Side-Channel Analysis and Countermeasure Design for Implementation of Curve448 on Cortex-M4

Mojtaba Bisheh-Niasar*
mbishehniasa2019@fau.edu
Florida Atlantic University
Boca Raton, Florida, USA

Mila Anastasova
manastasova2017@fau.edu
Florida Atlantic University
Boca Raton, Florida, USA

Abubakr Abdulgadir
abubakr.abdulgadir@pqsecurity.com
PQSecure Technologies, LLC
Boca Raton, Florida, USA

Hwajeong Seo
hwajeong84@gmail.com
Hansung University
Seoul, South Korea

Reza Azarderakhsh
razarderakhsh@fau.edu
Florida Atlantic University
PQSecure Technologies, LLC
Boca Raton, Florida, USA

ABSTRACT

The highly secure Curve448 cryptographic algorithm has been recently recommended by NIST. While this algorithm provides 224-bit security over elliptic curve cryptography, its implementation may still be vulnerable to physical side-channel attacks. In this paper, we present a speed-optimized implementation on a 32-bit ARM Cortex-M4 platform achieving more than 40% improvement compared to the best previous work. Our design can perform 43 scalar multiplications per second on an STM32F4 working at 168 MHz. At 24 MHz, our proposed implementation takes only 3,740k clock cycles. On the other hand, the security of Curve448 is thoroughly evaluated to have a trade-off between performance and required protection. We apply different effective countermeasures to prevent a subset of side-channel and fault injection attacks at the cost of 8%-22% overhead.

CCS CONCEPTS

• **Security and privacy → Cryptanalysis and other attacks**.

KEYWORDS

Cortex-M4, Curve448, elliptic curve cryptography, hardware security, scalar multiplication, side-channel

ACM Reference Format:
Mojtaba Bisheh-Niasar, Mila Anastasova, Abubakr Abdulgadir, Hwajeong Seo, and Reza Azarderakhsh. 2022. Side-Channel Analysis and Countermeasure Design for Implementation of Curve448 on Cortex-M4. In *Hardware and Architectural Support for Security and Privacy (HASP '22), October 01, 2022, Chicago, IL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3569562.3569564

HASP '22, October 01, 2022, Chicago, IL, USA

ACM ISBN 978-1-4503-9871-8/22/10…$15.00
https://doi.org/10.1145/3569562.3569564

1 INTRODUCTION

Key exchange protocols establish secret keys between two or more parties through an insecure channel such as the internet. Classical elliptic curve cryptography (ECC) key exchange protocols such as Diffie-Hellman rely on the difficulty of solving discrete logarithm problems. Recently, Curve448 has been recommended by NIST [11] and IETF [19] to address backdoor issues in other ECC constructions [5], while Safe-Curve policies [6] are considered in its design procedures. Curve448, designed by Hamburg [17, 22], offers 224-bit security for applications at a higher security level as a part of the TLS [25] and OpenSSH protocols.

On the other hand, although classical cryptosystems will be broken by emerging large-scale quantum computers, we have to develop hybrid cryptosystems to transition to post-quantum cryptography (PQC) for maintaining accordance with industry or government regulations while PQC updates will be thoroughly applied. Therefore, classical cryptosystems cannot be eliminated even if PQC will be developed.

The demand for Internet of Things (IoT) devices in the everyday world is constantly increasing; thus, the most popular and highly used ARM-based platform, Cortex-M4, has been a target of several research groups working on different projects and topics. The simple architecture of the reduced instruction set allows highly optimized energy and power consumption and area efficiency of the platform, therefore, it is used in most embedded and real-time systems.

A side-channel analysis attack (SCA) can extract the secret key by analyzing the information leakage, including timing, power consumption, and electromagnetic emissions, etc. According to the types of leakage, SCA can be categorized into timing attack, simple power analysis (SPA) attack, differential power analysis attack (DPA), and electromagnetic attack.

Although there are intensive research and many published papers dealing with Curve25519 [3, 9, 10, 16, 27], Curve448 has not been thoroughly investigated in the literature. To the best of our knowledge, there appear to be extremely few Curve448 implementations. The only Curve448 implementation over Cortex-M4 was proposed by Seo et al. [31], computing 26 scalar multiplications per second at 168 MHz. Although this work did consider optimized field arithmetic, it utilized extended affine and projective coordinates

and only investigated constant-time algorithms. Our proposed architecture is both faster and more secure than [31].

This work builds upon [31] in two critical ways: (i) This work employs a significantly improved field arithmetic with careful memory management resulting in an 18% speedup in low-level arithmetics. (ii) This work achieves 40% speedup using efficient restricted-X coordinates wherein not only does our architecture inherently provide protection against timing and SPA attacks, but also advanced security mechanisms can be included with a limited performance penalty to avoid DPA attacks, which is missing in the literature.

Curve448 implementations and SCA evaluation over the FPGA platform were proposed in [7, 28, 29]. Additionally, the scalar multiplication architectures over Ed448, equivalent to Curve448, were presented on Cortex-M4, AVR, MSP, and FPGA platforms in [2], [30], and [8], respectively.

Given the complexities of the Curve448 with extended field size on resource constraint devices, a study on designing speed-optimized implementation is well deserved; however, the secure implementation challenges are intensified due to information leakages. Note that the protected scheme should be resistant against timing, SPA, and DPA attacks.

