SCA-secure ECC in software – mission impossible?

Lejla Batina
Radboud University, The Netherlands
lejla@cs.ru.nl

Łukasz Chmielewski
Radboud University, The Netherlands
Riscure, The Netherlands
lukaszc@cs.ru.nl

Björn Haase
Endress+Hauser Liquid Analysis GmbH&Co.
KG, Germany
bjoern.m.haase@web.de

Niels Samwel
Radboud University, The Netherlands
nsamwel@cs.ru.nl

Peter Schwabe
Max Planck Institute for Security and Privacy,
Bochum, Germany
Radboud University, The Netherlands
peter@cryptojedi.org

ABSTRACT

This paper describes an ECC implementation computing the X25519 key-exchange protocol on the ARM-Cortex M4 microcontroller. This software comes with extensive mitigations against various side-channel and fault attacks and is, to our best knowledge, the first to claim affordable protection against multiple classes of attacks that are motivated by distinct real-world application scenarios. We also present results of a comprehensive side-channel evaluation. We distinguish between X25519 with ephemeral keys and X25519 with static keys and show that the overhead to protect the two is about 36% and 239% respectively. While this might seem to be a high price to pay for security, we also show that even our (most protected) static implementation is more efficient than widely-deployed ECC cryptographic libraries, which offer much less protections.

KEYWORDS

Elliptic Curve Cryptography; Side-Channel Analysis; Fault Injection

1 INTRODUCTION

Elliptic-curve cryptography (ECC) presents the current state of the art in public-key cryptography, both for key agreement and for digital signatures. The reason for ECC getting ahead of RSA is mainly in its suitability for constrained devices due to much shorter keys and certificates, which results in ECC consuming much less energy and power than RSA. Also, ECC has an impressive security record: while certain classes of elliptic curves have shown to be weak [76, 99], attacks against ECC with a conservative choice of curve have not seen any substantial improvement since ECC was proposed in the mid-80s independently by Koblitz [63] and Miller [77]. The best known attack against ECC is still the Pollard rho algorithm [88] and its parallel version by van Oorschot and Wiener [106].

The situation is different for *implementation attacks* against elliptic-curve cryptosystems. Since Kocher put forward the concept of side-channel analysis (SCA) against cryptographic implementations and fault injection (FI) analysis, i.e., the Bellcore attacks, was introduced in the late 90s [22, 64, 65], research into attacks against implementations of ECC and suitable countermeasures has been a very active area. The most common forms of SCA are Simple Power Analysis (SPA) and Differential Power Analysis (DPA) (both introduced in [65]), and profiled attacks. SPA involves visually interpreting power consumption traces over time to recover the secret key while DPA relies on performing a statistical analysis between intermediate values of cryptographic computations and the corresponding side-channel traces[1]. Profiling attacks, most notably template attacks (TAs), were introduced in [25]. Here the adversary first uses a device (for instance, a copy of the attacked device) that is under his/her full control to characterize device leakage; this characterization information is called a *template* for TAs. Then the adversary uses that information to efficiently recover the secret data from the device under attack.

We will review the relevant classes of attacks later in the paper, but to name a few recent real-world examples of such implementation attacks against ECC consider the “Minerva” timing attack against multiple implementations of ECDSA [59], or the power-analysis attack recovering the secret key from a TREZOR hardware bitcoin wallet [52]. In fact, the research area has been so active that it produced at least four survey papers reviewing the state of the art in attacks and countermeasures at different points in time [1, 30, 35, 36].

In this paper we also survey the state of the art in attacks and countermeasures and consequently parts of this paper have the character of a SoK paper. However we go further than just systematizing knowledge: In this paper we aim at *consolidating* knowledge by bringing together (and extending) the state of the art through implementations of one of the most widely used ECC primitives with extensive SCA and FI countermeasures.

Complete, specific, additive, and public. The survey by Fan and Verbauwhede from 2013 [36] concludes that a side-channel-protected implementation of ECC needs to satisfy three properties:

- the protection mechanisms need to be *complete*, i.e., protect against all relevant attacks;
- they need to be *specific* to the application scenario and resulting attacker model; and
- they need to *additive*, i.e., combined in a way that does not introduce new vulnerabilities.

In this paper we add a fourth attribute to this list: to enable independent verification of these properties, the implementation needs to be *public*.

Surprisingly, despite the extensive body of literature on implementation attacks on ECC, we are not aware of a single paper describing an implementation fulfilling these four properties or

[1]Nowadays DPA is rarely used in favor of an improved method called Correlation Power Analysis (CPA) [23] that replaces difference of means with Pearson correlation.

even making somewhat substantiated claims to do so. For a more detailed discussion see the section on related work below.

Note that there are implementations of ECC that claim to fulfill the three properties introduced in [36]. For example, NXP advertises the SmartMX2-P40 *"secure smart card controller"* as supporting *"DES, AES, ECC, RSA cryptography, hash computation, and random-number generation"* and claims that the protection mechanisms are *"neutralizing all side channel and fault attacks as well as reverse engineering efforts"* [84]. However, like essentially all commercial smart cards, all implementation details are kept secret, which actively prevents academic discourse and public evaluation of these claims. As a result of this situation many papers are presenting side-channel attacks against ECC target implementations that never claimed protection against such attacks; for example, see [43, 93].

We do not mean to say that there are no public implementations of ECC including *certain* countermeasures against some specific attacks. On the contrary: most papers describing attacks also suggest countermeasures and some of these papers also present implementations of those countermeasures; see the section on related work below. However, more than 3 decades after the invention of ECC and 2 decades after the invention of side-channel attacks we still don't know the cost of ECC implementations with *complete*, *specific*, and *additive* SCA countermeasures, or if such implementations are even achievable with respect to the budgets in area, memory, power and energy that are typically sparse for low-cost devices.

In this paper we present the first—to our knowledge—publicly available implementation of a widely used ECC primitive claiming to fulfill the three properties for a concrete real-world application scenario and assuming a rather strong adversarial model on both, side-channel and fault analysis.

The overhead required by our protected implementations is about 36% for the ephemeral implementation and 239% for the static one. While this might seem a relatively hefty price to be paid for security, note that it is rather modest in comparison to securing AES on ARM architectures. In particular, Schwabe et al. [97] show that masked bitsliced AES has an overhead of 458% compared to unprotected bitsliced AES and of 1339% compared to a table-based AES; all on the same ARM Cortex-M4 platform we consider in this paper. Moreover, we also show in Section 5.1 that even our most protected static implementation is more efficient than widely-deployed ECC cryptographic libraries, which provide much less SCA protections.

Security evaluation vs. formal proofs. The current state of the art of formally verifying side-channel protection of asymmetric crypto is to prove the absence of timing leakage with tools such as ct-verif [4] (on LLVM IR level) or Binsec/Rel [67] (on binary level). Computer-verified proofs of side-channel protections beyond timing-attack resistance are much further in the symmetric-crypto realm (see, e.g., [8, 9, 13, 33, 79]) and we believe that it is a very important direction of research to extend such results also to asymmetric cryptography. For example, we believe that it is possible to prove that our static X25519 implementation is secure against first-order DPA attacks, but such a proof would require significant advances of the currently available tools for formal verification. For many attacks that are we aim to protect against in this paper because of their real-world importance—most notably profiling attacks and fault-injection attacks—we are not even aware of a widely accepted sound and complete *definition* of what it means for an implementation to be secure. This is why we follow the current state of the art in practical security evaluation and provide extensive experimental evidence to evaluate the security of our implementations.

Application scenario. Long-term secrets as used in authentication protocols typically form the most precious assets of any security infrastructure. Today, sophisticated communication capabilities are incorporated in more and more systems that formerly used to operate in a stand-alone setting. One prominent example are Internet-of-Things (IoT) devices, serving applications both for industry as well as consumer applications. Most of these devices today do not incorporate dedicated cryptographic hardware designed for adequately protecting long-term-keys.

Moreover, physical access to many devices cannot be effectively restricted, making installations significantly more vulnerable to implementation attacks exploiting all kinds of leakage such as electromagnetic emanation, power consumption etc., than typical for the traditional office or server-room setting. This notably applies to large-scale or de-centralized industrial plants such as common in petrochemistry or local drinking-water wells.

In most critical-infrastructure systems, conventional smart-card circuits cannot not be used, specifically when considering constraints imposed by power-budget or rough environmental conditions or frequent update requirements.

For instance, wireless modules for off-the shelf industrial sensors often have to operate within a power budget of less than 2 mW [48] clearly exceeded by most off-the-shelf smart-card chips. Appliances for medical use or for production of food and drugs might have to withstand high sterilization temperatures. Conventional smart-card circuits are typically not designed for these environments, forcing implementers to choose conventional microcontrollers specifically designed for the respective constraints.

Moreover note that in many consumer applications, commercial pressure pushes designers to use conventional microcontrollers for cryptographic operations instead of dedicated smart-card circuitry.

One aspect which should be considered is that customization of typical smart-card chipsets is often impossible. For high-volume applications specialized software, such as java-card applets is common. However, the access to freely programmable circuits is limited. For applications with smaller market volumes tailored solutions mostly could not be realized. The common "turn-key" smart-card products on the other hand often allow only for very limited flexibility regarding the supported cryptographic protocols.

Why consider software only? The side-channel and fault-attack-protected implementation of X25519 that we present in this paper is a software-only implementation targeting the ARM Cortex-M4 microcontroller. Some would argue that applications that are seriously concerned about side-channel security will rely (at least to some extent) on countermeasures implemented in hardware.

The main reason that we go for a software-only implementation is the application scenario described above, which cannot afford to use specialized hardware[2] Clearly this application scenario calls for an evaluation of how far we can go in software-only countermeasures. We also stress that, while we use this application as a motivation, there are other scenarios —for example, many open-source projects—that cannot afford producing special-purpose hardware.

However, another reason for following a software-only approach is our goal to encourage independent security evaluation of our implementation. Development boards featuring our target platform are widely available for around US$ 20, allowing any side-channel researcher to try to attack our implementation.

Why target X25519? The concrete primitive we target with side-channel and fault-attack protections in this paper is the X25519 elliptic-curve key-exchange originally proposed under the name "Curve25519" by Bernstein in [16]. To avoid confusion between naming of the specific elliptic-curve used in the protocol and the protocol itself, Bernstein later suggested to call the curve Curve25519 and the protocol X25519 [17].

One reason to target this primitive is its widespread use in many state-of-the-art security protocols like in TLS [83], SSH [2], the Noise Framework [104, Sec. 12], the Signal protocol [101], and Tor [73]. For a more extensive list see [54]. Moreover X25519 is now also considered as primitive for the OPC/UA protocol family [37] by the OPC/UA security working group, specifically targeting IoT applications with both increased protection demands and low computational resources.

Another reason is that in this paper we make an attempt to settle discussions about the cost of SCA-protected X25519 implementations. On the one hand it is generally acknowledged that the "Montgomery ladder" (see Section 2), which is typically used in X25519 implementations, is a good starting point for SCA protected implementations. On the other hand, during standardization of Curve25519 for TLS, concerns were expressed about the structure of the underlying finite field and group order [69, Sec. 3.1] leading to a significant increase of the cost of side-channel protections. The main argument refers to the presumed need of significantly larger scalar blinding. We show that these concerns can be resolved without excessive performance penalties in comparison to, e.g., curves over fields without structure allowing for fast reduction.

Also the fact that the full group of curve points has a co-factor of 8 has raised concerns in the context of SCA [44]. As one way to alleviate these subgroup concerns it is often suggested to validate inputs [5, 7, 32]; however Bernstein's Curve25519 FAQ [15] states *"How do I validate Curve25519 public keys? Don't."*. For the sake of security we have decided to include a full on-curve check for the static implementation and an early-abort strategy on certain malicious inputs for the ephemeral case.

Related work. There is a vast amount of literature devoted to the topic of side-channel secure implementations of ECC in both, software and hardware. Most related work considers different types of curves, but on the same platform, i.e., the ARM Cortex-M4 microcontroller such as [68] or the same curve but without such a comprehensive side-channel protection [42, 94].

[2]While we concentrate on software only, note that our implementation can be relatively easily modified to use a hardware co-processor for modular arithmetic instead of our assembler arithmetic routines, without modifying higher-level SCA countermeasures.

In [68] a range of high-speed implementations are proposed, including: scalar multiplication, ECDH key exchange, and digital signatures on various embedded devices using the Four$\mathbb{Q}$ elliptic curve of Costello and Longa. They also add countermeasures to thwart a variety of side-channel attacks. All implementations are constant time and include countermeasures such as scalar and projective coordinate randomizations and point blinding. The implementation with the countermeasures resulted in a slowdown of factor 2 for the ARM platform. To validate the effectiveness of the countermeasures, leakage detection is performed using test vector leakage assessment (TVLA) [14]. They show an improved resistance to SCA, as expected. DPA also failed when countermeasures were deployed. However, active and profiling adversaries are not considered. Hence, this work does not offer a comprehensive evaluation but rather a solid benchmark in evaluating trade-offs for certain classes of side-channel attacks (SPA and DPA).

