G$		$pk_B = [sk_B] \cdot G$
	exchange $pk_A \longleftrightarrow pk_B$	
$ss_A = [sk_A] \cdot pk_B$		$ss_B = [sk_B] \cdot pk_A$
$ss_A = [sk_A] \cdot sk_B \cdot G$	$ss_A = ss_B$	$ss_B = [sk_B] \cdot sk_A \cdot G$

in further speed optimizations and it represents the method deployed in most implementations, including this work.

2.2 X448

Elliptic Curve Diffie-Hellman (ECDH) protocol implementation enables communication parties to agree on a shared secret that is later utilized in low-cost symmetric encryption schemes. To execute ECDH over a finite field using Curve448 requires both computing parties to generate secret key values represented by a long-integer value. Later on, each must apply the scalar-point multiplication function X448 depending on the scalar value of their secret key and a public base point G, for instance using the Montgomery ladder Algorithm 1. The newly computed points, representing the public keys of each party, are exchanged and another point multiplication is computed, applying their own secret key scalar value and the received point public key value. Algorithm 3 provides a representation of the ECDH algorithm in detail.

Following the execution of the two-point multiplications, as presented in Algorithm 3, both parties can ensure the privacy of their communication via efficient symmetric encryption scheme. The symmetric key is being derived through the equivalent values of the shared secrets and is then being used to encrypt data based on symmetric algorithm such as AES.

2.3 Ed448

The digital signature algorithm is mainly used to verify that the communication was sent by the intended recipient. The Edwards-Curve Digital Signature Algorithm (EdDSA) is defined in three phases - Key Generation, Sign and Verify. Ed448 DSA has a thorough explanation of these procedures, which may be found in Algorithm 4. The key generation uses a seed value to produce a secret key and its respective public key that are generated using a eXtendable Output Function (XOF) SHAKE256 (denoted with capital letter H in Algorithm 4) and scalar-point multiplication function. After running the signing function, it returns a

Algorithm 4 Ed448 algorithm [31]. H denotes $SHAKE256$. L represents the order of Ed448 curve. G represents the value of the base point

Key Generation

Input: $seed$
Output: $(p, s), pk_A$
1. $sk_A \in_R^{seed} \mathbb{Z}/\mathbb{F}_p$
2. $(p, s) \leftarrow H(sk_A)$
3. $pk_A \leftarrow encode([s] \cdot G)$
Return $(p, s), pk_A$

Sign

Input: $pk_A, (p, s), M$
Output: $sign \equiv R||S$
1. $r \leftarrow (H(p||M))(modL)$
2. $R \leftarrow encode([r] \cdot G)$
3. $k \leftarrow (H(R||pk_A||M))(modL)$
4. $S \leftarrow encode((r + k * s)(modL))$
Return $R||S$

Verify

Input: $pk_A, M, R||S$
Output: $true/fasle$
1. $k \leftarrow H(R||pk_A||M)(modL)$
2. $A \leftarrow decode(pk_A)$
Return $[S] \cdot G == R + [k] \cdot A$

signature $R||S$ generated based on the secret key and the message value. Finally, the verification is executed based on the public key and the message value and returns success upon the correctness of the equation $[S] \cdot G == R + [k] \cdot A$.

As noted, the scalar multiplication subroutine is forming the basis of both - elliptic curve based key agreement and digital signature algorithms, thus, its optimization is the main focus of this work. A new design and a performance record of the multi-precision multiplication and squaring, the base operations of point multiplication, are described later in the paper and the timing results of both cryptographic primitives are reported based on the proposed finite field arithmetic design.

2.4 Target Architecture

The ARM Cortex-M4 processor's Reduced Instruction Set Computer (RISC) architecture delivers a set of basic yet powerful instructions that are devoid of structural hazards and data dependence delays. This is why it is in such great demand in the realm of IoT and real-time systems. Furthermore, NIST recommended ARMv7-M Cortex-M4-based STM32F407VG discovery board microcon-

Table 1. ARMv7-M ISA [32] for memory access and MAC instructions

Instruction	Functionality	Latency (CC)
(V)LDR/ (V)STR	$R_n \leftarrow$ memory memory $\leftarrow R_n$ $S_n \leftarrow$ memory memory $\leftarrow S_n$	2
VMOV	$R_n \leftarrow S_m$ $S_m \leftarrow R_n$	1
UMULL	Rd_1, $Rd_2 \leftarrow R_n \times R_m$	1
UMAAL	Rd_1, $Rd_2 \leftarrow R_n \times R_m + Rd_1 + Rd_2$	1

troller for low-end device performance assessment, featuring 192KB of RAM and 1MB of flash memory, thus, it represents the board chosen by NIST for performance evaluation of the cryptographic algorithms and is, therefore, the target platform of this article.

