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 the 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 as efficient as widely-deployed ECC cryptographic libraries, which offer much less protection.**

***Index Terms*—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 [79], [104], 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 [66] and Miller [81]. The best known attack against ECC is still the Pollard rho algorithm [92] and its parallel version by van Oorschot and Wiener [111].

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], [67], [68], 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 [68], 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 [27]. Here the adversary first uses a device (for instance, a copy of the attacked device) that is under their 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 [62], or the power-analysis attack recovering the secret key from a TREZOR hardware bitcoin wallet [55]. 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], [32], [38], [39].

We also will 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. We admit that most individual countermeasures used by us have been proposed before – the contribution is to carefully analyze, combine, and implement these. For example, as the Montgomery ladder operates on x-coordinates only, point blinding is not straightforward; also multiplicative scalar blinding is non-standard but more efficient because of the special shape of the group order.

Complete, specific, additive, and public. The survey by Fan and Verbauwhede from 2013 [39] concludes that a side-channel-protected implementation of ECC needs to satisfy three properties:

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.

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

There are implementations of ECC that claim to fulfill the three properties introduced in [39]. 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"* [88]. 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 implementations that never claimed protection against such attacks; see [46], [98], for example.

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 do not 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 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 hefty price to be paid, note that it is rather modest in comparison to securing AES on ARM architectures. In particular, Schwabe and Stoffelen [102] showed 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 show in Subsection 5.1 that even our most protected implementation is at least as efficient as widely-deployed ECC cryptographic libraries, which provide much less SCA protections. The reason for the exceptional performance of our implementations is careful optimizing on the assembly level. We identify synergies between SCA protections and performance, for example in carefully tuning register allocation to minimize memory access.

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 [70] (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], [36], [83]) and we believe that it is a very important direction of research to extend such results also to asymmetric cryptography. In particular, we consider proving that our static X25519 implementation is secure against first-order DPA attacks to be future work, as it would require significant advances of the currently available tools for formal verification.

Moreover, for many attacks that 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 [51] 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.

It is also worth noting that in many consumer applications, commercial pressure pushes designers to use conventional microcontrollers for cryptographic operations instead of dedicated smart-card circuitry.

Another aspect which should be considered is that customization of typical smart-card chipsets is often impossible. For high-volume applications specialized software, e.g., 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. Conversely, the common "turn-key" smart-card products 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 aforementioned application scenario, which cannot afford specialized hardware[2]. Clearly this approach 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.

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

Finally, we choose Cortex-M4 as our platform due to its popularity, for example, it is a software platform of choice for recent NIST competitions. Moreover, we believe that our implementations can be relatively easily adjusted for other ARM architectures (for example, the Cortex-M33 and Cortex-M4 instructions are very similar).

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 [87], SSH [2], the Noise Framework [109, Sec. 12], the Signal protocol [106], and Tor [76]. For a more extensive list see [57]. Moreover, X25519 is now also considered as primitive for the OPC/UA protocol family [40] 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 [72, 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 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 SCA context [47]. As one way to alleviate these subgroup concerns it is often suggested to validate inputs [5], [7], [35]; 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, e.g., Four$\mathbb{Q}$ on the ARM Cortex-M4 microcontroller [71], or the same curve but without such a comprehensive side-channel protection [45], [99].

A range of high-speed implementations [71] are proposed for the Four$\mathbb{Q}$ elliptic curve of Costello and Longa on various embedded devices, including scalar multiplication, ECDH key exchange, and digital signatures. Many countermeasures to thwart various types of SCA are added. 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 side-channel resistance, as expected, and 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 side-channel attacks (namely, SPA and DPA).

Fujii and Aranha [45] present an X25519 implementation that is protected against timing attacks by constant-time execution: 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 [100], [101]. In both cases, the Montgomery ladder was deployed to provide a basic protection against timing and 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

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.