Contributions. In this work, we report speed record results for the implementation of Curve448 improving 40% of the total time compared to the best previous work, targeting the resource-restricted Cortex-M4. Our contributions are: (i) We propose an optimized low-level field arithmetics based on continuous alternation between addition/subtraction to reduce memory access for the carry/borrow catcher technique. The newly proposed design permits to development of two arithmetic operations working on long integers in parallel show 18% performance improvement. (ii) We also utilize the Refined-Operand Caching method achieved by a Cortex-M4 DSP instruction and register optimization techniques to reduce the memory access instructions. We also employ the interleaved reduction technique into the multi-precision multiplication providing 18% speedup. (iii) Our modular operation cleverly exploits the special form of the prime numbers used in Curve448 to reduce the number of memory accesses along with completely modifying the implementation. (iv) Additionally, several side-channel and fault injection (FI) countermeasures are implemented and evaluated to protect the scheme from the most relevant attacks. We provide performance results and side-channel countermeasures for our implementation providing a trade-off between performance and required protection.

The rest of this paper is organized as follows: In Sec. 2, some relevant mathematical background are reviewed. In Sec. 3, the proposed implementation is investigated. In Sec. 4, the results and comparison with other works are discussed. We evaluate SCA countermeausres in Sec. 5. Eventually, we conclude this paper in Sec. 6.

2 PRELIMINARIES

In this section, the mathematical background of Curve448 will be briefly introduced.

2.1 Curve448 Arithmetic

A Montgomery curve E is expressed over $GF(p)$ is defined by: $E: \quad y^2 \equiv x^3 + A \cdot x^2 + x \bmod p$, where $p = 2^{448} - 2^{224} - 1$ and $A = 156326$. This curve is birationally equivalent to an untwisted Edwards curve called Edwards448 [19]. scalar multiplication can be efficiently performed using the Double-and-Add algorithm. However, this algorithm is vulnerable if the point doubling and point addition can be visually distinguished in the power measurements of a scalar multiplication [13]. The Double-and-always-Add algorithm is used to prevent timing and SPA attacks.

Algorithm 1 The Montgomery ladder based scalar multiplication over Curve448

Input: $k, \mathcal{P} = (x_p, y_p)$
Require: the x-coordinate of $\mathcal{Q} = k \cdot \mathcal{P}$
Initial step: $\mathcal{P}_1 = (X_1, Z_1) = (1, 0), \mathcal{P}_2 = (X_2, Z_2) = (x_p, 1)$

1: **for** i **from** 447 **downto** 0 **do**
2: **if** $k_i = 0$ **then**
3: $(\mathcal{P}_1, \mathcal{P}_2) = ladderstep(x_p, \mathcal{P}_1, \mathcal{P}_2)$
4: **else**
5: $(\mathcal{P}_2, \mathcal{P}_1) = ladderstep(x_p, \mathcal{P}_2, \mathcal{P}_1)$
6: **end if**
7: **end for**
8: **return** $x_q = X_1/Z_1$

Group operation over the Montgomery curves can be accelerated and also protected using the Montgomery ladder [23] using differential addition formulas presented in projective coordinates. This algorithm provides fast and constant-time execution of PA and PD. A point $\mathcal{P} = (x, y)$ from affine is presented in projective coordinates such that $(x, y) = (X/Z, Y/Z)$. Given two points $\mathcal{P}_1 = (X_1, Z_1)$ and $\mathcal{P}_2 = (X_2, Z_2)$, and the difference $\mathcal{P}_1 - \mathcal{P}_2 = (X_3, Z_3)$, a single step of the Montgomery ladder computes two points $\mathcal{P}_{PD} = 2 \cdot \mathcal{P}_1$ and $\mathcal{P}_{PA} = \mathcal{P}_1 + \mathcal{P}_2$ such that:

$$X_{PD} = (X_1 - Z_1)^2 \cdot (X_1 + Z_1)^2 \tag{1}$$
$$Z_{PD} = 4X_1Z_1 \cdot (X_1^2 + dX_1Z_1 + Z_1^2) \tag{2}$$
$$X_{PA} = Z_3((X_1 - Z_1) \cdot (X_2 + Z_2) + (X_1 + Z_1) \cdot (X_2 - Z_2)) \tag{3}$$
$$Z_{PA} = X_3((X_1 - Z_1) \cdot (X_2 + Z_2) - (X_1 + Z_1) \cdot (X_2 - Z_2)) \tag{4}$$

Algorithm 1 shows the point multiplication based on Montgomery ladder step. After performing 448 steps of the Montgomery ladder, a modular inversion is required to map the result from projective to affine coordinates. By Fermat's Little Theorem, $a^{-1} \equiv a^{p-2} \bmod p$ will be computed by consecutive operations, including 447 squaring and 15 multiplications.

2.2 Side-Channel Protection

Implementations of scalar multiplication are known to be vulnerable to side-channel analysis, e.g., [3, 15, 29]. SCA protection should be considered at both algorithmic and implementation levels.Performing inherently resistant algorithms, e.g., Montgomery Ladder or Fermat's little theorem, achieves constant-time implementation and secret-independent implementation. The authors in [13] introduced several countermeasures to scalar multiplication to prevent DPA vulnerability, including point randomization, scalar blinding.