The limited register set of just 16 32-bit General-Purpose Registers (GPRs) `R0-R15` where two of them are reserved for the Stack Pointer SP and the Program Counter PC and are not accessible by the programmer, converts the implementation of multi-precision arithmetic operations into a challenging task. The ARMv7-M architecture offers another 32 32-bit Floating-Point Registers (FPRs) S0-S31. The transition of register values between the two register banks is ensured to be instant via the powerful VMOV instruction. The single clock cycle instruction latency, specific for the ARMv7-M 3-stage pipeline, has as only exception the memory access `LDR/STR` instructions where if not properly scheduled they can induce an additional clock cycle before another instruction can be processed. The nature of the long-integer arithmetic does not always allow to schedule the instructions, thus, to avoid stalling the pipeline. To maximize the performance of a hand-crafted assembly code, a thorough structure of the instruction flow and an in-depth examination of the Instruction Set Architecture (ISA) Table 1 are required. The precise order of the instruction flow is a combinatoric problem which requires careful analysis and deployment to provide the most optimal execution path.

The ARMv7-M ISA supports powerful multiplication instructions, referred to as Multiply ACcumulate (MAC), ensuring the execution of 32×32-bit multiplication, resulting in a 64-bit long value. The simple long multiply `UMULL` instruction offers an accumulative variant `UMAAL` executing another two 32-bit accumulative additions. The single clock cycle latency of the MAC instruction considerably improves the performance of long integer multi-precision multiplication and squaring subroutines when utilized correctly as presented in this work.

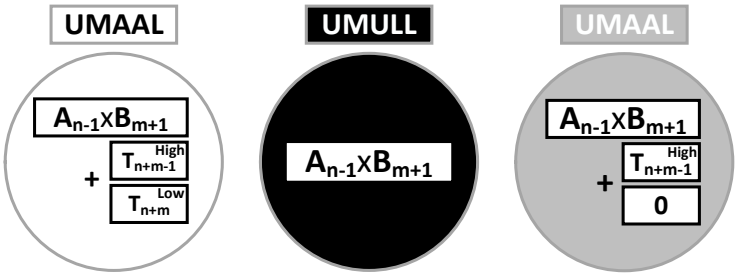

Fig. 1. Instruction set notation in dot format for the rhombus representation of the arithmetic operations multiplication and squaring.

3 Proposed Design for Field Arithmetic

3.1 Notation

The proposed architecture for multi-precision multiplication and squaring may be presented in a variety of ways, the most visually appealing of which is through the use of a rhombus representation. The implementation of finite field procedures needs distinct instruction sequences, which we represent with dots on the rhombus figures.

We utilize a different pattern of dots to a different color of dots, as shown in Figure 1, to denote various MAC instructions. The white color dot indicates the execution of Unsigned Multiply with double Accumulate Long `UMAAL` where the multiplication of two 32-bit registers is performed, resulting in a 64-bit value accumulated with the content of the destination registers as two 32-bit numbers. The black dot represents an Unsigned MULtiply Long `UMUUL` with the destination registers containing the result's low and high 32-bits. Finally, the implementation of the multi-precision squaring routine requires the accumulation of a single 32-bit value to the 64-bit multiplication result, thus, we use the `UMAAL` and zero out one of the accumulated values. We denote the use of this instruction by a gray dot on the rhombus schemes.

For our implementation, we refer to the previous multi-precision strategies, such as Product-Scanning (PS), Operand-Scanning (OS) and Refined Operand-Caching techniques (R-OC). OS method is predicated on the concept of reusing a single limb from one operand while employing the whole set of limbs from the second operand. In particular, each calculation step should accumulate the previously stored partial result limb with a single partial result value. The computation of a partial result limb in each iteration can be represented as $T_i = T_i + A_k B_{i-k}$ with k being the iteration count. Finally, the product limb $R_i = T_i$ when the iteration count equals the result index being computed. Therefore, the computation of R_4 requires the execution of 5 iterations, where in each iteration one partial result is being computed and accumulated to the current partial result value.

The PS method is based on computing the entire set of accumulative partial multiplication results at once, allowing to obtain the final result, and, thus, not re-load any partial product values. Therefore, for the computation of the result limb R_i all partial multiplications would be computed and accumulated, i.e. $R_i = \sum_{k=0}^{i} A_k B_{i-k}$. Both techniques are the base of the modern multi-precision multiplication strategies and are combined, depending on the characteristics of the target device, to offer optimal performance results.