Fujii and Aranha [42] present an X25519 implementation that is protected against timing attacks by constant-time execution and also randomized projective coordinates and constant-time conditional swaps were implemented. However, the authors do not specify any details about those countermeasures and they do not consider protection against other side-channel attacks.

Considering hardware implementations, Sasdrich and Güneysu performed extensive investigations of costs for countermeasures in hardware, i.e., on an FPGA platform for Curve 25519 and Curve448 [95, 96]. In both cases, the Montgomery ladder was deployed to provide a basic protection against timing and Simple Power Analysis (SPA). To offer some DPA protection, point randomization and scalar blinding were added, increasing the amount of look-up tables (LUTs) by 5% and of flip-flops (FFs) by 40% and increasing the overall latency in terms of clock cycles by 45% for Curve448. For Curve25519 the performance penalty was 30% with a similar increase in required LUTs and FFs. In addition, the authors add memory address scrambling to secure the memory accesses against side-channel attacks and correspondingly make DPA more complex. For this purpose 2^6 different random addresses were used.

A protected implementation of the new complete formulas for Weierstrass curve from [90] is presented in [26]. The implementation is put on an FPGA and evaluated against SCA. More specifically, three different versions were evaluated: (1) an unprotected architecture; (2) an architecture protected through coordinate randomization; and (3) an architecture with both coordinate randomization and scalar splitting. The evaluation is done through timing analysis and TVLA. The results show that applying an increasing level of countermeasures leads to an improved resistance against SCA, but they only consider a side-channel adversary of a limited capacity i.e. being only passive and not capable of profiling.

Availability of the software We place all software described in this paper into the public domain. It is available from an anonymous GitHub repository at https://anonymous.4open.science/r/40fe2d05-17f5-439c-9e89-7a4737e3322e/.

2 PRELIMINARIES

In this section we introduce the necessary background on the X25519 key-exchange protocol and on the ARM Cortex-M4 microcontroller. Furthermore we introduce our attacker model.

2.1 X25519 key exchange

The X25519 elliptic-curve key-exchange protocol is based on arithmetic on the elliptic curve in Montgomery form [78]:

$$E : y^2 = x^3 + 486662x^2 + x$$

over the finite field $\mathbb{F}_p$ with $p = 2^{255} - 19$. The group of $\mathbb{F}_p$-rational points on E has order $8 \cdot \ell$, where ℓ is a 252-bit prime. The central operation is scalar multiplication $\mathsf{smult}(k, x_P)$, which receives as input two 32-byte arrays k and x_P. Each of those arrays is interpreted as a 256-bit integer in little-endian encoding; the integer x_P is further interpreted as an element of $\mathbb{F}_p$. The scalar-multiplication routine first sets the most significant bit of x_P to zero (ensuring that $x_P \in \{0, \ldots, 2^{255} - 1\}$ and sets the least significant 3 bits of k and the most significant bit of k to zero, and the second-most significant bit of k to one (ensuring that $k \in 8 \cdot \{2^{251}, \ldots, 2^{252} - 1\}$). This operation on bits of the input k is often referred to as "clamping", in our pseudocode we denote it by clamp. Subsequently, scalar multiplication outputs the x-coordinate $x_{[k]P}$ of the point $[k]P$ where P is one of the two points with x-coordinate x_P on E (if there are such points) or the quadratic twist of E (otherwise), and where $[k]$ denotes scalar multiplication by k. The scalar multiplication is commonly implemented using the Montgomery ladder [78] using a projective representation $(X : Z)$ of an x-coordinate $x = X/Y$. Pseudocode for this ladder is given in Algorithm 1.

Algorithm 1 The Montgomery ladder for x-coordinate-based scalar multiplication on $E : y^2 = x^3 + 486662x^2 + x$

```
Input: $k \in \{0, \ldots, 2^{255} - 1\}$ and the $x$-coord. $x_P$ of point $P$
Output: $x_{[k]P}$, the $x$-coordinate of $[k]P$
  $X_1 \leftarrow 1; Z_1 \leftarrow 0; X_2 \leftarrow x_P; Z_2 \leftarrow 1, p \leftarrow 0$
  for $i \leftarrow 254$ downto 0 do
      $c \leftarrow k[i] \oplus p$                          ▹ $b[i]$ denotes bit $i$ of $k$
      $p \leftarrow b[i]$
      $(X_1, X_2) \leftarrow \mathsf{cswap}(X_1, X_2, c)$
      $(Z_1, Z_2) \leftarrow \mathsf{cswap}(Z_1, Z_2, c)$
      $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}(x_P, X_1, Z_1, X_2, Z_2)$
  end for
  return $(X_1, Z_1)$
```

This algorithm uses two sub-routines cswap and ladderstep. The cswap ("conditional swap") routines swaps the first two inputs iff the last input is 1. The ladderstep routine, on input coordinates $x_{Q-P}, X_P, Z_P, X_Q, Z_Q$, computes $(X_{[2]P}, Z_{[2]P}, X_{P+Q}, Z_{P+Q})$. This computation takes 5 multiplications, 4 squarings, one multiplication by a small constant, and a few additions and subtractions in $\mathbb{F}_{2^{255}-19}$.

The key-exchange protocol uses this smult function and the fixed basepoint $x_B = [9, 0, \ldots, 0]$ and proceeds with straight-forward elliptic-curve Diffie Hellman.

2.2 The ARM Cortex-M4 microcontroller

Our target platform is the ARM Cortex-M4 microprocessor, which implements the ARMv7ME 32-bit RISC architecture [100]. The most relevant features of this platform for the software described in this paper are the following:

- it features 16 general-purpose 32-bit registers, out of which 14 are freely usable (one is used as instruction pointer and one as stack pointer);
- it features a single-cycle 32×32-bit multiplier producing a 64-bit result, which can be accumulated for free through a fused multiply-accumulate instruction.

The Cortex-M4 features a three-stage pipeline. Recent findings [74, 98, 114] indicate that specific properties of the internal pipeline architecture are highly relevant for the amount of side-channel leakage generated. In the instruction set, input and output operands of the ALU are retrieved from and stored to the register file. However, based on the analysis of [114] and own findings, we presume that the Cortex-M4 features also shortcut data paths which generate additional leakage when the result of a first arithmetic or logic instruction in the pipeline is required as input operand for a subsequent instruction within the pipeline.

Randomness generation. The side-channel countermeasures require a source of uniformly random bytes. Random-number generation is not part of the Cortex-M4, however the specific STM32F407 device that we used for evaluation features a hardware random number generator, which generates 4 random bytes every 40 clock cycles [100]. We use this hardware RNG for all randomness generation. In some contexts we need uniformly random values modulo p or modulo ℓ. In those cases we sample a 512-bit integer and reduce modulo q or ℓ. This approach is also used, for example, for sampling close to uniformly random modulo ℓ in the Ed25519 signature scheme [18, 19]. Compared to a software RNG, the hardware RNG provides faster and higher quality randomness.

Fast X25519 on the Cortex-M4. Multiple earlier papers describe optimized implementations of X25519 on the ARM Cortex-M4 [41, 42]. The speed-record within the scientific literature is currently held by an implementation by Haase and Labrique described in [49]. They report 625 358 cycles for one scalar multiplication on an STM32F407 running at 16 MHz. The optimized routines for field arithmetic from this implementation are in the public domain and available from https://github.com/BjoernMHaase/fe25519. They are the starting point for our protected implementations.

2.3 Attacker model

Our attacker model is motivated by typical capabilities of a real-world attacker. We assume that an attacker controls the input to the scalar multiplication and obtains the contents of the output buffer after the computation has finished. Furthermore, we assume certain capabilities with respect to the side-channel leakage that becomes available (passive attacker) and FI capability (active attacker).

Passive side-channel attacker. We assume that an attacker can collect sufficiently many (noisy) traces of power or EM leakage together with the chosen input and corresponding output. In addition, we allow for the attacker to generate templates (as in profiled attacks) for self-controlled inputs on a device of the same make and model as the target device. For our experiments we generate templates on the same device as the one we target in the attack.

As mentioned above, instead of proving the side-channel security of our implementation in a certain model (e.g., the noisy leakage model [89]), we use the TVLA methodology (t-test analysis) to show the absence of leakage in traces collected from an actual device running our implementation. This kind of analysis, although not perfect, is standard for security evaluation labs [55, 110].

A limited fault attacker. A software-only implementation, such as the one we describe in this paper, cannot be protected against arbitrarily powerful fault attackers. The reason is that such an attacker could simply skip the execution of the program over any dedicated countermeasure or rewrite values in registers and memory to eliminate all effects of fault-attack countermeasures. The fault attacker we consider in this paper is therefore limited to injecting only one fault per scalar multiplication. We assume that this fault may either skip a short block of consecutive instructions, set an arbitrary register value to zero, or set an arbitrary register value to a random value not controlled by or known to the attacker. These faults are the most common to occur in practice based on our experience and on reviewed literature; see, for example, [103].

Restricting instruction skips to "short" blocks is motivated by the fact that in practice attacks are typically able to skip only up to 2–3 instructions. Furthermore allowing arbitrarily long blocks of code to be skipped would enable trivial attack skipping all cryptographic computations. For similar reasons we also exclude the instruction pointer from the set of "arbitrary" registers that the attacker may fault. Note also that we consider attacks skipping over the function call to our protected implementations out of scope – such attacks need to be protected by the surrounding context.

3 SCA-PROTECTED EPHEMERAL X25519

In this section we describe our implementation of X25519 protected against side-channel and fault attacks when deployed in an *ephemeral* key-exchange scenario. In such a scenario each secret scalar is used only twice, once in public-key generation and once in the computation of the shared secret key. By itself such an ephemeral key exchange makes little sense in most communication scenarios because communication partners are unauthenticated. However, combined with either signatures in a SIGMA-style protocol [66] or in combination with static key exchanges in, for example, 3DH [72], it is a critical building block to achieve forward secrecy.

The goal of an SCA or FI attacker against such an ephemeral key exchange is to learn the shared secret. This goal can be achieved either by learning the secret scalar or by influencing the scalar multiplication through FI during both key generation and shared-key computation (and scalar multiplication is the main part of that) to an easily guessable value. The advantage of having two instead of just one trace is very small for a passive attacker – it is certainly too few traces to allow any kind of differential attacks (see Section 4).

As we focus on protecting the scalar multiplication, we consider generation of the 32-byte secret key as out of scope and assume that it is provided as an input by the user of the library.

In Subsection 3.1 we list most relevant passive attacks and in Subsection 3.2 we list most relevant active attacks for the ephemeral setting. Due to space constraints we do not list all possible SCA or FI attacks, but only focus on the most relevant ones. For a complete list of attacks we refer the reader to either [36] or [61].

3.1 Relevant passive attacks

SPA. Simple power analysis (SPA) attacks were introduced in [65]. Note that we do not emphasize on the word "power", but also include attacks that use a trace of electromagnetic radiation or any other side-channel. Modern definitions of SPA often include any attack that works with side-channel information of a single execution. This definition would include horizontal and certain template attacks. We discuss those separately and hence define SPA attacks in the more classical sense as attacks that visually identify secret-data-dependent control flow. Simply speaking, SPA attacks typically target instruction-dependent leakages that are often data-dependent.

Horizontal (collision) attacks. A class of single-trace attacks is often called *horizontal* attacks. These attacks trace back to [107], and was applied against ECC in [12, 28, 50]. The general idea of horizontal collision is to use key-dependent collisions of the same values across multiple iterations of the scalar multiplication loop. Consider, for example, an attacker able to distinguish whether two finite-field multiplications have an input in common. If this is the case then the attacker can learn whether two scalar multiplication iterations share the same scalar bit. Such single-trace attacks have been described against RSA [107] and ECC [50].

Horizontal correlation attacks. In [28], the authors proposed an attack based on predictions of intermediate arithmetic results within a single trace. Although this method is effective against private-key blinding countermeasures, other mitigation methods, e.g., message or point randomization, prevent this horizontal attack.

Template attacks. Template attacks (TAs), introduced in [25], require a profiling stage in which the adversary estimating the probability distribution of the leaked information to make a better use of information present in each sample. TA is the best attack technique from an information-theoretic point of view but it makes strong assumptions on the attacker, e.g., that they can collect an unbounded number of template traces and the noise is close to Gaussian.

An approach from [75], is to target the internal state of the ladder step after a small number of scalar bits are processed. If a single-trace TA is successful then the number of bits is recovered and the attack can continue on the subsequent bits targeting the same trace.