The point $\mathcal{P} = (x_p, y_p)$ can be projected using a random value $\lambda \in \mathbb{Z}_{2^{448}} \setminus \{0\}$ such that $\mathcal{P}_r = (\lambda \cdot x_p, \lambda)$. Although the point is

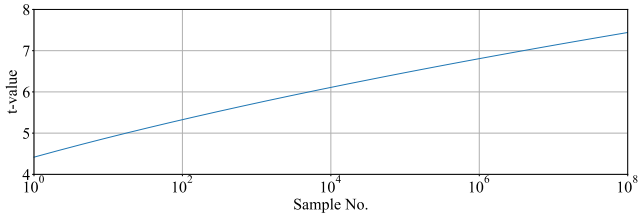

Figure 1: TVLA threshold as a function of the number of samples per trace. The value is calculated based on [14] assuming a statistical confidence level < 0.00001 and a large degree of freedom.

randomized, the same point $Q = k \cdot \mathcal{P}_r$ corresponding to a constant scalar will be computed, i.e., $x_p = \frac{X}{Z} = \frac{\lambda X}{\lambda Z}$.

The scalar blinding can be achieved by adding a multiple of group order $\#E$ such that $k_r = k + r \times \#E$ where r is a random value. However, the same point $Q = k_r \cdot \mathcal{P}$ corresponding to a constant base point will be computed, as proven as $k_r \cdot \mathcal{P} = (k + r \times \#E) \cdot \mathcal{P} = k \cdot \mathcal{P} + r \cdot \mathcal{O} = k \cdot \mathcal{P}$.

2.3 TVLA

Test vector leakage assessment (TVLA), introduced in [4], provides a robust test using a t-test to evaluate the differences between sets of acquisitions to determine if one set of measurement can be distinguished from the other. This technique can detect different types of leakages, providing a clear indication of leakage or lack thereof [34].

Given two sets of traces with n_1 and n_2 samples, we compute the corresponding sample means, $\overline{x_1}$ and $\overline{x_2}$, and respective sample standard deviations, σ_1 and σ_2. A t-statistic using Welch's t-test can be computed such that:

$$\alpha = \frac{\overline{x_1} - \overline{x_2}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_1^2}{n_2}}} \tag{5}$$

In practice, observing α greater than a specific threshold indicates the presence of leakage. The confidence threshold in [34] was set to 4.5, leading to many false positives, particularly when the captured trace is significantly extended. To avoid the false positives, the authors in [3] computed the confidence threshold for Curve25519 scalar multiplication using the threshold formula proposed in [14] and [24] equal to 7. Fig. 1 depicts the confident threshold for different captured trace length based on [14].

3 PROPOSED ALGORITHM AND ARCHITECTURE

3.1 Cortex-M4 Microarchitecture

We use the NIST recommended STM32F407-VG platform to implement our proposed protected Curve448 architecture. The Cortex-M4 architecture offers 16 general purpose register (GPR) set, including reserved registers for the Stack Pointer (SP), Link Register (LR), and Program Counter (PC). Although LR is a special register, its value can be manipulated when previously stored in the memory.

The 3-stage pipeline architecture of the platform allows a throughput of one RISC instruction per cycle. However, the load/store instructions may take up to 2 clock cycles when not properly scheduled. The consecutive memory accesses ensure pipelined execution of the instructions. The simplicity of the instruction set of the target platform makes the ARMv7-M architecture suitable for the execution of complex problems efficiently.

The powerful Multiply ACcumulate (MAC) instructions perform one multiplication storing the result in two different 32-bit registers, ensuring the entire result of 64-bit value. Possibly another two 32-bit additions can be executed where the value of the destination registers is added to the result.

3.2 Field Arithmetic

In the implementation design of the hand-coded assembly subroutines, we use the previously described instruction to speed up the low level field arithmetic operations.

The modular addition makes use of the powerful `ADD{S}`, `ADC{S}`, `SUB{S}`, and `SBC{S}` to propagate the carry/borrow among the words of the operands. In addition, we implement a carry/borrow catcher technique, which allows to develop two arithmetic operations working on long integers in parallel. In our proposed implementation, $A + B - P$ is performed on blocks of the operands, where we implement $A[0-3] + B[0-3] - P[0-3]$ and store the carry/borrow results of the last addition/subtraction in registers to later propagate to the following blocks. The carry/borrow catcher technique allows to significantly reduce the number of memory accessing instructions, thus reducing the latency of operations.

We utilized the Refined-Operand Caching method with a width of 4 for optimal 448-bit wise multi-precision multiplication, which is achieved by powerful `UMAAL` instruction and register optimization techniques. The `UMAAL` instruction performs the MAC operation without overflow conditions. For this reason, no additional carry catcher registers are required. The beginning of multiplication is performed with the `UMULL` instruction. This instruction can initialize destination registers and avoid the register initialization step efficiently. The order of instructions was also optimized to reduce the number of pipeline stalls by removing the interdependency between registers since memory access and instructions can be performed in a parallel way.

Figure 2 shows detailed descriptions of 448-bit wise multi-precision multiplication. 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 896-bit result $C = A \cdot B$ is represented as $C = (C[27], ..., C[1], C[0])$. 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 $(i = 27)$. The order of computation is from ① to ④. Since the size of caching width is 4, i.e., 128-bit, the computation is performed in 4 steps, i.e., 448/128=3.5.

3.3 Group Operations

To implement scalar multiplication of Curve448, we employ both Double-and-always-Add or Montgomery ladder algorithms which

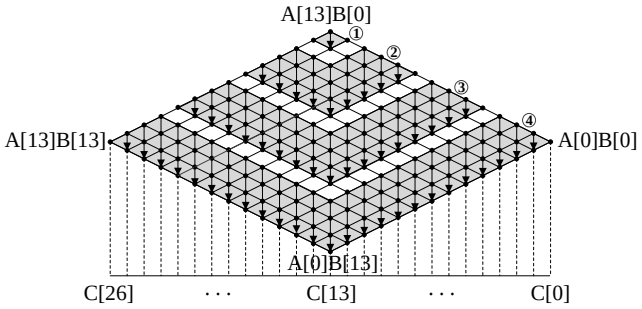

Figure 2: 448-bit wise multi-precision multiplication.

results in constant-time implementation. The design of these algorithms is important, where about 90% of the time for scalar multiplication is spent.