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 FPGA implementation of the complete formulas for Weierstrass curves by Renes, Costello, and Batina [94] has been presented by Chmielewski, Costa Massolino, Vliegen, Batina, and Mentens [28], 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.

Software availability. We place all software described in this paper into the public domain. It is available from an anonymous 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 and on the ARM Cortex-M4 microcontroller. We also introduce our attacker model.

2.1. X25519 key-exchange

In the elliptic-curve Diffie-Hellman key-exchange, each party generates a private and public key pair and distributes the public key. Then the parties can compute a shared key offline using scalar multiplication and the private scalar. The shared secret can be used, for instance, as the session key for a symmetric cipher.

The X25519 elliptic-curve key-exchange is based on arithmetic on the elliptic curve in Montgomery form [82]:

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

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 of shared-key computation 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 smult routine first sets the most significant bit of x_P to zero (ensuring that $x_P \in \{0, \dots, 2^{255} - 1\}$ 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 $k \in 8 \cdot \{2^{251}, \dots, 2^{252} - 1\}$). This operation on bits of the input k is often referred to as "clamping"; in our pseudocode we denote it by clamp. Subsequently, smult 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 smult operation is commonly implemented using the Montgomery ladder [82] 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: $k \in \{0, \dots, 2^{255} - 1\}$ and the $x$-coord. $x_P$ of a 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") routine swaps the first two inputs iff the last input is 1. The ladderstep computes $(X_{[2]P}, Z_{[2]P}, X_{P+Q}, Z_{P+Q})$ on input coordinates $x_{Q-P}, X_P, Z_P, X_Q, Z_Q$. This 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 $x_B = [9, 0, \dots, 0]$ 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 [105]. The most relevant features of this platform for our software are the following:

- 16 general-purpose 32-bit registers, out of which 14 are freely usable (one is used as instruction pointer and one as stack pointer); and
- 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 [77], [103], [119] indicate that specific properties of the internal pipeline architecture are highly relevant for the amount of generated side-channel leakage. 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 [119] 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 [105]. 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-uniform random values 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 [44], [45]. The speed-record within the scientific literature is currently held by an implementation by Haase and Labrique [52]. 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 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 [93]), 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 [58], [115].

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, [108].

Restricting instruction skips to "short" blocks is motivated by the fact that in practice single fault 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 [69] or in combination with static key exchanges in, for example, 3DH [75], it is a critical building block to achieve forward secrecy.

The goal of SCA and FI attackers against the 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 fault injection during both key generation and shared-key computation (and scalar multiplication is the main part of those) 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, which we only consider for the static key-exchange in Section 4.

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

We list the most relevant passive attacks in Subsection 3.1 and the most relevant active attacks in Subsection 3.2, for the ephemeral setting. Due to space constraints we do not list all possible attacks, but only focus on the most relevant ones; for a complete list of attacks see the surveys [39], [64]. Finally, in Subsection 3.3 we explain our implementation and how it protects against the listed attacks.

3.1. Relevant passive attacks

SPA. Simple power analysis (SPA) [68] is one of the most common forms of SCA. 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. Single-trace attacks beyond SPA are often called *horizontal* attacks. These attacks trace back to the Big-Mac attack [112], and were applied against ECC implementations [12], [30], [53]. The main idea of horizontal collision is to use key-dependent collisions of the same values across multiple iterations of the scalar multiplication loop. Consider, e.g., 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 [112] and ECC [53].

Horizontal correlation attacks. Horizontal correlation attacks [30] are 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., coordinate or point randomization, prevent this horizontal attack.

Template attacks. Template attacks (TAs) [27] require a profiling stage in which the adversary estimates the probability distribution of the leaked information to use information present in each sample. Traditionally, TA is considered 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.

Medwed and Oswald [78] introduced a TA targeting the internal state of the ECC ladder after a few scalar bits are processed. In our context if the single-trace TA is successful then the few bits are 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 [86]. The cmov instructions are 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 targets key-transfer [89]. 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 trace.