Another relevant profiling attack is the TA targeting the conditional move (cmov) instruction as introduced in [82]. The cmov instructions is often used to implement cswap. The idea of this attack in our context is to learn about the behavior of cswap and create the corresponding templates (from multiple traces). Subsequently, the templates can be used to recover scalar bits from all cswap operations used in a single scalar multiplication. The attack is complex, but it can be successful even against a single trace.

The third relevant class for TA attacks key-transfer [85]. Note that in most implementations the scalar is copied, from flash to RAM or within RAM, just before the scalar multiplication is executed. This might allow the attacker to template the copy process and try to recover information about the scalar or the session key (i.e., the output of the scalar multiplication) from a single attacked trace.

Deep Learning Attacks. Another class of profiled attacks, similar to TAs, are deep learning (DL) side-channel attacks [24, 62, 71].

These attacks use algorithms like multilayer perceptron (MLP) or convolutional neural networks (CNNs) to recover the secret keys from cryptographic implementations. Their main advantage over TAs is that they do not require pre-processing of leakage traces, require less trace alignment, and make the attack simpler to run. They can be applied to attack ECC as demonstrated in [108, 109, 116]. A similar attack, but unsupervised, is presented in [87].

Online Template Attacks. Online template attacks (OTA) [11] use horizontal techniques to exploit the fact that an internal state of scalar multiplication depends only on the known input and the scalar. Advanced types of those attacks need only one leakage trace and can defeat implementations protected only with scalar blinding or splitting. OTA traces back and extends the doubling and collision attacks [40, 53]. These kind of attacks are thwarted by randomizing the internal state using point blinding and projective coordinate randomization [29] or coordinate re-randomization [82].

Horizontal cmov attacks. Horizontal cmov attacks [81] are similar to TA from [82], but they are unsupervised. They combine a heuristic approach based on clustering with multiple-trace unsupervised points-of-interest selection. Due to the relaxation of the attack setting, these attacks have slightly lower success rate and their complexity is increased in comparison to [82].

3.2 Relevant active attacks

Weak curve attacks. The first weak-curve attack on ECC was introduced in [20]. The key observation is that a_6 in the definition of E is not used in the addition. Hence, the addition formula for curve E generates correct results for any curve E' that differs from E only in a_6: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6'$. Therefore, the adversary can cheat by providing a point P on the insecure curve E'. Then the adversary can solve ECDLP on E' and find out the scalar.

The method of moving a scalar multiplication from a strong curve E to a weak curve E' often requires fault induction. With the help of faults, the adversary can make use of invalid points [20], invalid curves [27], and twist curves [38] to hit a weak curve.

Loop-abort attacks. A rather simple fault-injection attack is to jump out of the scalar multiplication main loop after only a few iterations or completely skip over the loop, which makes the resulting output easily predictable. An attacker against an ephemeral key exchange who is able to jump out of the loop in the same iteration in both key generation and shared-key computation can force the device into generating a weak, easily predictable shared key.

Key Shortening. Another simple fault injection attack is to shorten the scalar. The idea of the attacker is to stop the copy loop of the scalar before it is being used in the scalar multiplication. This way entropy in the scalar is reduced, if the previous memory content is predictable (e.g., if it is zeroed).

Fixing scalar-multiplication output. A straightforward attack to enforce predictable output is to prevent copying of the final result or use a fault to directly write, e.g., zero values, to the output buffer.

3.3 Protected implementation

Algorithm 2 presents a pseudocode of our protected implementation of ephemeral X25519. Like most previous X25519 implementations it is running in "constant time", i.e., is not leaking any information about the scalar through timing. Furthermore, we add the following countermeasures: re-randomizing the projective representation in the cswaprr procedure and control-flow counter.

The strategy of our cswaprr, which merges conditional swapping with projective re-randomization, takes into account that memory access leaks significantly more than register operations. We thus fetch input words from memory, conditionally swap and randomize in registers and then store the results back. Randomization here means multiplying both the X and the Z coordinate by a 29-bit random value. We also considered the risk of increased leakage due to shortcut data paths in the core's pipeline. We avoided using a result from an immediately preceding instruction as input for the subsequent operations if operands hold information on the secret scalar.

We moreover take into account that the CPU core always processes 32 bits simultaneously in its single-cycle multiplication and logic instructions. We aim at increasing the noise level by embedding random data into unused operand bits in addition to the confidential information. For instance, conditional moves are implemented on a half-word basis by using two multiplicative masks M_1,M_2 having random data in bits 16 to 31. Bits 15 to 0 are zeroed and the condition (M_1) and negated condition (M_2) is held in bit 0. For inputs A, B, the calculation $A' \leftarrow M_1 \cdot A + M_2 \cdot B$; $B' \leftarrow M_1 \cdot B + M_2 \cdot A$; yields the swapped lower half-words of the inputs in bits 0 to 15.

In the following we explain why this implementation is not vulnerable against the attacks from the last two subsections.

SPA attacks are thwarted by using secret-independent control flow. This is rather clearly achieved in the Montgomery ladder, as long as the conditional-swap operation (implemented by us in cswaprr) is carefully implemented without conditional branches.

Using a ladder to compute the scalar multiplication is a commonly recommended countermeasure against "classical" SPA attacks; see, for example, [65].

Horizontal correlation attacks rely on the fact that identical values (i.e., field elements) are used across two or more loop iterations. In our implementation, this class of attacks is thwarted by re-randomizing the projective representation of the two points in the state in each iteration. This re-randomization is merged into the cswaprr operation as detailed above. Each re-randomization is using 29 bits of randomness, which leaves a small chance that occasionally this random value is of a special form that does not fully remove correlation (e.g., the randomizer could be 1 or 2). We do not expect this to be a problem in practice, as an attacker would only very occasionally be able to learn one or two bits of an ephemeral scalar by exploiting these rare correlations.

Horizontal, Template, Deep Learning, and Online Template attacks extract information about the secret scalar by identifying temporary values used in the computation through matching against templates. As a recommended [10, 11] countermeasure against online-template attacks we employ projective randomization [80] with a full-size randomizer chosen uniformly random from $\mathbb{F}_p$. This countermeasure also stops template attacks that are based on exploiting the internal ladder state leakage.

Algorithm 2 Pseudocode of side-channel and fault-attack protected ephemeral X25519

```
Input: A 255-bit scalar $k$ and the $x$-coordinate $x_P$ of some point $P$
Output: $x_{[k]P}$
1: $ctr \leftarrow 0$                                              ▷ Initialize iteration counter
2: $x_P \xleftarrow{\$} \{0, \dots, 2^{256} - 1\}$                  ▷ Initialize output buffer to random bytes
3: $k \leftarrow \mathrm{clamp}(k)$
4: $k \leftarrow k/8$                                              ▷ Divide scalar $k$ by 8 to account for initial 3 doublings
5: Increase($ctr$)
6: $(X_P, Z_P) \leftarrow \mathrm{montdouble}(x_p, 1)$
7: $(X_P, Z_P) \leftarrow \mathrm{montdouble}(X_p, Z_P)$
8: $(X_P, Z_P) \leftarrow \mathrm{montdouble}(X_p, Z_P)$            ▷ 3 doublings to multiply by co-factor 8
9: Increase($ctr$)
10: if $Z_P = 0$ then
11:     go to Line 27                                              ▷ Early-abort if input point is in order-8 subgroup
12: end if
13: $x_P \leftarrow X_P \cdot Z_P^{-1}$                             ▷ Return to affine $x$-coordinate
14: $X_1 \leftarrow 1, Z_1 \leftarrow 0$
15: $Z_2 \xleftarrow{\$} \{0, \dots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$   ▷ Initial randomization of projective representation
16: $k \leftarrow k \oplus 2k$                                      ▷ Precompute condition bits for cswap
17: Increase($ctr$)
18: for $i$ from 252 downto 1 do                                    ▷ Main scalar-multiplication loop
19:     $(X_1, Z_1, X_2, Z_2) \leftarrow \mathrm{cswaprr}\,(X_1, Z_1, X_2, Z_2, k[i])$   ▷ projective re-randomization merged with cswap
20:     $(X_1, Z_1, X_2, Z_2) \leftarrow \mathrm{ladderstep}\,(x_P, X_1, Z_1, X_2, Z_2)$
21:     Increase($ctr$)
22: end for
23: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathrm{cswaprr}\,((X_1, Z_1, X_2, Z_2), k[0])$
24: $x_P \leftarrow X_2 \cdot Z_2^{-1}$
25: Increase($ctr$)
26: if Verify($ctr$) then                                          ▷ Detected wrong flow, including iteration count
27:     $x_P \xleftarrow{\$} \{0, \dots, 2^{256} - 1\}$             ▷ Set output buffer to random bytes
28: end if
29: return $x_P$
```

However, that countermeasure does not stop the single-trace attacks. Therefore, the following attacks might be feasible: single-trace TA or DL targeting cswaprr (see [82], for example) and single-trace TA or DL targeting key loading. The ephemeral implementation does not explicitly protects against such attacks – see the evaluation in Section 5.2. However, the static implementation adds additional protections against such attack; for details, see Section 4.

Weak curve attacks. Since we are implementing Curve25519, many weak-curve attacks are not possible; for example, see the paragraph on small-subgroup attacks in [16, Sec. 3]. However, an attacker can try to insert a point in the order-8 subgroup; such inputs are mapped to the neutral element (represented by $Z_P = 0$) through the three doubling steps in lines 5–7 of Algorithm 2. Lines 9–10 detect this neutral element and abort. In typical X25519 implementations those doublings are performed at the end of the scalar-multiplication loop, because the lowest 3 bits of the scalar are set to zero through clamping. Moving them to the beginning ensures that the input to the main loop is either of order ℓ or of order 1; even if an attack skips the early-abort through an injected fault.

Loop-abort attacks. In order to protect against loop-abort attacks we employ a flow-counter countermeasure [111]. Specifically, we use a single counter monotonously incremented throughout the scalar multiplication to detect changes in the execution flow. If the value of this counter does not match the expected value at the end of the computation then we return a random value.

Key Shortening. There are two ways how a fault-injection attacker can reduce entropy in the secret scalar: either by setting parts of the key to zero or by skipping over instructions that copy the scalar. We do not implement any particular countermeasure against those attacks for the following reasons. First, as we are on a 32-bit architecture, an attacker can set at most 32 out of the 256 bits to zero with a single fault, which is not sufficient to allow any practical attacks. Second, we have verified that the copying loop is unrolled by the compiler so again, an attacker can control at most 32 bits of the scalar with single FI. We did consider duplicating the scalar-copy operation, but while this would help against fault-injection attacks, it would make SCA easier.

Fixing scalar-multiplication output. In order to prevent an attacker from directly influencing the values in the output buffer, we set the content of this buffer to a random value before starting with the main computation and only copy the result to this buffer if the flow-counter check was successful. An attacker who is restricted to one fault can prevent either randomization or copying of the valid result, but not both – which would be required to obtain a predictable value.

Combined active/passive attacks. It might be possible for an advanced attacker to combine SCA and FI attack as in [34]. This specific attack is disabled by randomization of point coordinates and point blinding. Another strategy for such an attacker could be to use FI to disable some SCA countermeasures and then proceed with SCA. In this paper we consider such an attacker largely out of scope, but in particular the re-randomization of the projective coordinates in every loop iteration makes life for even such a combined attacker hard: skipping over *one* re-randomization step would not reveal much information and skipping over re-randomization in multiple steps would require more than one fault.

4 SCA-PROTECTED STATIC X25519

This section describes our implementation of X25519 protected against SCA and FI attacks when deployed in an *static* key-exchange scenario. In this scenario the secret scalar can be used an arbitrary number of times, contrary to the implementation from Section 3.

The goal of an attacker against the static key exchange is to either obtain the long-term secret key or the output, i.e., session key. This goal can be either achieved by learning the scalar or by influencing scalar multiplication through FI during both key generation and shared-key computation to an easily guessable value.

The static scalar in our library is protected with various static masks. The scalar and these masks need to be generated by the user and then stored in the library. Afterwards the static masks are automatically updated during scalar multiplication executions and should not be modified or accessed by the user. If possible they should be stored in the memory isolated from the user. The Cortex-M4 does not feature memory isolation, thus for this work we consider that protecting the private key in memory is out of scope.

Below in Section 4.1 we list most relevant passive attacks and in Section 4.3 we list most relevant active attacks. Note that these lists are extensions of the lists from Section 3. Moreover, since the static implementation implements scalar blinding, a countermeasure that randomizes a scalar for each scalar multiplication execution, most of the attacks are effectively reduced to the ephemeral case.

4.1 Relevant passive attacks

DPA attacks. With the first publication introducing DPA [65] it was clear that the technique applies to all cryptosystems relying on long-term secrets. The idea is based on the fact that the same secret is used over and over again, which allows the attacker to build a statistical attack. The only requirement is to identify the so-called selection function that takes as inputs some known data and the secret he/she aims at recovering. In a static key exchange, the secret is typically the scalar i.e. the private key and the known input is the point it gets multiplied with. Considering this scenario, the attacker can recover the secret key bit by bit, by simply collecting enough traces for each loop of scalar multiplication and making the hypothesis on intermediate results.