The Double-and-always-Add algorithm requires performing one point doubling and one point addition operation. We utilized extended projective coordinates and extended affine coordinates to perform point addition, while the result will be presented in extended projective coordinates. Additionally, point doubling can be executed in extended projective coordinates. Hence, the point addition and doubling require $7\mathbf{M} + 6\mathbf{A}$ and $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{A}$, where **M**, **S**, and **A** are a multiplication, a squaring, and an addition cost, respectively.

Our implementation uses the Montgomery ladder in homogeneous projective coordinates to perform a variable-base scalar multiplication. With respect to Algorithm 1, Algorithm 2 presents the proposed computation in one step of Montgomery ladder requiring only two additional field operands t_0 and t_1. The modular multiplication of λ with X_2 is used for randomization. Therefore, a differential point addition and point doubling formula can be performed at the cost of $5\mathbf{M} + 4\mathbf{S} + 1\mathbf{k} + 8\mathbf{A}$, where **k** is a cost of multiplication with a constant. Hence, the Montgomery ladder provides more efficient implementation.

Algorithm 2 Montgomery ladder step in randomized projective coordinates over Curve448.

Input: x_P, $\lambda \in \mathbb{Z}_{2^{448}} \setminus \{0\}$, $\mathcal{P}_1 = (X_1, Z_1)$, and $\mathcal{P}_2 = (X_2, Z_2)$
Initial Step: $(X_3, Z_3) = (\lambda \cdot x_p, \lambda)$
Require: $\mathcal{P}_{PD} = 2 \cdot \mathcal{P}_1, \mathcal{P}_{PA} = \mathcal{P}_1 + \mathcal{P}_2$

1: $t_0 \leftarrow X_1 + Z_1$
2: $Z_1 \leftarrow X_1 - Z_1$
3: $X_1 \leftarrow X_2 + Z_2$
4: $X_2 \leftarrow X_2 - Z_2$
5: $X_2 \leftarrow X_2 \times t_0$
6: $X_1 \leftarrow X_1 \times Z_1$
7: $t_1 \leftarrow X_1 + X_2$
8: $Z_2 \leftarrow X_2 - X_1$
9: $X_2 \leftarrow t_1 \times t_1$
10: $X_2 \leftarrow X_2 \times Z_3$
11: $Z_2 \leftarrow Z_2 \times Z_2$
12: $Z_2 \leftarrow Z_2 \times X_3$
13: $t_1 \leftarrow t_0 \times t_0$
14: $Z_1 \leftarrow Z_1 \times Z_1$
15: $t_0 \leftarrow t_1 - Z_1$
16: $X_1 \leftarrow t_1 \times Z_1$
17: $Z_1 \leftarrow t_0 \times 39081$
18: $Z_1 \leftarrow Z_1 + t_1$
19: $Z_1 \leftarrow Z_1 \times t_0$
20: **return** $\mathcal{P}_{PD} = (X_1, Z_1), \mathcal{P}_{PA} = (X_2, Z_2)$

3.4 Memory Management

The target platform features 1MB of flash memory and another 192KB of RAM [32]. To eliminate the slow 16KB of SRAM, we have modified the linker file, where we have specified only the first 112KB as an SRAM.

4 EXPERIMENTAL RESULTS AND COMPARISON

The implementations are benchmarked on STM32F407-VG Discovery board equipped with 32-bit ARM Cortex-M4 microcontrollers clocked at 24MHz and 168MHz. The higher frequency is more suitable for real-world application, while 24MHz requires fewer cycles due to the speed of the memory controller. The arithmetic implementations are implemented in the ARM assembly, and the libraries are compiled by GCC with optimization flags set to `-O3`.

4.1 Scalar Multiplication

As shown in [18], the measured cycle counts on the same Cortex-M4 can be different based on the clock frequency set on the chip. Hence, we report the latency requirements in two different operating frequencies. Different clock frequencies set may cause stalls on the controller if the memory is slower. For instance, the proposed multiplier employing the memory operations uses 3% more cycles when the controller is set to a 7× higher frequency.

We implement the scalar multiplication over Curve448 using two different algorithms, i.e., Double-and-always-Add and Montgomery ladder. Table 1 summarizes the number of clock cycles and latency requirements for the proposed implementation in our unprotected scheme. Both these algorithms are constant-time, while the Montgomery ladder is faster, taking 3.7×10^6 cycles, and more secure compared to Double-and-always-Add. Our results show 29% performance improvement by using the Montgomery ladder over the X-coordinate at the cost of a 40% memory utilization penalty. Compared to the previous work [31], our proposed implementation results in 1.6× speedup computing almost 43 scalar multiplications per second at 168 MHz.

Compared to other platforms, Curve448 scalar multiplication on Cortex-M4 shows more than 20× and 27× speedup in terms of cycle counts compared to 16-bit MSP430 and 8-bit AVR ATmega processors reported in [30], respectively.