One such combination among PS and OS is the so called Operand Caching (and its variants), where the multiplication implementation is split into different sections, referred as rows, and each row consists of straightforward implementation of the product scanning multiplication technique. The size of the row represents the number of consecutive accumulated partial results computed in each iteration. Each row, thus, produces *row-size* number of accumulated partial results. The rows among them employ the OS approach, wherein the previously computed partial result is accumulated with the newly computed partial value at each iteration. The multi-precision multiplication design is also referred to as a combination of two loops - inner and outer multiplication loop. The OC deploys PS in the inner loop, i.e. the computations inside the scope of the rows. The outer loop in OC implements OS method which is applied among the different rows. The size of the row is one of the factors to determine the variant of the OC multiplication method, where the latest and most efficient variant of the OC targeting the Cortex-M4 platform is the Refined-OC (R-OC) method with row size equal to four.

The use of one or another multi-precision multiplication approach to enhance performance is entirely dependent on the platform being targeted. Large processors, for instance, have a large register set bank, so they can store more operand limbs and favor the PS technique. In addition, modern processors feature instant memory access instructions, therefore, multiple operand re-loading is inexpensive. However, the usage of low-end devices, such as the intended ARM Cortex-M4 chip, makes long-integer computations difficult due to register bank constraints and expensive memory accesses. We disclose the obtained speedup for the provided arithmetic and the performance record after incorporating our design into the ECDH and EdDSA cryptographic algorithms in this paper.

3.2 Multi-precision multiplication

The design and implementation of the multi-precision multiplication subroutines are extremely important for the efficiency of the cryptographic protocols when based on long-integer values. The nature of the multi-precision multiplication places it into the most frequently invoked routines in Elliptic Curve and Isogeny-based protocols. Additionally, due to the high computational cost of these procedures, optimizing them results in an overall speedup of the protocols. This is why several academics have concentrated their efforts on optimizing it for various platforms.

In this paper, we present a novel multi-precision multiplication approach, demonstrating a speed record on the target platform.

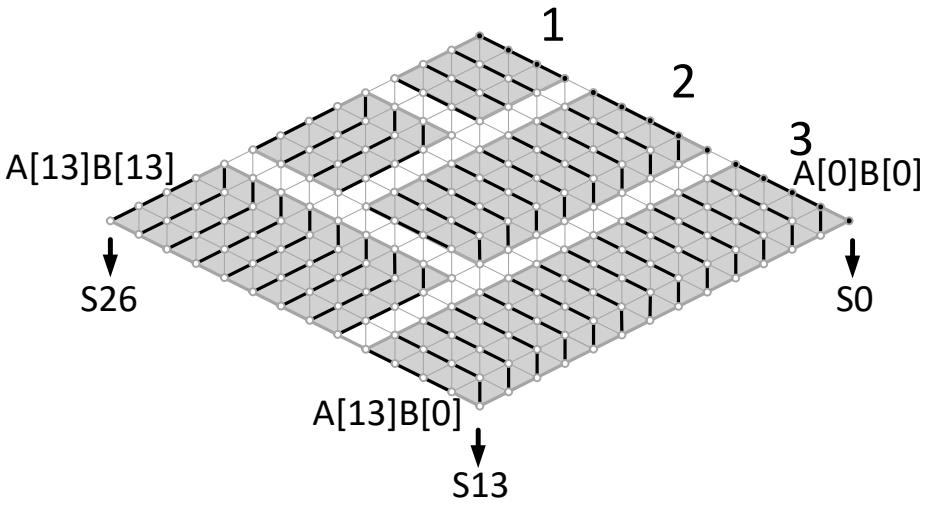

Fig. 2. Proposed architecture for 448-bit multi-precision multiplication. Black lines denote inner loop execution flow.

We illustrate our architecture in detail in Figure 2, where the diagonal lines in the rhombus indicate the operands `A` and `B` and the dots represent the word-level operand partial operations. Numerous studies in the literature focus on the combinatorics of this topic with the goal of optimizing the performance outcomes of this function. The primary multiplication approaches are frequently used in high-level implementation designs that include product- or operand-scanning. However, owing to resource limits, a single application of one of the multiplication algorithms is not practical. Additionally, both have some significant downsides. To be more precise, the former requires numerous accesses to the value of the (partial) product in order to compute the result, whereas the latter requires continual reloading of the operand words in the accessible register set. Which approach is the most optimal is entirely dependent on the technical requirements of the target platform.