Cross-Correlation Attack Cross-correlation exploits collisions in subsequent additions and doubling of a scalar multiplication algorithm using many traces [112]. The horizontal correlation attacks from Section 4.3 can be traced back to this multi-trace attack.

Special-point attacks. A refined power analysis (RPA) attack exploits the existence of special points: $(x, 0)$ and $(0, y)$. Feeding to a device a point P that leads to a special point $R(0, y)$ (or $R(x, 0)$) at step i under the assumption of processed bits of the scalar will generate exploitable side-channel leakage [47]. A zero-value point attack (ZPA) [3] is an extension of RPA. Not only considering the points generated at step i, a ZPA also considers values of auxiliary registers. For some special points P, some auxiliary registers might predictably have zero value at step i. The attacker can then use the same procedure of RPA to incrementally reveal the whole scalar.

Carry-based attack. The carry-based attack [39] does not attack the scalar multiplication itself but its countermeasures. It relies on the carry propagation occurring when long-integer additions are performed as repeated sub-word additions. For instance, on an 8-bit processor, Coron's first countermeasure, $k' = k + r\#E$, is normally performed with repeated 8-bit additions. Let k_i and $r_i\#E$ denote the i-th sub-word of k and $r_i\#E$, respectively. Note that k_i is fixed and $r_i\#E$ is random in different executions. The crucial observation here is that, when adding k_i to $r_i\#E$, the probability of the carry out $c = 1$ depends mainly on k_i. The adversary can then monitor the outgoing carry bit of the adder to estimate the probability of $c = 1$. With this probability, the value of k_i can be guessed reliably.

Address-bit DPA. The address-bit DPA attack [29] explores the link between the register address and the key. It was the first time applied to ECC in [56]. This attack is applicable if the addresses of coordinates processed in the ladder step depend in some way on the corresponding scalar bit. Essentially, the scalar bit can be recovered if the attacker can distinguish between data read from different addresses.

Address template attacks. The address leakage mentioned above in the address-bit DPA, can be exploited using a template attack.

Observe that when attacking a single trace address TA is essentially equivalent to the TA targeting the cswap [82] (Subsection 3.1), even if the attacked leakage is not coming from the address but from the cswap logic. Therefore, when we talk about address leakage in this paper, we also mean the cswap leakage.

4.2 Relevant active attacks.

Safe-error attacks. The concept of safe-error was introduced in [60, 115]. Two types of safe-error are reported: C safe-error and M safe-error. The C safe-error attack exploits dummy operations which are usually introduced to achieve SPA resistance, like add-and-double-always, for example. The adversary can try to induce temporary faults during the execution of the dummy operation. If the result is unaffected then it means that indeed a dummy operation was affected. Otherwise, in case of different result, the attacker learn that a real operation was affected. This is enough to learn a scalar bit in the attacked iteration. The M safe-error attack exploits the fact that faults in some memory blocks will be cleared. The attack was first proposed in [60] to attack RSA, but it also applies to scalar multiplication. The goal of the attack is to affect memory that is only overwritten if a scalar bit has a certain value, e.g., 1.

Differential fault analysis. The Differential Fault Attacks (DFA) on scalar multiplication [20, 21] use the difference between the correct results and the faulty results to deduce certain bits of the scalar. These attacks require multiple correct and incorrect results of scalar multiplications to learn the static scalar. Since we randomize the scalar, as described later, these attacks are not applicable and we do not detail on them here due to the space constraints.

4.3 Protected implementation

The static protected scalar multiplication is presented in Algorithm 3. Similarly to Section 3, we employ re-randomizing the projective representation using cswaprr and control-flow counter.

Moreover, to prevent SCA attacks against key transfer, we implement scalar storage blinding: the scalar k is stored as $k_{f^{-1}} = k \cdot f^{-1}$ together with f, which is a 64-bit random blinding. To protect against attacks that use special points as input, we also use static random points R and S for input point blinding, where $R, S = [-k]R$. These blindings f, R, S and $k_{f^{-1}}$ are always securely randomized after each scalar multiplication. They should also be stored securely. In particular, if possible in the given architecture, they should be stored in a secure memory not accessible to users of the library. That is why these values are marked as secure input in Algorithm 3.

To further improve security, the scalar is randomized with an additional 64-bit value r that multiplicatively masks each scalar multiplication. Then the mask is removed by performing an additional 64-bit scalar multiplication and is not stored.

To circumvent address leakage we follow the approach of [51] and implement the address-randomization technique from [57]. This countermeasure adds additional random cswaprr executions to make the cswaprr sequence in the scalar multiplication independent from the scalar itself. Although this countermeasure is called address-randomization for historical reasons, it also should thwart cswap-like leakage as shown in [51].

Below we explain why our static implementation is not vulnerable against the attacks from the last two subsections.

DPA attacks. To prevent DPA, Coron [29] suggested three countermeasures for an EC-based key exchange: randomizing a point, projective coordinates, and the scalar for every protocol execution. We implement all these methods in our implementation and, as confirmed by the evaluation in Section 5, we show its DPA-resistance.

Cross-Correlation Attack Due to the scalar blinding this attack is essentially reduced to horizontal correlation attack and this is stopped by the projective re-randomization.

Special-point attacks. All special points attacks are stopped by the initial point blinding with $R, S = [-k]R$. This countermeasure ensures that in each scalar multiplication iteration the point used by the ladder is independent from the input point.

Carry-based attack. The carry-based attack is thwarted by the usage of the storage scalar blinding. Essentially the scalar is re-blinded after every usage and therefore, no single guess about k or k_i can be made between multiple blinding computations.

Address-bit DPA. This attack is not possible since scalar is freshly randomized in each scalar multiplication execution.

Address template attacks. Only single-trace attacks are applicable due to the employed randomizations. The single-trace horizontal, template, and DL attacks can directly target whatever instructions involve the scalar bits and are much harder to thwart. To stop them we implement the address-randomization [57] following the approach from [51]. We check its effectiveness in Section 5.2.

An alternative countermeasure against address attacks is suggested in [70]. It is shown to be effective, but the proposed cswap implementation does not consider the risk of correlation between memory loads and stores of the unchanged subwords, before and after swapping. We choose the more conservative option of storing conditionally swapped operands only after having them projectively re-randomized first. Thus our implementation avoids the risk of correlation between loaded and stored (swapped) operands.

Safe-error attacks and differential fault analysis. Due to the scalar blinding, these attacks are not applicable.

5 EVALUATION

This section presents benchmark results of our implementations (Section 5.1) and our side-channel evaluation results to support our claims of resistance against DPA and profiled attacks (Section 5.2).

5.1 Performance Evaluation

We measured clock cycles of the implementations on an STM32F407 Discovery development board and applied the common practice of clocking it down to at 24 MHz for obtaining reproducible cycle counts that are not significantly affected by wait cycles of the memory subsystem [49]. We use the gcc compiler, version 10.2.1 20201103 (release), GNU Arm Embedded Toolchain 10-2020-q4-major. The performance evaluation results are presented in Table 1.

We see that both SCA and FI protections come at a significant cost. However, the protections for the ephemeral implementation, mostly the projective re-randomization, only incur a relatively small penalty. The additional protections included in the static implementation, mainly against profiled attacks on key transfer and conditional swap, have a more significant cost even when implemented with efficiency in mind. The total overhead to protect the ephemeral implementation is about 36% and the overhead of protecting the static one is 239% in comparison to the unprotected case.

Table 1: Performance Evaluation (using -O2 optimizations).

Implementation / Countermeasure	Clock cycles:
Complete unprotected:	683 061
Complete ephemeral:	930 915
Complete static:	2 316 986
Projective re-randomization merged with cswap, cost per iteration (ephemeral & static)	1041
Randomized Addressing (static):	330 481
Update static blinding points (static):	153 911
Update static scalar (static):	199 947
Blinding of the scalar (static):	197 376
Additional 64-bit scalar multiplication (static):	331 742

Comparison. As expected our countermeasures significantly reduce the speed. Still we believe our protected implementations to be highly competitive. Speed-optimized versions of unprotected off-the-shelf-libraries such as boringSSL, bearSSL or wolfSSL require in between roughly 2 and 2.5 million cpu cycles on our target platform. They are, thus, more than two times slower than our protected ephemeral and about as fast as our fully protected static implementation. A more detailed performance and security analysis is presented in Appendix A.

5.2 Side-channel Evaluation

In this section we describe our SCA setup and then we present the results of the evaluation of the unprotected, ephemeral, and static implementations. We also discuss the resistance to profiled attacks.

Side-channel analysis setup We run our side-channel experiments using a Cortex-M4 on an STM32F407IGT6 board clocked at 168 MHz. We measure current with the Riscure Current Probe [91]

Algorithm 3 Pseudocode of side-channel and fault-attack protected static X25519

```
Input: the $x$-coordinate $x_P$ of $P$. Secure Input: a 64-bit blinding $f$, blinded scalar $k_{f^{-1}} = k \cdot f^{-1}$, and blinding points $R, S = [-k]R$
Output: $x_{[k]P}$
 1: $ctr \leftarrow 0$                                            ▷ Initialize iteration counter
 2: $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$             ▷ Initialize output buffer to random bytes
 3: Copy $k_f$ to internal state while increasing $ctr$.          ▷ Updating $ctr$ in a loop makes sure that copying cannot be affected by faults
 4: $y_P \leftarrow$ ycompute($x_P$)
 5: Increase($ctr$)
 6: $(X_P, Y_P, Z_P) \leftarrow$ ecadd($(x_P, y_P), R$)           ▷ Point blinding, output of addition of $R$ is projective
 7: $(X_P, Y_P, Z_P) \leftarrow$ ecdouble($(X_P, Y_P, Z_P)$)
 8: $(X_P, Y_P, Z_P) \leftarrow$ ecdouble($(X_P, Y_P, Z_P)$)
 9: $(X_P, Y_P, Z_P) \leftarrow$ ecdouble($(X_P, Y_P, Z_P)$)      ▷ 3 doublings to multiply by co-factor 8
10: $r \leftarrow \{1, ..., 2^{64} - 1\}$                         ▷ Sample 64-bit non-zero random value for scalar blinding
11: $b \leftarrow \{0, ..., \ell\}$                               ▷ Sample blinding factor of non-constant-time inversion
12: $t \leftarrow r \cdot b \mod \ell$                            ▷ Invert using extended binary gcd
13: $s \leftarrow t^{-1} \cdot b \mod \ell$                       ▷ Unblind result of inversion
14: $k'_{f^{-1}} \leftarrow k_{f^{-1}} \cdot s \mod l$            ▷ Multiplicatively blind scalar $k_{f^{-1}}$
15: $k' \leftarrow k'_{f^{-1}} \cdot f \mod l$                    ▷ Multiplicatively unblind scalar $k'_{f^{-1}}$ with $f$
16: Increase($ctr$)
17: $x_P \leftarrow X_P \cdot Z_P^{-1}$                           ▷ Return to affine $x$-coordinate
18: $y_P \leftarrow Y_P \cdot Z_P^{-1}$                           ▷ Return to affine $y$-coordinate
19: $X_1 \leftarrow 1, Z_1 \leftarrow 0$
20: $Z_2 \xleftarrow{\$} \{0, \ldots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$   ▷ Initial randomization of projective representation
21: $k' \leftarrow k' \oplus 2k'$                                 ▷ Precompute condition bits for cswap
22: $a \leftarrow \{0, ..., 2^{253} - 1\}$                        ▷ Sample mask for address-randomization
23: $k' \leftarrow k' \oplus a$                                   ▷ Mask the scalar
24: Increase($ctr$)
25: $(X_1, Z_1, X_2, Z_2) \leftarrow$ cswaprr $(X_1, Z_1, X_2, Z_2, a[252])$   ▷ projective re-randomization merged with cswap based on mask.
26: for $i$ from 252 downto 0 do                                  ▷ scalar multiplication by $k' = k \cdot r^{-1}$
27:     $(X_1, Z_1, X_2, Z_2) \leftarrow$ cswaprr $(X_1, Z_1, X_2, Z_2, k'[i])$   ▷ projective re-randomization merged with cswap based on masked $k$
28:     if $i \geq 1$ then
29:         $(X_1, Z_1, X_2, Z_2) \leftarrow$ ladderstep $(x_P, X_1, Z_1, X_2, Z_2)$
30:         $(X_1, Z_1, X_2, Z_2) \leftarrow$ cswaprr $(X_1, Z_1, X_2, Z_2, a[i-1])$   ▷ projective re-randomization merged with cswap based on mask
31:         Increase($ctr$)
32:     end if
33: end for
34: $y_P \leftarrow$ yrecover($X_1, Z_1, X_2, Z_2, x_P, y_P$)
35: $x_P \leftarrow X_2 \cdot Z_2^{-1}$
36: $X_1 \leftarrow 1, Z_1 \leftarrow 0$
37: $Z_2 \xleftarrow{\$} \{0, \ldots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$   ▷ Again randomize projective representation
38: $a' \leftarrow \{0, ..., 2^{65} - 1\}$                        ▷ Sample additional mask for address-randomization
39: $r \leftarrow r \oplus 2r$                                    ▷ Precompute condition bits for cswap
40: $r \leftarrow r \oplus a'$                                    ▷ Mask the random scalar $r$
41: Increase($ctr$)
42: $(X_1, Z_1, X_2, Z_2) \leftarrow$ cswaprr $(X_1, Z_1, X_2, Z_2, a'[64])$   ▷ projective re-randomization merged with cswap based on mask
43: for $i$ from 64 downto 0 do                                   ▷ scalar multiplication by $r$
44:     $(X_1, Z_1, X_2, Z_2) \leftarrow$ cswaprr $(X_1, Z_1, X_2, Z_2, r[i])$   ▷ projective re-randomization merged with cswap based on masked $r$
45:     if $i \geq 1$ then
46:         $(X_1, Z_1, X_2, Z_2) \leftarrow$ ladderstep $(x_P, X_1, Z_1, X_2, Z_2)$
47:         $(X_1, Z_1, X_2, Z_2) \leftarrow$ cswaprr $(X_1, Z_1, X_2, Z_2, a'[i-1])$   ▷ projective re-randomization merged with cswap based on mask
48:         Increase($ctr$)
49:     end if
50: end for
51: $Y_2 \leftarrow$ yrecover($X_1, Z_1, X_2, Z_2, x_P, y_P$)
52: $(X_2, Y_2, Z_2) \leftarrow$ ecadd($(X_2, Y_2, Z_2), S$)      ▷ Remove point blinding, add in $S = [-k]R$
53: $x_P \leftarrow X_2 \cdot Z_2^{-1}$
54: Increase($ctr$)
55: if Verify($ctr$) then                                         ▷ Detected wrong flow, including iteration count
56:     $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$          ▷ Set output buffer to random bytes
57: end if
58: Update($R, S$)                                                ▷ Perform double-and-add scalar multiplication with the same 8-bit random number on both $R$ and $S$
59: Randomize($k_{f^{-1}}, f$)                                    ▷ Generate new 64-bit random value $f$, securely compute $f^{-1}$ and update $k_{f^{-1}}$
60: Save($R, S, k_{f^{-1}}, f$)
61: return $x_P$
```