Table 2 compares some pre and post-quantum schemes on embedded processors to the proposed design over the same architecture (i.e., ARMv7-M). A more technology-independent comparison is the required cycle. An operating frequency in a limited range is mostly considered to reduce the required power. Thus, our Curve448 implementation improves 5.2× and 8.9× required cycle counts compared to Secp384r1 [33] and Secp521r1 [33], respectively. However, our implementation requires 4.3× and 3.9× more cycles compared to Curve25519 [16] and Secp256r1 [20].

Curve448 can provide 224-bit security compared to the Curve25519 and Secp256r1 with 128-bit security. Therefore, higher security levels come with a performance penalty, and industry usually resists

Table 1: Cortex-M4 implementation results. Our Design I and II are based on Double-and-always-Add, and Montgomery ladder algorithms, respectively.

Algorithm	Freq. [MHz]	Latency [CC×10^3]	Time [ms]	Throughput [Op/Sec]	Memory [B]
Seo *et al.* [31]	24	6,218	259.1	3.9	-
Our Design I		5,269	219.5	4.6	564
Our Design II		3,740	155.8	6.4	788
Seo *et al.* [31]	168	6,286	37.4	26.7	-
Our Design I		5,532	32.9	30.4	564
Our Design II		3,917	23.3	42.9	788

Table 2: Implementation results on Embedded Processors

Algorithm	pre/post quantum	Cortex	Freq. [MHz]	Latency [CC×10^3]	Time [ms]	Throughput [Op/Sec]
Curve25519 [16]	pre	M4	48	907	18.9	52.9
Secp256r1 [20]	pre	M4	64	994	15.5	64.3
Four$\mathbb{Q}$ [21]	pre	M4	168	511	3.0	328.8
Secp384r1 [33]	pre	M3	100	20,200	202	4.9
Secp521r1 [33]	pre	M3	100	35,100	351	2.8
SIKEp434 KeyGen [1]	post	M4	24	68,260	2,844	0.3
Curve448 [This Work]	pre	M4	168	3,917	23.3	42.9

them. On the other hand, although we are confident with the security of ECC, there is always the possibility that algorithmic improvements reduce the required computation to break ECC. Therefore, moving to a higher security level will help keep a margin against unknown attack improvements. Hence, we propose an implementation for a level of security that can still be feasible subject to the performance requirement of the target application, such as high-end servers of constrained devices (particularly, compared to other implementations in this level, e.g., Secp384r1, Secp521r1.).

4.2 SCA-Protected Performance Results

Table 3 lists the performance results for the different SCA countermeasures. While implementing the Montgomery ladder algorithm accelerates computation over Curve448, it would also be a straightforward countermeasure against timing, SPA, and sign change fault attacks [15]. Taking advantage of the Curve448 specification, some countermeasures, such as a point validity check, is not required.

Base point randomization requires 11 multiplication operations per Montgomery ladder step, while the unprotected scheme needs 10 modular multiplication operations. Hence, as one can see, base point randomization increases 8% latency due to extending the Montgomery ladder step cycles. Additionally, the required memory for implementing this countermeasure will be increased to 796 B.

Scalar blinding is an effective countermeasure to avoid DPA, cross-correlation, safe-error, and differential fault analysis attacks. However, it can be defeated by advanced online template attacks [3] and carry-based attacks [15]. Although the scalar blinding countermeasure performs a similar Montgomery ladder step in an unprotected scheme, the number of iterations will be extended. Performing 512 iterations of the Montgomery ladder step corresponding to 64-bit randomization of scalar results in 13% latency extension, while the memory usage will be unchanged.

Stand-alone scalar blinding can be broken by the methods described in [26]. Further, stand-alone base point randomization can be broken by refined power analysis. Hence, our fourth profile

Table 3: Protected Cortex-M4 implementation results in terms of latency and dynamic memory requirements

Countermeasure	Freq. [MHz]	Latency [CC×10^3]	Time [ms]	Throughput [Op/Sec]	Memory [B]
Unprotected	24	3,740	155.8	6.4	788
Point Randomization		4,043	168.4	5.9	796
Scalar Blinding		4,226	176.1	5.7	788
Both Countermeasures		4,572	190.5	5.2	796
Unprotected	168	3,917	23.3	42.9	788
Point Randomization		4,222	25.1	39.8	796
Scalar Blinding		4,417	26.3	38.0	788
Both Countermeasures		4,789	28.5	35.1	796

combines all countermeasures in order to investigate their interaction. The latency in terms of clock cycles per operation is increased significantly due to the extended size of the secret scalar (512 bits instead of 448 bits). In addition, since we have to include an additional multiplication by Z_3, the latency is increased as well. Therefore, the required time is increased by 22% for performing a scalar multiplication.

Although a software implementation cannot be protected against arbitrarily powerful fault attackers [3], we implement a flow-counter countermeasure to improve protection against FI loop-abort attacks with negligible latency overhead. Hence, an incremental counter is implemented to detect changes in the execution flow. The attack can be detected at the end of scalar multiplication if the stored value in the counter would not be matched with the expected value.

5 SCA EVALUATION

Curve448 private key can be ephemeral or static. In the ephemeral case, the secret scalar is used only for one operation, allowing the attacker to collect one side-channel trace. The trace can then be used to recover the shared secret. In the static case, the secret scalar can be reused an arbitrary number of times. This allows the attacker to collect several side-channel traces and target the static key. Hence, the static use case requires more robust protection compared to the ephemeral setting. As will be shown, our protected implementation eliminates scalar-dependent leakage in the case of the static key, which is favorable to the attacker. However, to reduce the protection cost, either base point randomization or scalar blinding with a fewer length of randomization can be used for ephemeral implementation.