To optimize multiplication performance, the authors in [33,24,25,27] provide designs that combine the two major approaches to take use of the benefits of each one, namely the Hybrid, Operand Caching, Consecutive Operand Caching, and Refined-Operand Caching (R-OC) multi-precision multiplication. The methods already proposed in the literature apply one of the major techniques to the outer multiplication loop and one to the inner loop. The primary goal of routine optimization for low-end devices is to minimize memory accesses, since it can cause additional stalls when no cache memory is available, which is frequently the case with embedded low-cost devices. The R-OC, the most efficient multiplication in the literature, method's concept is to load operands into the register set and reuse the operands' values. To do so, the authors in [27] introduce a method for storing four words of both operands in the instant memory units, thus, increase

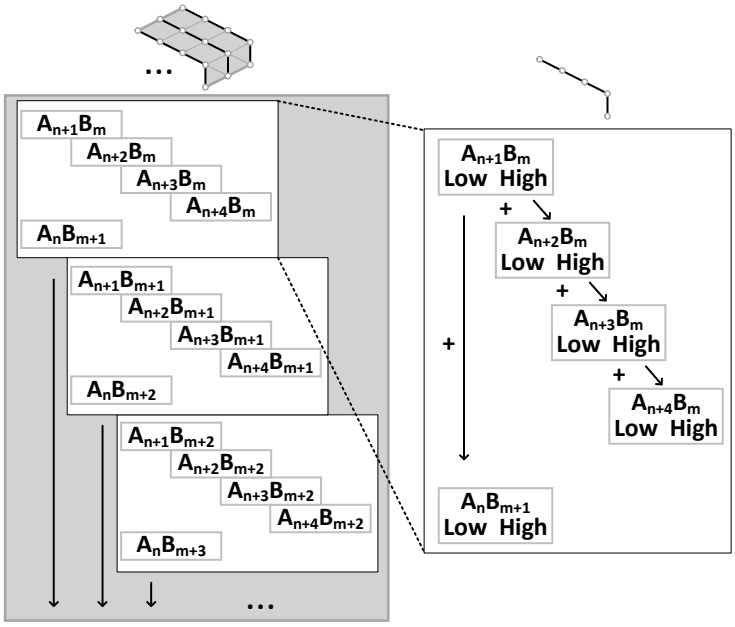

Fig. 3. Proposed design, register utilization and carry propagation for the multi-precision multiplication outer (left) and inner (right) loop execution flow.

the size of the inner loop (i.e., row size) to four 32×32-bit multiplications. This technique optimizes the memory accesses by reusing the four loaded limbs from the second operand, when the growing operand index is switched (i.e., at the middle of the inner loop).

In this work, we present a novel technique for multi-precision multiplication, with an emphasis on increasing row (inner loop) size and hence decreasing memory accesses for partial value accumulation. To accomplish this aim of creating rows of size five, we reserve five registers for the value of operand `A`, and three registers for operand `B` (further detail of the register utilization is provided in Figure 3). In our approach, two partial results are computed for the current column of calculation and one multiplication is performed for the subsequent three columns. This reduces register requirement for the storage of the second operand. Therefore, we maximize the available registers and increase the row size to five 32-bit multiplications per iteration, as there are sufficient free registers for the partial column-wise results. A close view of the inner loop operations is shown in Figure 3, where it is presented that each step computes a single partial column value, stores it into memory and keeps another four columns partial values in the register set.

The row's length implies that further six registers should be reserved for the 32-bit partial results. In the previous best performance implementation design (i.e., R-OC) in [27], the authors use a single register to hold the current column's

Algorithm 5 Proposed design (pseudocode), register utilization and carry propagation for the multi-precision multiplication inner loop execution flow.

```
VMOV R0, S12                    // R12
UMAAL R0, R10, R2, R6           // a6b6
UMAAL R11, R10, R3, R6          // a7b6
UMAAL R12, R10, R4, R6          // a8b6
UMAAL R14, R10, R5, R6          // a9b6
LDR R7, [R8, #4*7]              // b7
UMAAL R0, R9, R1, R7            // a5b7
VMOV S12, R0                    // R12
```

lower 32-bit accumulative result and another four registers to store four separate upper 32-bit partial values.

In contrast to this strategy, in our work, we utilize a single register for the upper 32-bit value, which is continually propagated to the subsequent column, and another five registers for the lower 32-bit values. Figure 2 illustrates graphically our architecture design with each inner loop iteration denoted by a black line. A more detailed view is shown in Figure 3 and a pseudocode is presented in Algorithm 5, where one register R_{n+1+m} stores the low results of $A_{n+1}B_m$ accumulated to A_nB_{m+1}, R_{n+2+m} stores low $A_{n+2}B_m$, R_{n+3+m} stores low $A_{n+3}B_m$, R_{n+4+m} stores low $A_{n+4}B_m$ and R_{n+5+m} stores high $A_{n+4}B_m$ result, which has accumulated all previously created upper 32 bits (carry propagation).