and we record traces using the LeCroy WaveRunner 610Zi oscilloscope. For analysis of the traces we use the Inspector software by Riscure [92].

For all the results presented in this section we compile the code using the standard -O2 optimization flag. However, we perform every test also using no optimizations (-O0) as a double check (since using no optimizations often inflate existing leakage). All results are consistent for both flags with respect to detecting leakage.

We perform the leakage detection in the following scenarios:

- The unprotected implementation, without any countermeasures except constant-time implementations of all operations, including field arithmetic and Montgomery ladder;
- The ephemeral implementation, with projective coordinate re-randomization enabled (see Algorithm 2);
- The static implementation as in Algorithm 3, with all countermeasures enabled (projective re-randomization, randomized addressing, point, scalar, and storage blinding);
- A static implementation modified for the profiled-attacks evaluation with scalar and storage blinding disabled. We test this setting with and without the randomized addressing countermeasure to verify resistance to profiled attacks.

A comparison of power profiles of our standard unprotected, ephemeral, and static implementations is presented in Appendix B.

We apply a commonly-used TVLA methodology [14, 45, 58, 105] to our implementations using the aforementioned measurement settings. Following this leakage-evaluation methodology, we use three sets of traces that are equal in size one third of the traces is taken with a fixed input and a fixed scalar (*group* 0), another third with a random input and the fixed scalar (*group* 1), and the last group with the fixed input and a random scalar (*group* 2). If the null hypothesis holds and no leakage is detected then there should be little differences in the Welch's t-test statistics measured between group 0 and 1, and between 0 and 2.

Traditionally, in [58, 105] the TVLA confidence threshold to detect leakage in the t-test statistic was set to 4.5. However, we compute the confidence threshold for our experiments, using the formula from [31] following the approach [86]. The threshold computed this way is more accurate and ensures that false positives are avoided in the leakage assessment. For all our experiments the computed threshold is between 7 and 7.3 and therefore, we assume that peaks above 7 indicate leakage.

In our evaluation we concentrate not only the scalar multiplication, but we also analyze the rest of the trace. Note that we usually expect to see leakage at the beginning of the trace, due to varying input or scalar, and at the end due to varying output.

The leakage is usually detected only around the area that we align on[3]. This is caused by jitter and, especially for the static case, by intended randomizations (e.g., for the inverse operations). Therefore, it might happen that if the trace is aligned at the beginning of trace then the leakage is only detected there, even if leakage is present everywhere. To avoid that we align the traces in multiple locations (usually at the beginning, middle, and end of computational blocks). Due to the space constraints, we report only the meaningful results when leakage is detected or no leakage is confirmed.

In the following subsections we always first analyze the difference between groups 0 and 1. The goal is to evaluate the correlation of the implementation leakage to a fixed or a random input point; such leakage, if present, can be used to mount CPA. Secondly we concentrate on groups 0 and 2. The aim is typically to check whether the private key (i.e, scalar) leaks directly; such leakage might be used to mount a template attack.

[3]We align on a distinctive pattern in the trace. We select such a pattern from the first trace and then we match the pattern to all subsequent traces by shifting them horizontally. For matching quality we use Pearson correlation – we simply choose the shift in the trace with the maximum correlation coefficient.

TVLA of the unprotected implementation. Figure 1 depicts the results of the t-tests for groups 0 and 1. We also aligned the traces at various locations and results were similar. The result for groups 0 and 2 is similar and is presented in the Appendix B. In both cases each group consists of 1000 traces.

The highest peak is reaching 71 for groups 0 and 1, and 65 for groups 0 and 2. Therefore, we conclude that the unprotected implementation leaks significantly.

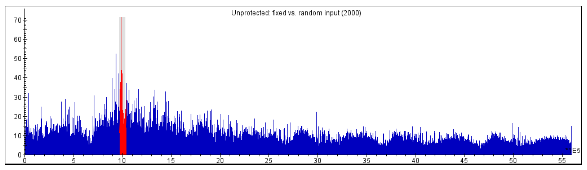

Figure 1: TVLA results for the unprotected implementation: fixed vs random point. The red color marks the alignment.

TVLA of the ephemeral scalar implementation. Similarly to the unprotected implementation we perform the TVLA on the ephemeral implementation, but this time due to the implemented countermeasures, we expect less leakage. Therefore, we also increase the size of each group to 2000 traces.

In Figure 2 we present the results of the t-tests for the ephemeral scalar implementation. The leakage is less significant than for the unprotected case. The highest peak is reaching 15 for groups 0 and 2, but it is to be expected since the ephemeral scalar should be different in every execution. As we see if the same scalar is used multiple times then the leakage can be detected.

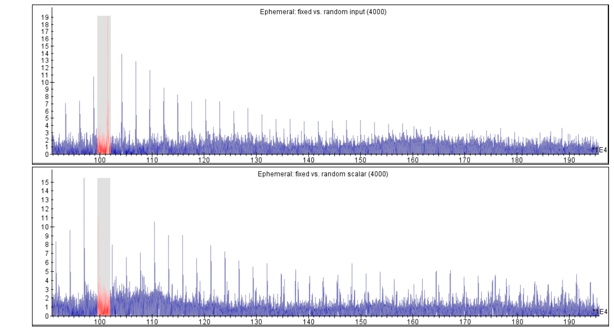

Figure 2: TVLA for the ephemeral impl. (0.9ms-2.0ms): fixed vs random point (top) and fixed vs random scalar (bottom).

We also note that the peak reaches 19 for groups 0 and 1. This might be unexpected since we employ the coordinate projective re-randomization. However, we have realized that the leakage might be present due to the using a non-randomized x coordinate of the input point in the ladder step. The input leakage is not exploitable but it would generate t-test peak (since the input is a parameter to TVLA). Essentially, we encounter a well-know limitation of TVLA [14].

To validate the above hypothesis we have modified the implementation by removing the following line of the ladder step:

`fe25519_mul(b4,b1,&pState->x0)`, where `&pState->x0` corresponds the affine coordinate of the input x_P. Note that this operation leakage is not exploitable since x_P is public and constant per execution and the other parameters to `fe25519_mul`, and therefore the output too, are randomized. The results produced by this implementation are incorrect, but we only use it for the evaluation. The result of t-test has shown that the leakage is not present anymore, because the highest peak does not reach even 4.4; the plot of the t-test values is presented in Appendix B. We can conclude that the ephemeral implementation is protected against DPA and CPA attacks and most likely against the cross-correlation attacks. However, it might be vulnerable to single-trace profiled attacks.

We recommended to use the same key only once, otherwise more attacks might be possible. Furthermore, it may be possible to learn information about the ephemeral key when it is being copied. However, this would be hard since the ephemeral key should be used only once and copying is done in 32-bit chunks[4].

TVLA of the static implementation. First we run t-test for the traces collected from the static scalar multiplication that are aligned at the beginning. For groups 0 and 1 we notice leakage at the beginning of the execution, when the input point is being processed. For groups 0 and 2 we detect no leakage (the highest peak is less than 4.6), due to the scalar being stored blinded. The plot of the t-test values for these traces is presented in Appendix B.

For the static traces there are significant misalignments due to jitter, but also due to the used countermeasures. Therefore, we perform another t-test for which we align the traces at the beginning of the main scalar multiplication and the result is depicted in Figure 3; it is zoomed in around the alignment moment. We can see that the peaks in both experiments are below 4.3, and therefore leakage is not detected. We repeated the experiment aligning at various locations: the middle and the end of the main scalar multiplication, and at the beginning of the additional scalar multiplication. No leakage during both scalar multiplications is detected.

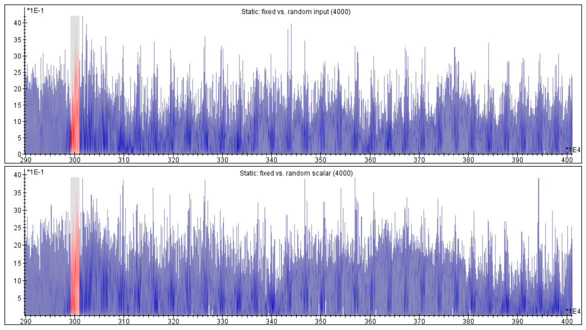

Figure 3: TVLA for the static impl. (2.9ms-4.0ms): fixed vs random point (top) and fixed vs random scalar (bottom).

However, we have detected leakage in both cases just after the scalar multiplication is finished: around 14.5ms. This leakage is caused by computing the final affine output. After the output is computed, the static key is updated and finally the output is sent. We aligned the traces at the key update procedure and we detected no leakage. At the end of the execution we discovered the leakage corresponding to sending the output (17ms). Such output leakage might be used for a sing-trace attack to recover the output point (and effectively the session key). Although this may be possible, this cannot be avoided since the library returns the unmasked output.

In this section we show that the static implementation is resistant to standard attacks like DPA/CPA, cross-correlation, and OTA.

Profiled attacks. The first most relevant profiling attack is a TA targeting the cswap introduced in [82]. Note that the blinded scalar and the additional 64-bit scalar blinding would need to be recovered from a single trace for the attack to succeed.

We verified the resistance of our implementation to this attack by performing a TVLA experiment on an implementation with turned off all scalar randomizations: the scalar and storage blinding. In this setting we checked whether turning on the randomized addressing countermeasure prevents scalar leakage from occurring.

Figure 4 depicts the t-test results for groups 0 and 1 for the traces collected from the scalar multiplication with all scalar blindings and address randomization turned off. The highest t-test peak is reaching 12.5. This leakage and the indicated points of interest might be used to recover the scalar from the cswaprr instructions using the aforementioned TA. For groups 0 and 1, as expected, we do not see any leakage, i.e., the peak is under 4.5; the plot of the t-test values for this case is presented in Appendix B.

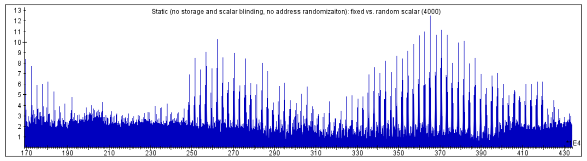

Figure 4: TVLA for static impl. (no blindings & address rand.) aligned at 1.65ms (1.7ms-4.3ms): fixed vs random scalar.

To validate whether the address randomization works we turned on this countermeasure. We have discovered that if all scalar blinding countermeasures are turned off, but the address randomization is turned on, no leakage for the TA [82] is present. The plot of the t-test values for this case is presented in Appendix B.