To evaluate scalar-dependent leakage, we adopt a procedure similar to [29] to evaluate the countermeasures incrementally. Specifically, we define four different leakage detection profiles to evaluate our main countermeasures gradually. The first profile provides a reference since all countermeasures are disabled. This profile allows us to confirm the validity of our test setup and serves as a baseline for the leakage assessment. The second and third profiles investigate the point randomization and scalar blinding countermeasures individually. Eventually, the fourth profile evaluates our fully protected implementation with both countermeasures enabled in combination. Hence, our leakage detection experiments are listed as follows:

- The unprotected implementation, without any countermeasures except constant-time operations, including field arithmetic and Montgomery ladder.

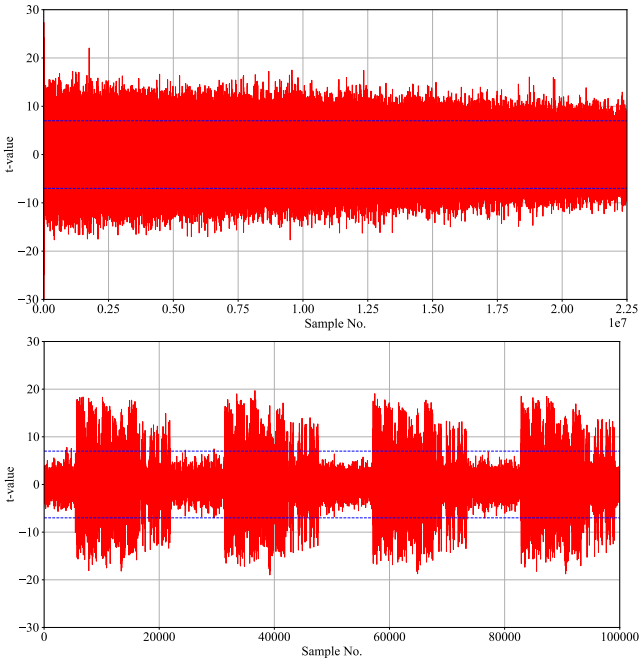

Figure 3: Overall and the part of leakage detection test on the unprotected implementation using 10,000 traces: (Up) t-test values for scalar multiplication using the Montgomery ladder, (Down) Magnified t-test values for 100,000 samples.

- The protected implementation with base point randomization countermeasure enabled.
- The protected implementation with scalar blinding countermeasure enabled.
- The protected implementation with both base point randomization and scalar blinding countermeasures enabled.

According to Fig. 1, the TVLA threshold for all our experiments is computed greater than 7; thus, we assume that peaks above 7 indicate leakage.

5.1 Side-channel analysis setup

Our experimental setup for power trace collection comprises the following components:

(1) An ARM Cortex-M4-based target board (NewAE CW308T-STM32F). The target is mounted on a NewAE CW308 UFO board [12] used to connect the target to the Chipwhisperer Lite board.
(2) The NewAE Chipwhisperer Lite board [12]. This board is used to communicate with the target board and is connected to the control PC.
(3) A control PC that sends test vectors one at a time to the control board and collects power traces from the oscilloscope.
(4) A USB3-based oscilloscope (Picoscope 3000). This oscilloscope has a bandwidth of 200 MHz and an 8-bit sample resolution.

We run the target Cortex-M4 at 25 MHz in all our experiments. The power traces are collected in AC through a passive probe connected to the CW308 UFO at a sampling rate of 125 MS/s. The

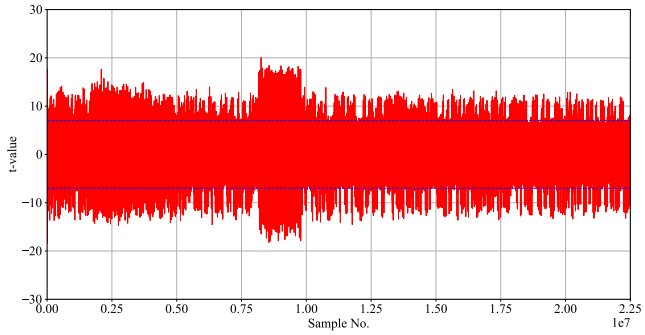

Figure 4: Protected implementation by point randomization after applying TVLA with a pool of 10,000 measurements.

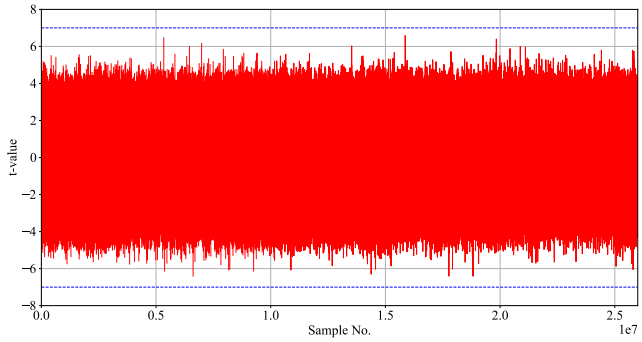

Figure 5: Protected implementation by scalar blinding after applying TVLA with a pool of 10,000 measurements.

control PC is simply connected to the ChipWhisperer Lite with a USB to micro-USB cable. The power trace is measured using the onboard shunt resistor located in the CW305 board. A trigger signal is asserted at the start of the scalar multiplication to instruct the oscilloscope to start collecting data to ensure trace alignment.

In all our leakage detection profiles, we evaluate scalar-dependent leakage and collect 10,000 traces using a non-specific fixed-vs-random TVLA. Specifically, we randomly interleave between feeding the target board with a fixed scalar and a randomly generated scalar.