To our knowledge, this new combination of product scanning and operand scanning approaches in the inner loop of multi-precision multiplication is introduced for the first time in the literature. As described in Figure 3, this hybrid-inner loop design enables a reduction in the number of words allotted for operand B to only two. We reserve three to minimize the cost of reloading operands in the midst of the multiplication; consequently, we require just two more reloads per row above R-OC. By utilizing the register set optimally and following the new instruction flow, we optimize the R-OC approach by raising the row widths, which lowers access to partial results for accumulation.

3.3 Multi-precision Squaring

Due to the high invocation ratio of the multi-precision squaring routine, its performance optimization benefits the overall execution time and resource requirements of Curve448- and Ed448-based key exchange and digital signature algorithm protocols. In this work, we propose and implement the first multi-precision squaring procedure in hand-crafted assembly target-specific ARMv7 architecture code for the finite field of length 448 bits.

Due to the fact that the bottom portion of the rhombus representation mirrors the top part, the software design of the squaring benefits from the duplication of some of the partial result values, and so can produce higher performance results than multi-precision multiplication. Since the result of A_nA_m equals the

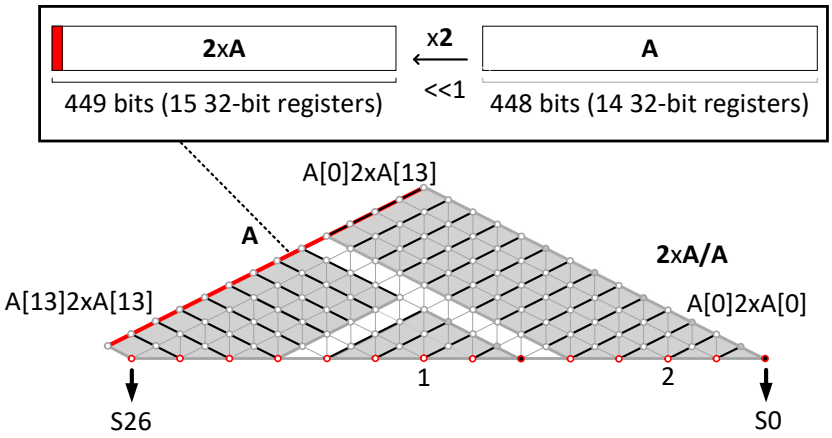

Fig. 4. Proposed architecture for 448-bit multi-precision multiplication with 14th index word for the doubled operand value (red line).

result of $\mathtt{A_m A_n}$ the accumulation of bottom and top rhombus sides may be implemented by doubling one of both values. The computation for the rhombus diagonal consists of operands with coinciding indexes (e.g. $\mathtt{A_k A_k}$), as seen in Figure 4 where they are marked with a red line. Their partial outcomes occur just once and are hence not duplicated. Additionally, because only one operand is involved in the operation, loading the operand words into the register set is eased. Numerous teams have worked in the literature to exploit these aspects of the subroutine and optimize the output for low-end target systems. There has been no software implementation of multi-precision squaring for the Curve448 and Ed448 protocols; hence, this work covers this gap by offering the first and most optimal design of 448-bit multi-precision squaring on the Cortex-M4 platform in comparison to other similar research.

Scott et al. provide one of the first attempts for multi-precision squaring function aiming low-end devices in [34]. The authors propose a carry catcher approach with an additional number of registers dedicated to storing and accumulating the generated carry. Later, *Lee et al.* propose lazy doubling method in [35] the authors, where each computed column is doubled and then accumulated to the non-doubled values. While this approach is the closest similar to our novel design, it does not produce ideal results due to the numerous result doublings and accumulations required.

In this work, we propose a new design for the implementation of finite field long-integer squaring. Our implementation's inner loop is based on the operand scanning mechanism. The rationale for picking this method is the enormous amount of free registers available due to the routine's single operand nature. Our primary goal is to raise the row size. For our design, we used the double-operand approach presented in [27]. Additionally, we use the improved execution flow described in [29], which decreases the row number.

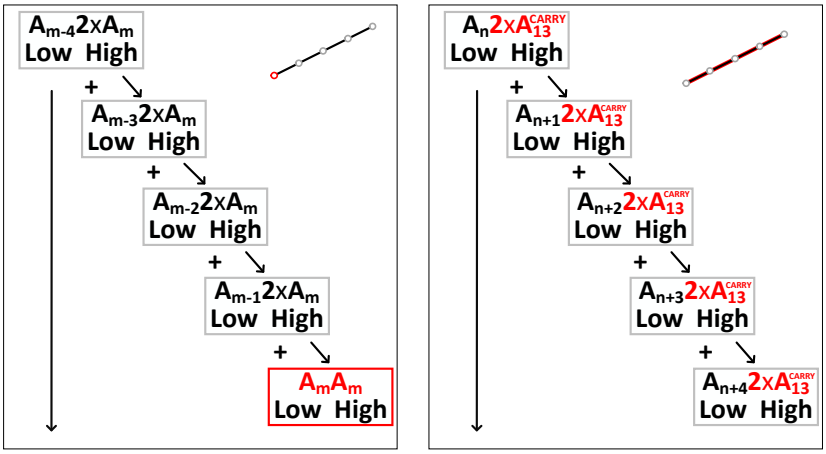

Fig. 5. Proposed design, register utilization and carry propagation for the multi-precision squaring with coinciding indexes of the operand (left) and with the carry word produced after doubling the operand (right).