Although we have discovered that the leakage that can be exploited by “classical” TA is not present, a DL-based attack might still be theoretically possible. However, we are convinced that the absence of “classical” leakage provide enough assurance even against more complex profiled attacks (including DL).

We consciously decided not to protect the ephemeral scalar multiplication with the address randomization for the sake of efficiency. However, it would be easy to add address randomization in the same way as for the static case.

The second relevant profiled attack, which we have not discussed here yet, is against the scalar transfer, as presented in [109]. Observe that the blinded scalar is copied just before the scalar multiplication. We have verified that the scalar data is being copied word by word in the disassembled code. Therefore, we can expect that even a successful profiled attack on key transfer would recover Hamming

[4]Since 32-bit words are being copied then we expect that 32-bit Hamming weight may be leaking. If the key is used once then a profiled attack would require a single trace and it might be infeasible due to measurement noise.

weight of 32-bit chunks of the blinded scalar. Furthermore, the attack phase cannot use multiple traces, since a blinded scalar is stored together with the blinding and these values are updated at the end of every scalar multiplication. Therefore, we conclude that it would be hard to mount a profiled attack along the lines of [109] on the scalar transfer.

6 CONCLUSIONS AND FUTURE WORK

We have described software computing the X25519 key-exchange on the ARM-Cortex M4 microcontroller, This software comes with extensive protections against both side-channel and fault attacks. It is, to the best of our knowledge, the first to claim such protections motivated from a real-world application. We present an extensive side-channel and fault-injection analysis and we also perform in-depth side-channel evaluation that shows strong resistance of our implementation.

We leave a detailed investigation into single-trace complex profiled attacks (in particular, deep learning) for future work. Furthermore, we leave formal proofs of side-channel resistance and higher-order protection including corresponding evaluation for future work.

ACKNOWLEDGEMENTS

Łukasz Chmielewski is supported by European Commission through the ERC Starting Grant 805031 (EPOQUE) of Peter Schwabe.

REFERENCES

[1] Rodrigo Abarzúa, Claudio Valencia, and Julio Lópe z. 2019. Survey for Performance & Security Problems of Passive Side-channel Attacks Countermeasures in ECC. Cryptology ePrint Archive, Report 2019/010. (2019). https://eprint.iacr.org/2019/010.

[2] A. Adamantiadis, S. Josefsson, and M. Baushke. 2020. Secure Shell (SSH) Key Exchange Method Using Curve25519 and Curve448. IETF RFC 8731. (2020). https://tools.ietf.org/html/rfc8731.

[3] Toru Akishita and Tsuyoshi Takagi. 2003. Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In *Information Security (LNCS)*, Colin Boyd and Wenbo Mao (Eds.), Vol. 2851. Springer, 218–233.

[4] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. 2016. Verifying Constant-Time Implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, 53–70. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_almeida.pdf.

[5] Adrian Antipa, Daniel Brown, Alfred Menezes, René Struik, and Scott Vanstone. 2003. Validation of Elliptic Curve Public Keys. In *Public-Key Cryptography – PKC 2003 (LNCS)*, Yvo G. Desmedt (Ed.), Vol. 2567. Springer, 211–223. https://iacr.org/archive/pkc2003/25670211/25670211.pdf.

[6] Arm 2021. Mbed TLS. (2021). https://github.com/ARMmbed/mbedtls (accessed 2021-05-05).

[7] Jean-Philippe Aumasson. 2017. Should Curve25519 keys be validated? (2017). https://research.kudelskisecurity.com/2017/04/25/should-ecdh-keys-be-validated/ (accessed 2020-12-02).

[8] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and Francois-Xavier Standaert. 2019. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *Computer Security – ESORICS 2019 (LNCS)*, Kazue Sako, Steve Schneider, and Peter Y. A. Ryan (Eds.), Vol. 11735. Springer, 300–318.

[9] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. 2015. Verified Proofs of Higher-Order Masking. In *Advances in Cryptology – EUROCRYPT 2015 (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9056. Springer, 457–485.

[10] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, , and Michael Tunstall. 2014. Online Template Attacks. In *Progress in Cryptology – INDOCRYPT 2014 (LNCS)*, Willi Meier and Debdeep Mukhopadhyay (Eds.), Vol. 21–36. Springer, 8885. http://cryptojedi.org/papers/#ota.

[11] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. 2017. Online template attacks. *Journal of Cryptographic Engineering* (2017), 1–16. http://cryptojedi.org/papers/#ota, see also short version [11].

[12] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. 2014. Horizontal Collision Correlation Attack on Elliptic Curves. In *Selected Areas in Cryptography – SAC 2013 (LNCS)*, Tanja Lange, Kristin Lauter, and Petr Lisoněk (Eds.), Vol. 8282. Springer, 553–570.

[13] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. 2013. Sleuth: Automated Verification of Software Power Analysis Countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2013 (LNCS)*, Guido Bertoni and Jean-Sébastien Coron (Eds.), Vol. 8086. Springer, 293–310.

[14] Georg T. Becker, Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Timofei Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. 2013. Test vector leakage assessment (TVLA) methodology in practice. International Cryptographic Module Conference. (2013).

[15] Daniel J. Bernstein. A state-of-the-art Diffie-Hellman function. https://cr.yp.to/ecdh.html (accessed 2021-05-05).

[16] Daniel J. Bernstein. 2006. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography – PKC 2006 (LNCS)*, Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin (Eds.), Vol. 3958. Springer, 207–228. http://cr.yp.to/papers.html#curve25519.

[17] Daniel J. Bernstein. 2014. 25519 naming. Posting to the CFRG mailing list. (2014). https://www.ietf.org/mail-archive/web/cfrg/current/msg04996.html.

[18] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2011. High-speed high-security signatures. In *Cryptographic Hardware and Embedded Systems – CHES 2011 (LNCS)*, Bart Preneel and Tsuyoshi Takagi (Eds.), Vol. 6917. Springer, 124–142. see also full version [19].

[19] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (2012), 77–89. http://cryptojedi.org/papers/#ed25519, see also short version [18].

[20] Ingrid Biehl, Bernd Meyer, and Volker Müller. 2000. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *Advances in Cryptology – CRYPTO 2000 (LNCS)*, Mihir Bellare (Ed.), Vol. 1880. Springer, 131–146.

[21] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. 2006. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In *Fault Diagnosis and Tolerance in Cryptography (LNCS)*, Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert (Eds.), Vol. 4236. Springer, 36–52.

[22] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 1997. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology – Eurocrypt '97 (LNCS)*, Walter Fumy (Ed.), Vol. 1233. Springer, 37–51.

[23] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems – CHES 2004 (LNCS)*, Marc Joye and Jean-Jacques Quisquater (Eds.), Vol. 3156. Springer, 16–29.

[24] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. 2017. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In *Cryptographic Hardware and Embedded Systems – CHES 2017 (LNCS)*, Wieland Fischer and Naofumi Homma (Eds.), Vol. 10529. Springer, 45–68.

[25] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. 2002. Template Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2002 (LNCS)*, Burton S. Kaliski Jr., Cetin K. Koc, and Christof Paar (Eds.), Vol. 2523. Springer, 13–28.

[26] Łukasz Chmielewski, Pedro Maat Costa Massolino, Jo Vliegen, Lejla Batina, and Nele Mentens. 2017. Completing the Complete ECC Formulae with Countermeasures. *Journal of Low Power Electronics and Applications* 7, 1, Article 3 (2017), 13 pages. https://www.mdpi.com/2079-9268/7/1/3.

[27] Mathieu Ciet and Marc Joye. 2005. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Designs, Codes and Cryptography* 36 (2005), 33–43.

[28] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. 2010. Horizontal Correlation Analysis on Exponentiation. In *Information and Communications Security (LNCS)*, Miguel Soriano, Sihan Qing, and Javier López (Eds.), Vol. 6476. Springer, 46–61. http://eprint.iacr.org/2003/237.

[29] Jean-Sébastien Coron. 1999. Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems. In *Cryptographic Hardware and Embedded Systems (LNCS)*, Çetin K. Koç and Christof Paar (Eds.), Vol. 1717. Springer, 292–302.

[30] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. 2013. A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. *Journal of Cryptographic Engineering* 3, 4 (2013), 1–25.

[31] 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 (LNCS)*, Thomas Eisenbarth and Yannick Teglia (Eds.), Vol. 10728. Springer, 105–122.

[32] Thai Duong. 2015. Why not validate Curve25519 public keys could be harmful. (2015). https://vnhacker.blogspot.com/2015/09/

why-not-validating-curve25519-public.html (accessed 2020-12-02).

[33] Hassan Eldib, Chao Wang, and Patrick Schaumont. 2014. SMT-Based Verification of Software Countermeasures against Side-Channel Attacks. In *Tools and Algorithms for the Construction and Analysis of Systems (LNCS)*, Erika Ábrahám and Klaus Havelund (Eds.), Vol. 8413. Springer, 62–77.

[34] Junfeng Fan, Benedikt Gierlichs, and Frederik Vercauteren. 2011. To infinity and beyond: Combined attack on ECC using points of low order. In *Cryptographic Hardware and Embedded Systems – CHES 2011 (LNCS)*, Bart Preneel and Tsuyoshi Takagi (Eds.), Vol. 6917. Springer, 143–159.

[35] 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 *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 76–87.

[36] Junfeng Fan and Ingrid Verbauwhede. 2012. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In *Cryptography and Security: From Theory to Applications (LNCS)*, David Naccache (Ed.), Vol. 6805. Springer, 265–282.

[37] OPC Foundation. 2008. Unified Architecture. (2008). https://opcfoundation.org/about/opc-technologies/opc-ua/ (accessed 2021-05-05).

[38] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. 2008. Fault Attack on elliptic curve with Montgomery ladder implementation. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 92–98. https://www.di.ens.fr/~fouque/pub/fdtc08.pdf.

[39] Pierre-Alain Fouque, Denis Réal, Frédéric Valette, and Mhamed Drissi. 2008. The Carry Leakage on the Randomized Exponent Countermeasure. In *Cryptographic Hardware and Embedded Systems – CHES 2008 (LNCS)*, Elisabeth Oswald and Pankaj Rohatgi (Eds.), Vol. 5154. Springer, 198–213.

[40] Pierre-Alain Fouque and Frederic Valette. 2003. The doubling attack — Why upwards is better than downwards. In *Cryptographic Hardware and Embedded Systems – CHES 2003 (LNCS)*, Colin D. Walter, Çetin K. Koç, and Christof Paar (Eds.), Vol. 2779. Springer, 269–280. www.ssi.gouv.fr/archive/fr/sciences/fichiers/lcr/fova03.pdf.

[41] Hayato Fujii and Diego F. Aranha. 2018. Efficient Curve25519 implementation for ARM microcontrollers. In *Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. Sociedade Brasileira de Computaçcão, 57–64. https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/4142/4071.

[42] Hayato Fujii and Diego F. Aranha. 2019. Curve25519 for the Cortex-M4 and beyond. In *Progress in Cryptology – LATINCRYPT 2017 (LNCS)*, Tanja Lange and Orr Dunkelman (Eds.), Vol. 11368. Springer, 109–127. http://www.cs.haifa.ac.il/~orrd/LC17/paper39.pdf.

[43] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. 2016. ECDH Key-Extraction via Low-Bandwidth Electromagnetic Attacks on PCs. (2016), 219–235 pages. https://web.eecs.umich.edu/~genkin/papers/ecdh.pdf.

[44] Daniel Genkin, Luke Valenta, and Yuval Yarom. 2017. May the fourth be with you: A microarchitectural side channel attack on several real-world applications of Curve25519. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS'17*. ACM, 845–858.

[45] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. 2011. A testing methodology for side-channel resistance validation, NIAT. Workshop record of the NIST Non-Invasive Attack Testing Workshop. (2011). https://csrc.nist.gov/CSRC/media/Events/Non-Invasive-Attack-Testing-Workshop/documents/08_Goodwill.pdf.

[46] Google 2021. BoringSSL. (2021). https://github.com/boringssl/boringssl (accessed 2021-05-05).

[47] Louis Goubin. 2002. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *Public Key Cryptography – PKC 2003 (LNCS)*, Yvo G. Desmedt (Ed.), Vol. 2567. Springer, 199–211.

[48] Björn Haase and Benoît Labrique. 2017. Making Password Authenticated Key Exchange Suitable for Resource-Constrained Industrial Control Devices. In *Cryptographic Hardware and Embedded Systems – CHES 2017 (LNCS)*, Wieland Fischer and Naofumi Homma (Eds.), Vol. 10529. Springer, 346–364.

[49] Björn Haase and Benoît Labrique. 2019. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 2 (2019), 1–48. https://tches.iacr.org/index.php/TCHES/article/view/7384.