We utilized the setup described above to perform the leakage assessment experiments. Profile 1, i.e., the unprotected implementation, is a constant-time implementation that makes it resistant to timing attacks and is not leaking any information about the scalar through timing. However, it can still leak the scalar through power consumption. Evaluating this implementation confirms the setup's ability to detect leakage and serves as a baseline for the tests.

Fig. 3 depicts the results of the t-tests for the unprotected implementation. In the case of the unprotected implementation, the highest peak is reaching 22, showing a significant information leakage for our unprotected baseline implementation. Fig. 3 (Down) magnifies 100,000 samples of the measured t-values. According to this figure, we can see that the information leaks periodically in each step of the Montgomery ladder.

With the same measurement setup, we then evaluate the leakage with enabling base point randomization and scalar blinding

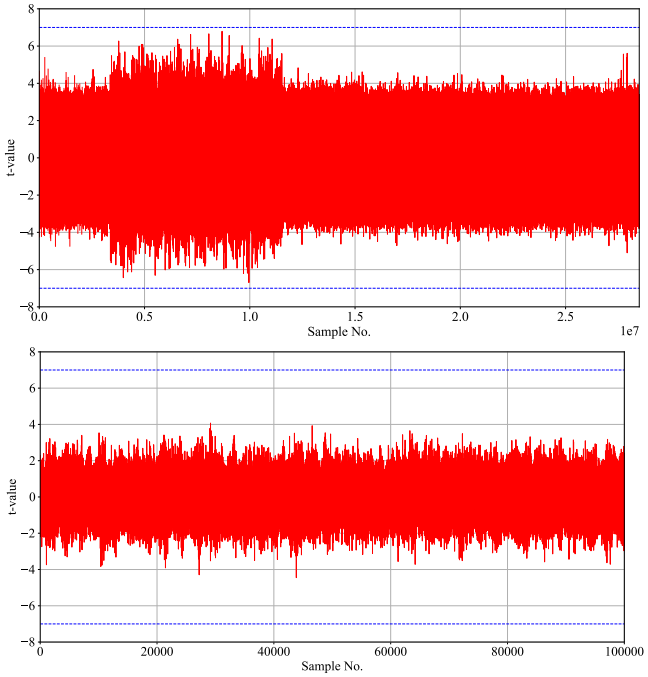

Figure 6: Overall and the part of leakage detection test on protected scalar multiplication by both base point randomization and scalar blinding after applying TVLA with a pool of 10,000 measurements: (Up) t-test values for scalar multiplication, (Down) Magnified t-test values for 100,000 samples.

countermeasures individually. The second leakage detection profile investigates the individual security gain by base point randomization without scalar blinding countermeasures. The results are shown in Fig. 4. Although the observed leakage is reduced compared with the unprotected implementation, the leakage is significant. This is expected since the countermeasure randomizes the point only, and scalar-dependent leakage is not prevented.

To evaluate the effectiveness of scalar blinding countermeasure, we show the t-test result in Fig. 5. Again, we capture 10,000 power traces, while each scalar is blinded externally using a 64-bit random value. As one can see, this countermeasure avoids any detectable scalar-dependent leakage.

Fig. 6 demonstrates the fourth evaluation profile combining all techniques in order to analyze and evaluate the interaction of both countermeasures. As we expect, first-order side-channel information leakage for Curve448 scalar multiplication cannot be observed. Fig. 6 (Down) magnifies 100,000 samples of the measured t-values showing no side-channel leakage compared to periodical leaking on unprotected implementation.

6 CONCLUSION

We present a secure implementation of Curve448 targeting a 224-bit security level for the 32-bit ARM Cortex-M4 architecture that performs the scalar multiplication computation in about 23 milliseconds at 168 MHz. We use randomized projective coordinates for the base point and scalar blinding countermeasures, reducing the speed by approximately 8% and 13%, respectively. However, we note that combining these techniques increases 22% total latency. Our leakage assessment with 10,000 power measurements shows that our implementation does not leak side-channel information while the scalar blinding and point randomization countermeasures are enabled.

ACKNOWLEDGMENTS

The authors would like to thank the comments by reviewers. This work is supported in parts by NSF 1801341 and 2101085.

REFERENCES

[1] Mila Anastasova, Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari Kermani. 2021. Compressed SIKE Round 3 on ARM Cortex-M4. In *SecureComm 2021, September 6-9, 2021, Proceedings, Part II*, Vol. 399. Springer, 441–457.

[2] Mila Anastasova, Mojtaba Bisheh-Niasar, Hwajeong Seo, Reza Azarderakhsh, and Mehran Mozaffari Kermani. 2022. Efficient and Side-Channel Resistant Design of High-Security Ed448 on ARM Cortex-M4. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2022, McLean, VA, USA, June 27-30, 2022*. IEEE, 93–96.

[3] Lejla Batina, Lukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. 2021. SCA-secure ECC in software - mission impossible? *IACR Cryptol. ePrint Arch.* (2021), 1003.

[4] Georg T. Becker, Jim Cooper, Elizabeth K. DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, T. Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. 2013. Test Vector Leakage Assessment (TVLA) methodology in practice.

[5] Daniel J. Bernstein and Tanja Lange. 2011. Security dangers of the NIST curves.

[6] D. J. Bernstein and T. Lange. 2016. SafeCurves: choosing safe curves for elliptic-curve cryptography. URL: https://safecurves.cr.yp.to/..