We should note that due to the exact fit of the prime number consisting of 448-bits into 14 32-bit value registers, the doubling the operand `A` may produce a carry. The carry bit specifies whether the doubled value of `A` is 448 or 449 bits. Thus, we simulate an additional word with index 14 of the duplicated `A` to obtain accurate results. We draw a red line through the word with index 14 of the doubled `A` in Figure 4 and regard it as a regular limb of `2xA`.

A thorough depiction of the implemented design is shown in Figure 5 which illustrates the execution flow of one inner step iteration. On the left is a representation of the 32-bit partial multiplication of four doubled operand words with the original `A`. As with multiplication, the carry is kept in a single register and is propagated across subsequent short multiplications through the MAC UMAAL instruction. The right side of Figure 5 depicts the execution flow when the `2xA` is utilized. As it could be noticed, there is not much difference with the rest of the iterations, except that the value of the carry A_{13} is dynamically computed during the execution of the program.

3.4 Side-channel implications

Side-channel analysis (SCA) attack uses data leakage based on timing, power consumption, or electromagnetism information to recover secret information about the communication parties.

In this work, we propose new constant-time implementation of the multi-precision multiplication and squaring architectures for our Curve448-based key agreement and digital signature algorithms. Our subroutines' designs do not contain conditional execution flow, thus, are robust against timing attacks.

A question that can emerge is based on the fact that multiplication and squaring deploy different architectures and, therefore, show different latencies. This, however, should not present an issues in terms of information leakage since we deploy Montgomery ladder point multiplication design, where in each Montgomery ladder step one point doubling and one point addition are executed, independently of the secret key bit values. The sequence of multi-precision multiplication/squaring is known and is the same in each Montgomery ladder step. We should note again that this work is focused on the multi-precision modular operations, not on the group operations. Thus, our proposed design is not directly dependent on the users' secret information, but rather forms part of the multiple executions of the Montgomery ladder step.

3.5 Implementation Results

The next section compares the implemented multi-precision multiplication and squaring methods to their literature counterparts.

We present the execution time of low-level finite field arithmetic operations and the speedup obtained by integrating them into group operations. We base our experiment results on the STM32F407VG microcontroller running at 24MHz in order to remove memory controller stalls and deliver more exact findings that remain relatively similar on any ARMv7-M-based board.

We compare our work with the best-known counterparts in the literature targeting the same platform for Curve448- and Ed448-based algorithms and we present the latency results in number of clock cycles in Table 2. In this work, we achieve 25% of improvement compared to [21] for finite field multiplication when integrating our new multi-precision designs. Additionally we observe another 35% of speedup after introducing the first 448-bit modular squaring subroutine. The low-level arithmetic optimization leads to a 31% speedup for the inversion routine.

The group operations point addition and point doubling show 49% of performance improvement for the execution of the point doubling and addition. Our design is based on the execution of Montgomery ladder step where the doubling and addition are performed by the subroutine. After integrating the low-level finite field arithmetic along with the Montgomery ladder point multiplication strategy we observe more than 48% improvement for the execution of point multiplication.

The work on Ed448 we compare with the recently published work [22] where we achieve 13.1% and 24.5% latency speedup for the execution of the multiplication and squaring routine. Sequentially, we observe around 24% better results for the execution of the inversion and another 13.5% for the execution of the Montgomery ladder step. Finally, we observe 12% better performance for the Montgomery ladder based scalar multiplication routine.

In the following section, we perform a more exhaustive report of the overall elliptic curve based key agreement and digital signature algorithms latencies when integrating our new low-level architecture designs.