[50] Neil Hanley, HeeSeok Kim, and Michael Tunstall. 2015. Exploiting Collisions in Addition Chain-Based Exponentiation Algorithms Using a Single Trace. In *Topics in Cryptology – CT-RSA 2015 (LNCS)*, Kaisa Nyberg (Ed.), Vol. 9048. Springer, 431–448.

[51] Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. 2012. Localized Electromagnetic Analysis of Cryptographic Implementations. In *Topics in Cryptology – CT-RSA 2012 (LNCS)*, Orr Dunkelman (Ed.), Vol. 7178. Springer, 231–244.

[52] Jochen Hoenicke. 2015. Extracting the Private Key from a TREZOR . . . with a 70$ Oscilloscope. (2015). https://jochen-hoenicke.de/crypto/trezor-power-analysis/ (accessed 2012-12-02).

[53] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. 2008. Collision-based power analysis of modular exponentiation using chosen-message pairs. In *Cryptographic Hardware and Embedded Systems – CHES 2008 (LNCS)*, Elisabeth Oswald and Pankaj Rohatgi (Eds.), Vol. 5154. Springer, 15–29. http://www.aoki.ecei.tohoku.ac.jp/crypto/pdf/CHES2008_homma.pdf.

[54] IANIX. Things that use Curve15519. https://ianix.com/pub/curve25519-deployment.html (accessed 2021-05-05).

[55] ISO/IEC 17825 2016. *Information technology – Security techniques – Testing methods for the mitigation of non-invasive attack classes against cryptographic modules*. Standard. International Organization for Standardization, Geneva, CH.

[56] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. 2003. Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In *Cryptographic Hardware and Embedded Systems – CHES 2002 (LNCS)*, Burton S. Kaliski, çetin K. Koç, and Christof Paar (Eds.), Vol. 2523. Springer, 129–143.

[57] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. 2003. A Practical Countermeasure against Address-Bit Differential Power Analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2003 (LNCS)*, Colin D. Walter, Çetin K. Koç, and Christof Paar (Eds.), Vol. 2779. Springer, 382–396.

[58] Josh Jaffe, Pankaj Rohatgi, and Marc Witteman. 2011. *Efficient side-channel testing for public key algorithms: RSA case study*. Technical Report. Cryptography Research Inc. & Riscure. http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/09_Jaffe.pdf.

[59] Jan Jancar, Vladimir Sedlacek, Petr Svenda, and Marek Sys. 2020. Minerva: The curse of ECDSA nonces (Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces). *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 4 (2020), 281–308. https://tches.iacr.org/index.php/TCHES/article/view/8684.

[60] Marc Joye and Sung-Ming Yen. 2003. The Montgomery Powering Ladder. In *Cryptographic Hardware and Embedded Systems – CHES 2002 (LNCS)*, Burton S. Kaliski, çetin K. Koç, and Christof Paar (Eds.), Vol. 2523. Springer, 291–302.

[61] Michael Kasper, Richard Petri, Dirk Feldhusen, Max Gebhardt, Georg Illies, Manfred Lochter, Oliver Stein, Wolfgang Thumserang, and Guntram Wicke. 2016. Minimum Requirements for Evaluating Side-Channel Attack Resistance of Elliptic Curve Implementations. (2016). http://publica.fraunhofer.de/documents/N-487544.html (accessed 2021-05-05).

[62] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. 2019. Make Some Noise. Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 3 (2019), 148–179. https://doi.org/10.13154/tches.v2019.i3.148-179.

[63] Neal Koblitz. 1987. Elliptic curve cryptosystems. *Math. Comp.* 48, 177 (1987), 203–209. http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf.

[64] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology – CRYPTO'96 (LNCS)*, Neal Koblitz (Ed.), Vol. 1109. Springer, 104–113. http://www.cryptography.com/public/pdf/TimingAttacks.pdf.

[65] Paul C. Kocher, Jushua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Advances in Cryptology – CRYPTO '99 (LNCS)*, Michael Wiener (Ed.), Vol. 1666. Springer, 388–397.

[66] Hugo Krawczyk. 2003. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In *Advances in Cryptology – CRYPTO 2003 (LNCS)*, Dan Boneh (Ed.), Vol. 2729. Springer, 400–425. https://webee.technion.ac.il/~hugo/sigma-pdf.pdf.

[67] Tamara Rezk Lesly-Ann Daniel, Sébastien Bardin. 2020. Binsec/Rel: Efficient Relational SymbolicExecution for Constant-Time at Binary-Level. In *2020 IEEE Symposium on Security and Privacy*. 1021–1038. https://people.csail.mit.edu/jgross/personal-website/papers/2019-fiat-crypto-ieee-sp.pdf.

[68] Zhe Liu, Patrick Longa, Geovandro C. C. F. Pereira, Oscar Reparaz, and Hwajeong Seo. 2017. Four$\mathbb{Q}$ on Embedded Devices with Strong Countermeasures Against Side-Channel Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2017 (LNCS)*, Wieland Fischer and Naofumi Homma (Eds.), Vol. 10529. Springer, 665–686. https://eprint.iacr.org/2017/434.

[69] Manfred Lochter, Johannes Merkle, Jörn-Marc Schmidt, and Torsten Schütze. 2015. Requirements for Elliptic Curves for High-Assurance Applications. Workshop record of the NIST Workshop on Elliptic Curve Cryptography Standards. (2015). https://csrc.nist.gov/csrc/media/events/workshop-on-elliptic-curve-cryptography-standards/documents/papers/session4-merkle-johannes.pdf.

[70] Antoine Loiseau, Maxime Lecomte, and Jacques J. A. Fournier. 2020. Template Attacks against ECC: practical implementation against Curve25519. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 13–22.

[71] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. 2016. Breaking Cryptographic Implementations Using Deep Learning Techniques. In *Security, Privacy, and Applied Cryptography Engineering (LNCS)*, Claude Carlet, M. Anwar

Hasan, and Vishal Saraswat (Eds.), Vol. 10076. Springer, 3–26.

[72] Moxie Marlinspike. 2013. Simplifying OTR deniability. (2013). https://signal.org/blog/simplifying-otr-deniability/ (accessed 2021-05-05).

[73] Nick Mathewson. 2016. Cryptographic directions in Tor. Presentation at Real-World Crypto 2016. (2016). https://rwc.iacr.org/2016/Slides/nickm-rwc-presentation.pdf.

[74] David McCann, Elisabeth Oswald, and Carolyn Whitnall. 2017. Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages. In *Proceedings of the 26th USENIX Security Symposium*. USENIX, 199–216. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-mccann.pdf.

[75] Marcel Medwed and Elisabeth Oswald. 2009. Template Attacks on ECDSA. In *Information Security Applications (LNCS)*, Kyo-Il Chung, Kiwook Sohn, and Moti Yung (Eds.), Vol. 5379. Springer, 14–27.

[76] Alfred J. Menezes, Tatsuaki Okamoto, and Scott Vanstone. 1993. Reducing elliptic curve logarithms to logarithms in a finite field. *Transactions on Information Theory* 39, 5 (1993), 1639–1646.

[77] Victor S. Miller. 1986. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology – CRYPTO '85: Proceedings (LNCS)*, Hugh C. Williams (Ed.), Vol. 218. Springer, 417–426.

[78] Peter L. Montgomery. 1987. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comp.* 48, 177 (1987), 243–264. http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866113-7/S0025-5718-1987-0866113-7.pdf.

[79] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. 2012. Compiler Assisted Masking. In *Cryptographic Hardware and Embedded Systems – CHES 2012 (LNCS)*, Emmanuel Prouff and Patrick Schaumont (Eds.), Vol. 7428. Springer, 58–75.

[80] David Naccache, Nigel P. Smart, and Jacques Stern. 2004. Projective Coordinates Leak. In *Advances in Cryptology – EUROCRYPT 2004 (LNCS)*, Christian Cachin and Jan Camenisch (Eds.), Vol. 3027. Springer, 257–267. https://www.iacr.org/archive/eurocrypt2004/30270258/projective.pdf.

[81] Erick Nascimento and Łukasz Chmielewski. 2017. Applying Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations. In *Smart Card Research and Advanced Applications (LNCS)*, Thomas Eisenbarth and Yannick Teglia (Eds.), Vol. 10728. Springer, 213–231.

[82] Erick Nascimento, Łukasz Chmielewski, David Oswald, and Peter Schwabe. 2017. Attacking embedded ECC implementations through cmov side channels. In *Selected Areas in Cryptology – SAC 2016 (LNCS)*, Roberto Avanzi and Howard Heys (Eds.), Vol. 10532. Springer, 99–119. https://cryptojedi.org/papers/#cmovsca.

[83] Y. Nir, S. Josefsson, and M. Pegourie-Gonnard. 2018. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier. IETF RFC 8422. (2018). https://tools.ietf.org/html/rfc8422.

[84] NXP. 2015. SmartMX2 P40 family P40C012/040/072 Secure smart card controller. (2015). https://www.nxp.com/docs/en/data-sheet/P40C040_C072_SMX2_FAM_SDS.pdf.

[85] David Oswald and Christof Paar. 2011. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In *Cryptographic Hardware and Embedded Systems – CHES 2011 (LNCS)*, Bart Preneel and Tsuyoshi Takagi (Eds.), Vol. 6917. Springer, 207–222.

[86] Louiza Papachristodoulou, Apostolos P. Fournaris, Kostas Papagianopoulos, and Lejla Batina. 2019. Practical Evaluation of Protected Residue Number System Scalar Multiplication. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 1 (2019), 259–282. https://doi.org/10.13154/tches.v2019.i1.259-282.

[87] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek. 2021. Keep it Unsupervised: Horizontal Attacks Meet Deep Learning. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 1 (2021), 343–372. https://doi.org/10.46586/tches.v2021.i1.343-372.

[88] John M. Pollard. 1978. Monte Carlo methods for index computation (mod p). *Math. Comp.* 32, 143 (1978), 918–924. http://www.ams.org/journals/mcom/1978-32-143/S0025-5718-1978-0491431-9/S0025-5718-1978-0491431-9.pdf.

[89] Emmanuel Prouff and Matthieu Rivain. 2013. Masking against Side-Channel Attacks: A Formal Security Proof. In *Advances in Cryptology – EUROCRYPT 2013*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 142–159.

[90] Joost Renes, Craig Costello, and Lejla Batina. 2016. Complete Addition Formulas for Prime Order Elliptic Curves. In *Advances in Cryptology - EUROCRYPT 2016 (LNCS)*, Marc Fischlin and Jean-Sébastien Coron (Eds.), Vol. 9665. Springer, 403–428.

[91] Riscure. 2018. Current Probe. Security Test Tool for Embedded Devices. (2018). https://www.riscure.com/product/current-probe/ (accessed 2021-05-05).

[92] Riscure. 2021. Side Channel Analysis Security Tools. (2021). https://www.riscure.com/security-tools/inspector-sca/.

[93] Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. 2018. Breaking Ed25519 in WolfSSL. In *Topics in Cryptology – CT-RSA 2018 (LNCS)*, Nigel P. Smart (Ed.), Vol. 10808. Springer, 1–20. https://eprint.iacr.org/2017/985.

[94] Fabrizio De Santis and Georg Sigl. 2016. Towards Side-Channel Protected X25519 on ARM Cortex-M4 Processors. (2016), XXX pages. http://ccccspeed.win.tue.nl/papers/SPEED-B_Final.pdf

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

[96] Pascal Sasdrich and Tim Güneysu. 2017. Cryptography for Next Generation TLS: Implementing the RFC 7748 Elliptic Curve448 Cryptosystem in Hardware. In *DAC '17: Proceedings of the 54th Annual Design Automation Conference 2017*. IEEE, 1–6.

[97] Peter Schwabe and Ko Stoffelen. 2017. All the AES you need on Cortex-M3 and M4. In *Selected Areas in Cryptology – SAC 2016 (Lecture Notes in Computer Science)*, Roberto Avanzi and Howard Heys (Eds.), Vol. 10532. Springer-Verlag Berlin Heidelberg, 180–194. Document ID: 9fc0b970660e40c264e50ca389dacd49, https://cryptojedi.org/papers/#aesarm.

[98] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. 2022 (to appear). Rosita: Towards automatic elimination of power-analysis leakage in ciphers. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 17. https://www.ndss-symposium.org/ndss-paper/rosita-towards-automatic-elimination-of-power-analysis-leakage-in-ciphers/.

[99] Nigel P. Smart. 1999. The Discrete Logarithm Problem on Elliptic Curves of Trace One. *Journal of Cryptology* 12, 3 (1999), 193–196.

[100] STMicroelectronics. 2018. *Reference manual – STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm-based 32-bit MCUs*. Technical Report RM0090 Rev 17. STMicroelectronics. https://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf.

[101] Open Whisper Systems. Signal Protocol. https://signal.org/ (accessed 2021-05-05).