[7] Mojtaba Bisheh Niasar, Reza Azarderakhsh, and Mehran Mozaffari Kermani. 2020. Efficient Hardware Implementations for Elliptic Curve Cryptography over Curve448. In *21st International Conference on Cryptology, Indocrypt 2020, India, December 13-16, 2020*.

[8] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. Area-Time Efficient Hardware Architecture for Signature Based on Ed448. *IEEE Trans. Circuits Syst. II Express Briefs* 68, 8 (2021), 2942–2946.

[9] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. Cryptographic Accelerators for Digital Signature Based on Ed25519. *IEEE Trans. Very Large Scale Integr. Syst.* 29, 7 (2021), 1297–1305.

[10] Mojtaba Bisheh Niasar, Rami El Khatib, Reza Azarderakhsh, and Mehran Mozaffari Kermani. 2020. Fast, Small, and Area-Time Efficient Architectures for Key-Exchange on Curve25519. In *27th IEEE Symposium on Computer Arithmetic, ARITH 2020, Portland, OR, USA, June 7-10, 2020*. 72–79.

[11] Lily Chen, Dustin Moody, Andrew Regenscheid, and Karen Randall. 2019. Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters. *Computer Security, Draft NIST Special Publication, National Institute of Standards and Technology* 800-186 (2019).

[12] New Technology Inc.: CHIPWHISPERER. 2021. URL: https://www.newae.com/chipwhisperer.

[13] Jean-Sébastien Coron. 1999, Worcester, MA, USA. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Cryptographic Hardware and Embedded Systems, CHES'99*, Çetin Kaya Koç and Christof Paar (Eds.). 292–302.

[14] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. 2017. Towards Sound and Optimal Leakage Detection Procedure. In *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10728)*. Springer, 105–122.

[15] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. 2010. State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures. In *HOST2010, 13-14 June 2010, California, USA*. 76–87.

[16] Hayato Fujii and Diego F. Aranha. 2017. Curve25519 for the Cortex-M4 and Beyond. In *Progress in Cryptology - LATINCRYPT 2017, Havana, Cuba, September 20-22, 2017*, Tanja Lange and Orr Dunkelman (Eds.), Vol. 11368. Springer, 109–127.

[17] Mike Hamburg. 2015. Ed448-Goldilocks, a new elliptic curve. *IACR Cryptology ePrint Archive* 2015 (2015), 625.

[18] B Hasse. 2017. Memory bandwidth influence makes Cortex M4 benchmarking difficult. *CHES2017 rump session* (2017).

[19] A. Langley, M. Hamburg, and S. Turner. 2016. Elliptic Curves for Security.

[20] Emill Lenngren. 2021. P256-Cortex-M4. URL: https://github.com/Emill/P256-Cortex-M4.

[21] Zhe Liu, Patrick Longa, Geovandro C. C. F. Pereira, Oscar Reparaz, and Hwajeong Seo. 2017. FourQ on embedded devices with strong countermeasures against side-channel attacks. *IACR Cryptol. ePrint Arch.* (2017), 434.

[22] Mike Hamburg. 2015. Ed448-Goldilocks, A new high-strength curve and implementation. URL: https://csrc.nist.gov/csrc/media/events/workshop-on-elliptic-curve-cryptography-standards/documents/presentations/session7-hamburg-michael.pdf.

[23] Peter L. Montgomery. 1987. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Math. Comp.* 48 (1987), 243–264.

[24] Louiza Papachristodoulou, Apostolos P. Fournaris, Kostas Papagiannopoulos, and Lejla Batina. 2019. Practical Evaluation of Protected Residue Number System Scalar Multiplication. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 1 (2019), 259–282.

[25] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[26] Thomas Roche, Laurent Imbert, and Victor Lomné. 2019. Side-channel attacks on blinded scalar multiplications revisited. In *International Conference on Smart Card Research and Advanced Applications*. Springer.

[27] Pascal Sasdrich and Tim Güneysu. 2015. Implementing Curve25519 for Side-Channel-Protected Elliptic Curve Cryptography. *ACM Transactions on Reconfigurable Technology and Systems* 9, 1 (2015), 3:1–3:15.

[28] Pascal Sasdrich and Tim Güneysu. 2017. Cryptography for Next Generation TLS: Implementing the RFC 7748 Elliptic Curve448 Cryptosystem in Hardware. In *Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, June 18-22, 2017*. 16:1–16:6.

[29] Pascal Sasdrich and Tim Guneysu. 2018. Exploring RFC 7748 for Hardware Implementation: Curve25519 and Curve448 with Side-Channel Protection. *J. Hardware and Systems Security* 2, 4 (2018), 297–313.

[30] Hwajeong Seo. 2019. Compact implementations of Curve Ed448 on low-end IoT platforms. *ETRI Journal* 41, 6 (2019), 863–872.

[31] Hwajeong Seo and Reza Azarderakhsh. 2020. Curve448 on 32-Bit ARM Cortex-M4. In *Information Security and Cryptology - ICISC 2020 - 23rd International Conference, Seoul, South Korea, December 2-4, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12593)*, Deukjo Hong (Ed.). Springer, 125–139.

[32] ST. 2020. STM32F405xx STM32F407xx Datasheet - production data.

[33] Hannes Tschofenig and Manuel Pegourie-Gonnard. 2015. Performance of State-of-the-Art Cryptography on ARM-based Microprocessors.

[34] Michael Tunstall and Gilbert Goodwill. 2016. Applying TVLA to Public Key Cryptographic Algorithms. *IACR Cryptol. ePrint Arch.* (2016), 513.