Table 2. Finite field operations for Curve448/Ed448 targeting ARMv7-M

Ref.	Arithmetic Performance Evaluation					
	Fp			Group		
	Mul	Sqr	Inv	Add	Double	Multiply
Curve448						
Seo et al.[1]	821	821	363,485	6,566	6,567	6,218,135
This work	**613**	**532**	**247,707**	**6,640(total)**		**3,220,682**
	25.33%	35.20%	31.85%	49.44%		48.21%
Ed448						
Anastasova et al.[2]	705	705	325,997	8,465(total)		3,703,755
This work	**613**	**532**	**247,934**	**7,323(total)**		**3,259,379**
	13.05%	24.54%	23.95%	13.49%		12.00%

Refer to:[1] [21], [2] [22]

4 Performance Evaluation

The next section analyzes the acquired findings in terms of performance. We report on the latency of our designs when they are executed on the STM32F407VG discovery board, which features a Cortex-M4 CPU. We run our results at 24MHz to assure a zero-wait condition and hence eliminate memory control unit stalls. Additionally, we report our results when the Curve448 ECDH key exchange and Ed448 DSA protocols are run at 168MHz in order to simulate a real-world scenario on the given microcontroller. Note that the high frequency measurement is extremely reliant on the target platform and varies between devices based on the memory control unit's clock speed.

We base our results on the version `gcc-arm-none-eabi-10.3-2021.07` cross-compiler setting the `-O3` optimization flag for optimized performance results. We compare our work with Curve25519- and Curve448-based implementations where we note that the results presented for Curve25519 are significantly better than our results due to the size of the prime number and thus the minimal length of the operands.

We provide the performance findings in Table 3 in terms of clock cycles $\times 10^3$. We notice that we obtain $\sim 3.6\times$ slower results for Curve448 in comparison to the Curve25519 results reported in [10] targeting the same platform. However, we should note that the Curve448 offers a much higher security level, in particular, 224-bit compared to 128-bit of Curve25519. We present more than $22\times$ better results for executing scalar-point multiplication X448 function than the work presented in [10]. However, we find that the limited resources available on their low-end target systems necessitate more extensive outcome improvements. Thus, a portion of the reason for the enormous latency discrepancy is due to the target restrictions experienced by the writers on such low-end architecture device.

We compare the best performance results on the target platform, as provided by *Seo et al.* in [21] and *Anastasova et al.* in [22], by assessing the X448

Table 3. Curve 25519 and Curve448 key exchange and digital signature computation latency performance on IoT platforms

Work	Platform	Freq. [MHz]	X448	Ed448 KeyGen	Ed448 Sign	Ed448 Verify
Curve25519[1]	Cortex-M4	84	894	390	544	1,331
Curve448[2]	AVR	32	103,229	-	-	-
	MSP	25	73,478	-	-	-
Curve448[3]	Cortex-M4	24	6,218	-	-	-
		168	6,286	-	-	-
Ed448[4]	Cortex-M4	24	-	4,069	6,571	8,452
		168	-	4,195	6,699	8,659
This work	Cortex-M4	24	**3,221**	**3,536**	**6,038**	**7,404**
		168	**3,975**	**4,282**	**6,787**	**8,854**

Refer to:[1] [10],[2] [20], [3] [21], [4] [22]

point multiplication function and the Ed448 DSA key generation, sign, and verify functions. We mark around 48.2% and 36.8% of speedup when comparing our optimized X448 design running at 24MHz and 168MHz, respectively. The gains realized are a result of the novel arithmetic architecture introduced in this study. Thus, we report the execution of point multiplication in $3,221 \times 10^3$CCs. We also report a speedup of 13.1%, 8.1%, and 12.4% compared to the most recent literature equivalents while analyzing Ed448 EdDSA on the STM32F407 discovery board. We note that our implementation design shows a 1.4% of latency increase when running the digital signature procedures at the maximum platform frequency of 168MHz. This is due to the floating-point register set being utilized as a storage unit rather than memory. This change, however, is negligible and is mostly due to the board's increased speed, a scenario that may not exist with other microcontrollers with different features.

5 Conclusion

In this work, we present a novel design for time-efficient finite field arithmetic over Curve448 and its birationally equivalent Ed448, where the pyramid-like structure of the protocols, results in an overall speedup of the key derivation and digital signature protocols based on the Montgomery and Edwards representation of Curve448. We describe an optimum multi-precision multiplication architecture and the first hybrid implementation of operand and product scanning techniques in the multiplication routine's inner loop. Additionally, we provide the first multi-precision squaring technique for 448-bit finite field arithmetic, where the carry of the operand doubling is represented as a new word and utilized to compute the right final value.

6 Acknowledgements

We would like to thank the reviewers for their comments. This work is supported in parts by research grants from NSF awards 214796 and 2101085.