[102] Thomas Pornin 2018. BearSSL. (2018). https://bearssl.org/ (accessed 2021-05-05).

[103] Niek Timmers, Albert Spruyt, and Marc Witteman. 2016. Controlling PC on ARM Using Fault Injection. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC*. IEEE, 25–35.

[104] Trevor Perrin 2018. The Noise Protocol Framework (revision 34). (2018). https://noiseprotocol.org/noise.pdf.

[105] Michael Tunstall and Gilbert Goodwill. 2016. Applying TVLA to Public Key Cryptographic Algorithms. Cryptology ePrint Archive, Report 2016/513. (2016). http://eprint.iacr.org/2016/513.

[106] Paul C. van Oorschot and Michael J. Wiener. 1999. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology* 12, 1 (1999), 1–28. http://www.scs.carleton.ca/~paulv/papers/JoC97.pdf.

[107] Colin D. Walter. 2001. Sliding Windows Succumbs to Big Mac Attack. In *Cryptographic Hardware and Embedded Systems – CHES 2001 (LNCS)*, Çetin K. Koç, David Naccache, and Christof Paar (Eds.), Vol. 2162. Springer, 286–299.

[108] Leo Weissbart, Lukasz Chmielewski, Stjepan Picek, and Lejla Batina. 2020. Systematic Side-Channel Analysis of Curve25519 with Machine Learning. *Journal of Hardware and Systems Security.* 4 (2020), 314–328. https://doi.org/10.1007/s41635-020-00106-w.

[109] Léo Weissbart, Stjepan Picek, and Lejla Batina. 2019. One trace is all it takes: Machine Learning-based Side-channel Attack on EdDSA. In *Security, Privacy, and Applied Cryptography Engineering (LNCS)*, Shivam Bhasin, Avi Mendelson, and Mridul Nandi (Eds.), Vol. 11947. Springer, 86–105.

[110] Carolyn Whitnall and Elisabeth Oswald. 2019. A Critical Analysis of ISO 17825 ('Testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In *Advances in Cryptology – ASIACRYPT 2019 (LNCS)*, Steven D. Galbraith and Shiho Moriai (Eds.), Vol. 11923. Springer, 256–284.

[111] Marc Witteman. 2018. *Secure Application Programming in the presence of Side Channel Attacks*. Technical Report. Riscure. https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf.

[112] Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. 2011. Defeating RSA Multiply-Always and Message Blinding Countermeasures. In *Topics in Cryptology – CT-RSA 2011 (LNCS)*, Aggelos Kiayias (Ed.), Vol. 6558. Springer, 77–88. https://www.riscure.com/benzine/documents/rsacc_ctrsa_final.pdf.

[113] wolfSSL Inc. 2021. The wolSSL crypto library as accessed at april 24th, 2021). (2021). https://github.com/wolfSSL/wolfssl/tree/master/wolfcrypt.

[114] Yan Yan and Elisabeth Oswald. 2019. Examining the Practical Side Channel Resilience of ARX-boxes. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*. ACM, 373–379. https://eprint.iacr.org/2019/335.

[115] Sung-Ming Yen and Marc Joye. 2000. Checking before output may not be enough against fault-based cryptanalysis. *Transactions on Computers* 49, 9 (2000), 967–970.

[116] Yuanyuan Zhou and François-Xavier Standaert. 2019. Simplified Single-Trace Side-Channel Attacks on Elliptic Curve Scalar Multiplication using Fully Convolutional Networks. Proceedings of the 40th WIC Symposium on Information Theory in the Benelux. (2019). https://perso.uclouvain.be/fstandae/PUBLIS/219.pdf.

A PERFORMANCE COMPARISON

Table 2: Performance Evaluation (using -O2 optimizations).

Library	CT	SCA	Clock cycles:
wolfSSL [113] size:	yes	no	45 930 947
wolfSSL [113] speed:	yes	no	1 974 047
bearSSL i31 [102]:	yes	no	2 576 639
boringSSL[46]: fixed base	yes	no	1 591 407
boringSSL[46]: var. base	yes	no	2 516 476
ARM MBed TLS [6]	no	no	6 438 233
This work			
unprotected[49]	yes	no	683 061
ephemeral DH	yes	yes	930 915
static DH	yes	yes	2 316 986

For the purpose of comparison we have also benchmarked our protected implementation against several commonly used cryptographic libraries. We chose the libraries for their wide-spread use (boringSSL) or because they claimed to be targeting smaller embedded systems (bearSSL, wolfSSL, ARM MBed TLS). We included both, publicly available commercial libraries and open source implementations.

The cycle count results and the essential security features are summarized in Table 2. Cycle counts were obtained by using the same ARM Cortex-M4 and parameters as used in our work.

A.1 Security properties

None of the other benchmarked implementations exceed the protection level of conventional constant-time execution. According to our assessment, the ARM Mbed TLS library even fails to provide constant execution timing as detailed below.

Regarding the security properties, boringSSL and bearSSL roughly should be comparable to our unprotected baseline algorithm, while ARM MBed TLS does not even reach that level.

The fixed-basepoint algorithm from boringSSL uses arithmetics on the Edwards curve and uses large precomputed tables, a strategy which we do not believe suitable for smaller embedded targets due to code size limitations. All other implementations use a Montgomery ladder strategy on the Montgomery curve in projective coordinates. The implementations essentially differ in the way field elements are represented and reduction is carried out and how aggressively optimization for code size and speed are carried out. For instance, the arithmetic of boringSSL and the speed-optimized strategy for wolfSSL represent field elements by using a 10-limb representation using 10 32-bit words, where field multiplication and reduction is merged together. Instead bearSSL uses a 9-limb representation of 9 words of 31 bits each, resulting in somewhat less memory consumption.

None of, wolfSSL, bearSSL and boringSSL include any SCA countermeasures. ARM Mbed TLS on the other hand integrates projective randomization also for public input points. As ARM MBed TLS uses an early-abort multiplication strategy that does not execute in constant time we presume that this projective randomization was considered to provide a mitigation for fending off simple timing-based SPA attacks. Unfortunately, as input points are not validated in ARM MBed TLS, this mitigation strategy does not actually work here. Simple timing attacks using the strategy of [44] becomes feasible if the adversary inserts a low order point as multiplications with zero could be distinguished due to the early-abort multiplication strategy. As a result we conclude that the projective randomization substep used in ARM MBed TLS adds additional computational overhead (to the already slow implementation) without actually providing a proper mitigation against timing-based attacks.

We have communicated our results and concerns regarding the timing vulnerability of ARM Mbed TLS to the developers of [6] and suggested different strategies for mitigating or resolving the problem. They acknowledged the issue, and are working on implementing countermeasures.

A.2 Speed comparison

The unprotected speed-optimized implementations of boringSSL, wolfSSL and bearSSL result in cycle counts in between $1\,974k$ and $2,576k$, roughly corresponding to the cycle count of $2\,316k$ that we obtain for our protected static implementation. Our protected ephemeral implementation still outperforms any of these off-the-shelf libraries while still providing stronger SCA and FI protections.

B SUPPLEMENTARY MATERIAL FOR SIDE-CHANNEL EVALUATION

Power profiles of all three implementations are presented in Figure 5. As mentioned in Section 5.1, the ephemeral implementation is only slightly slower than the unprotected one. In both cases, as marked in red, the implementations consists mainly of a scalar multiplication. In the static case, the trace consist of initial randomizations, two scalar multiplications (marked red), and the final update of the static key and points.

Figure 6 depicts the results of the t-tests for groups 0 and 2 (fixed versus random scalar) for the unprotected implementation.

The t-test result for the modified ephemeral implementation for groups 0 and 1 (fixed vs random point) is presented in Figure 7. As we can see the leakage is not present anymore, because the highest peak does not reach 4.7. We perform also TVLA for groups 0 and 2 and in this case the leakage was still present, as expected.

Figure 8 depicts the results of t-test for the traces collected for the static implementation; these traces are aligned at the beginning.

Figure 9 presents detected leakage after the static scalar multiplication is finished — around 14.5ms. This leakage is caused by computing the affine output.

Figure 10 presents TVLA results for groups 0 and 1 (fixed vs. random point) for the static scalar implementation with all scalar blindings and address randomization turned off.

Figure 11 shows TVLA results for groups 0 and 2 for the static implementation without scalar blindings, but with the address randomization turned on. For groups 0 and 1 we also do not detect leakage.

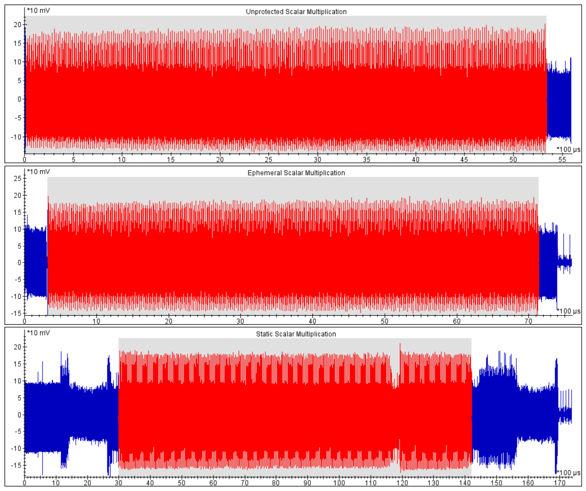

Figure 5: Power profiles of unprotected (top), ephemeral (middle), and static (bottom) implementations with scalar multiplications marked with red.

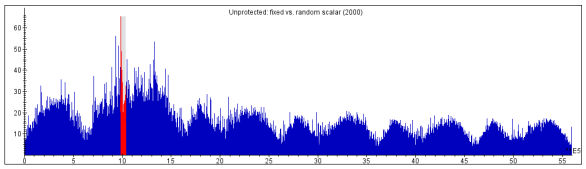

Figure 6: Unprotected impl. TVLA: fixed vs random scalar.

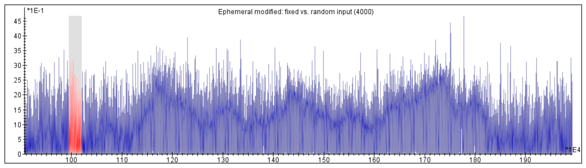

Figure 7: TVLA results for the modified ephemeral implementation (0.9ms-2.0ms): fixed vs random point.

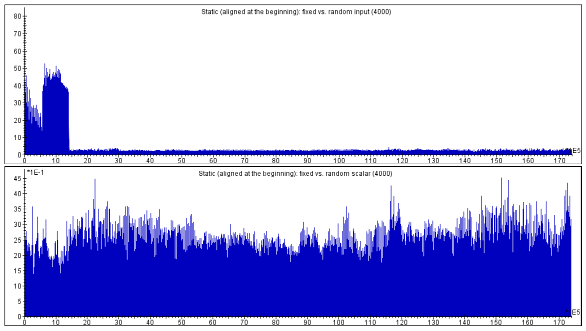

Figure 8: TVLA results for the static implementation: fixed vs random point (top) and fixed vs random scalar (bottom).

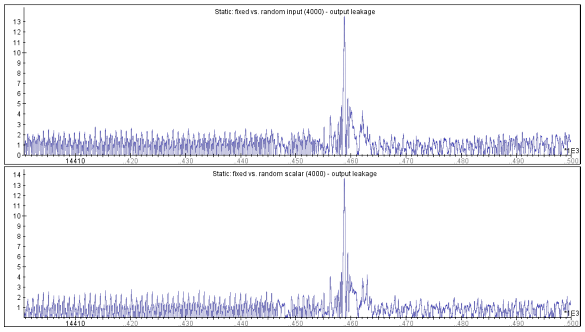

Figure 9: TVLA results for the static scalar implementation (aligned at 13.0ms and zoomed at 14.4ms-14.5ms): fixed vs random point (top) and fixed vs random scalar (bottom).

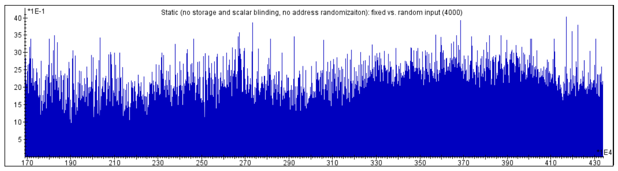

Figure 10: TVLA results for the static implementation with blindings and address randomization turned off (aligned at 1.65ms and zoomed at 1.7ms-4.3ms): fixed vs random point.

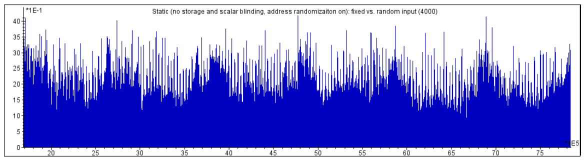

Figure 11: TVLA results for the static impl. with blindings turned off and the address rand. turned on (aligned at 1.65ms and zoomed at 1.7ms - 7.8ms): fixed vs random scalar.