Deep-learning attacks. A recent class of profiled attacks, similar to TAs, are deep-learning (DL) side-channel attacks [25], [65], [74]. 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 several articles [113], [114], [121]. Deep learning was also used to attack ECC in the unsupervised setting [91].

Online template attacks. Online template attacks (OTA) use horizontal techniques to exploit the fact that an internal state of scalar multiplication depends only on the scalar and the known input [11], [34], [97] or output [24]. Advanced types of those attacks need only one leakage trace and can defeat implementations protected with scalar blinding or splitting. OTA traces back and extends the doubling and collision attacks [43], [56].

If an attacker has only access to the affine, and not projective, input and output points then since OTA depends on predictability of the internal state, it is thwarted by randomizations, like point blinding, projective coordinate randomization [31], or coordinate re-randomization [86].

Horizontal cmov attacks. Horizontal cmov attacks [85] are similar to the aforementioned single-trace TA [86], but they are unsupervised. They combine a heuristic approach based on clustering with multiple-trace 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 the profiled approaches.

3.2. Relevant active attacks

Weak curve attacks. The first weak-curve attack [20] on ECC made use of invalid points. The key observation is that a_6 in the definition of a secure curve E is not used in the addition. Hence, the addition formula for the curve E generates correct results for any curve E' that differs from E only in a_6: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6'$. Thus, the attacker can provide a point P on an insecure curve E' and solve ECDLP on E' to find out the scalar.

Moving a scalar multiplication from a strong curve to a weak curve often requires fault injection. With the help of faults, the attacker can employ invalid points [20], invalid curves [29], and twist curves [41] 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 aims to shorten the scalar. The attacker uses FI to stop the copy loop of the scalar before it is 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

We extend the approach Algorithm 1 with SCA and FI protections. 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 a number of countermeasures, including re-randomizing the projective representation in the cswaprr procedure for the SCA resistance and a flow-counter to improve the FI resistance.

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 order to protect against FI, e.g., loop-abort attacks, we employ a flow-counter countermeasure [116]. 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.

We need to emphasize that we make an efficiency vs. security trade-off for the ephemeral implementation: namely, we protect it less than the static one. In particular, we do not employ the address randomization, storage and scalar blinding (for details, see Section 4). However, note that it would be easy to add some of those countermeasures based on the static implementation. Alternatively, for the best protection the static routine can be used in the ephemeral scenario at the cost of slowing down the computation by a factor of ≈ 2.5 (for exact numbers, see Table 1).

Below we explain why our implementation is resistant to the passive and active attacks listed above.

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, [68].

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 [84] 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.

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 [86], for example) and single-trace TA or DL targeting key loading. The ephemeral implementation does not explicitly protect against such attacks, as shown by the evaluation in Section 5.2. However, the static implementation adds additional protections; 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. We employ a flow-counter countermeasure [116] against the loop-abort attacks.

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. To prevent an attacker from directly influencing the values in the output buffer, we set its content 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. An advanced attacker can try to combine SCA and FI attacks, as presented in [37] for example. 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