References

1. National Institute of Standards and Technology, "Security Requirements for Cryptographic Modules," Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 140-2, Change Notice 2 December 03, 2002, U.S. Department of Commerce, Washington, D.C., 2001.
2. P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.
3. D. J. Bernstein and T. Lange, "Security dangers of the NIST curves," in *Invited talk, International State of the Art Cryptography Workshop, Athens, Greece*, 2013.
4. D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *International Workshop on Public Key Cryptography*, pp. 207–228, Springer, 2006.
5. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 124–142, Springer, 2011.
6. A. Faz-Hernández, J. López, and R. Dahab, "High-performance implementation of elliptic curve cryptography using vector instructions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 45, no. 3, pp. 1–35, 2019.
7. H. Hisil, B. Egrice, and M. Yassi, "Fast 4 way vectorized ladder for the complete set of montgomery curves," *Cryptology ePrint Archive*, 2020.
8. K. Nath and P. Sarkar, "Security and efficiency trade-offs for elliptic curve Diffie–Hellman at the 128-bit and 224-bit security levels," *Journal of Cryptographic Engineering*, pp. 1–15, 2021.
9. 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, pp. 493–514, 2015.
10. 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.
11. S. Ullah and R. Zahilah, "Curve25519 based lightweight end-to-end encryption in resource constrained autonomous 8-bit IoT devices," *Cybersecurity*, vol. 4, no. 1, pp. 1–13, 2021.
12. F. Turan and I. Verbauwhede, "Compact and flexible FPGA implementation of Ed25519 and X25519," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 3, pp. 1–21, 2019.
13. M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani, "Fast, small, and area-time efficient architectures for key-exchange on Curve25519," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, pp. 72–79, IEEE, 2020.
14. M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Cryptographic accelerators for digital signature based on Ed25519," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 7, pp. 1297–1305, 2021.
15. 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.
16. Z. Liu, P. Longa, G. C. Pereira, O. Reparaz, and H. Seo, "FourQ on Embedded Devices with Strong Countermeasures Against Side-Channel Attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 665–686, Springer, 2017.

17. L. Weissbart, Ł. Chmielewski, S. Picek, and L. Batina, "Systematic side-channel analysis of Curve25519 with machine learning," *Journal of Hardware and Systems Security*, vol. 4, no. 4, pp. 314–328, 2020.
18. M. Hamburg, "Ed448-Goldilocks, a new elliptic curve," *Cryptology ePrint Archive*, 2015.
19. T. Oliveira, J. López, H. Hışıl, A. Faz-Hernández, and F. Rodríguez-Henríquez, "How to (pre-) compute a ladder," in *International Conference on Selected Areas in Cryptography*, pp. 172–191, Springer, 2017.
20. H. Seo, "Compact implementations of Curve Ed448 on low-end IoT platforms," *ETRI Journal*, vol. 41, no. 6, pp. 863–872, 2019.
21. H. Seo and R. Azarderakhsh, "Curve448 on 32-bit ARM Cortex-M4," in *International Conference on Information Security and Cryptology*, pp. 125–139, Springer, 2020.
22. M. Anastasova, M. Bisheh-Niasar, H. Seo, R. Azarderakhsh, and M. M. Kermani, "Efficient and Side-Channel Resistant Design of High-Security Ed448 on ARM Cortex-M4," in *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 93–96, IEEE, 2022.
23. P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.
24. 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.
25. H. Seo and H. Kim, "Multi-precision multiplication for public-key cryptography on embedded microprocessors," in *International Workshop on Information Security Applications*, pp. 55–67, Springer, 2012.
26. H. Seo and H. Kim, "Consecutive operand-caching method for multiprecision multiplication," *Journal of information and communication convergence engineering*, vol. 13, no. 1, pp. 27–35, 2015.
27. H. Seo, M. Anastasova, A. Jalali, and R. Azarderakhsh, "Supersingular isogeny key encapsulation (SIKE) round 2 on ARM Cortex-M4," *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1705–1718, 2020.
28. H. Seo, "Memory efficient implementation of modular multiplication for 32-bit ARM Cortex-M4," *Applied Sciences*, vol. 10, no. 4, p. 1539, 2020.
29. M. Anastasova, R. Azarderakhsh, and M. M. Kermani, "Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4129–4141, 2021.
30. M. Anastasova, M. Bisheh-Niasar, R. Azarderakhsh, and M. M. Kermani, "Compressed SIKE Round 3 on ARM Cortex-M4," in *International Conference on Security and Privacy in Communication Systems*, pp. 441–457, Springer, 2021.
31. S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)." RFC 8032, Jan. 2017.
32. ARM., "Cortex-M4 ISA." Last accessed on May 1, 2022 from https://developer.arm.com/documentation/100166/0001.
33. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *International workshop on cryptographic hardware and embedded systems*, pp. 119–132, Springer, 2004.
34. M. Scott and P. Szczechowiak, "Optimizing multiprecision multiplication for public key cryptography," *Cryptology ePrint Archive*, 2007.
35. Y. Lee, I.-H. Kim, and Y. Park, "Improved multi-precision squaring for low-end RISC microcontrollers," *Journal of Systems and Software*, vol. 86, no. 1, pp. 60–71, 2013.