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: $k \leftarrow \mathsf{clamp}(k)$
4: $k \leftarrow k/8$                                           ▷ Divide scalar $k$ by 8 to account for initial 3 doublings
5: Increase($ctr$)
6: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(x_p, 1)$
7: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(X_p, Z_P)$
8: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(X_p, Z_P)$        ▷ 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:     $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, k[i])$   ▷ projective re-randomization merged with cswap
20:     $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}\ (x_P, X_1, Z_1, X_2, Z_2)$
21:     Increase($ctr$)
22: end for
23: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ ((X_1, Z_1, X_2, Z_2), k[0])$
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 $x_P$
```

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 such a combined attacker hard: skipping over *one* re-randomization step would only help in recovering one scalar bit 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 ephemeral 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 static masks. The scalar and these masks need to be generated by the user and then stored in the library. Then 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 protecting the private key in memory out of scope.

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

4.1. Relevant passive attacks

First-order and higher-order DPA attacks. With the first publication introducing DPA [68] 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 the attacker 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.

There also exist higher-order multivariate DPA [26], [80], which can combine multiple samples coming from many traces to derive side-channel information.

Cross-correlation attacks. Cross-correlation exploits collisions in subsequent additions and doubling of a scalar multiplication algorithm using many traces [117]. 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 [50]. 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 attacks. Carry-based attacks [42] do 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. Let us consider scalar blinding $k' = k + r\#E$, where k is the scalar, r is the blinding and $\#E$ is the curve order. This blinding is normally implemented with repeated 8-bit additions on 8-bit processors. We denote k_i and $r_i\#E$ as 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 the value of 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 [31] explores the link between the register address and the key and it was successfully applied to ECC [59]. 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 leakage used by address-bit DPA can be also exploited using a TA. Observe that when attacking a single trace, this attack is essentially equivalent to the TA targeting the cswap [86] (see Subsection 3.1), even if the leakage is not coming from the address but from the cswap logic. Therefore, when we talk about the address leakage, we also include to the cswap leakage in this context.

4.2. Relevant active attacks.

Safe-error attacks. The concept of safe-error was introduced by Yen and Joye [63], [120]. Two types of safe-errors 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 tries to induce temporary faults during the execution of the dummy operation. If the result is unaffected then it means that a dummy operation was affected and in case of different result, 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. These attacks were first applied to RSA [63], but they can also be applied 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 scalar bits. 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. We implement all protections from Algorithm 2 including re-randomization of the projective representation using cswaprr and the control-flow counter. Additionally, to prevent key-transfer attacks, 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 $S = [-k]R$. All these blindings f, R, S and the blinded scalar $k_{f^{-1}}$ are always securely re-randomized at the end of scalar multiplication. They should also be stored securely as they can be used to recover the private scalar. 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 we mark these values as secure input in Algorithm 3.

To further improve security, we generate a random 64-bit value r and we blind the scalar with its inverse r^{-1} at the beginning of each scalar multiplication. Then we remove the blinding by performing an additional scalar multiplication by r. This way scalar multiplication does not depend on stored values but it always freshly randomized.

During scalar blinding operations we compute inverses using the extended Euclidean algorithm. We protect these computations by additional multiplicative blinding.

To limit address leakage we implement the address-randomization technique [54], [60]. 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 can thwart cswap-like leakage as shown in [54].

Below we explain why our static implementation is resistant to the passive and active attacks listed above.

First-order and higher-order DPA attacks. To prevent DPA, Coron [31] 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. Note that since higher-order DPA employs multiple traces, these type of the attacks are also stopped by the Coron's countermeasures.

Cross-correlation attacks are essentially reduced to the horizontal correlation due to the usage of scalar blinding. Horizontal correlation is not possible due to employment of the projective re-randomization.

Special-point attacks are prevented by the initial point blinding with R: $S = [-k]R$. This ensures that the point used in each ladder iteration is independent from the input.

Carry-based attacks are thwarted by the usage of the storage scalar blinding. Essentially the scalar is re-blinded

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: $r \xleftarrow{\$} \{1, ..., 2^{64} - 1\}$    ▷ Sample 64-bit non-zero random value for scalar blinding
11: $b \xleftarrow{\$} \{0, ..., \ell\}$    ▷ 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: $a \xleftarrow{\$} \{0, ..., 2^{253} - 1\}$    ▷ 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: $a' \xleftarrow{\$} \{0, ..., 2^{65} - 1\}$    ▷ 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 $x_P$
```

after every usage and therefore, no single guess about k or k_i can be made between multiple blinding computations.

Address-bit DPA is not possible since the 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. Thus, these attacks are much harder to defeat. To thwart them we implement the address-randomization [54], [60] and we evaluate its effectiveness in Section 5.2.

Another effective countermeasure against these attacks was implemented for Curve25519 `cswap` [73]. However, the proposed `cswap` implementation does not consider the risk of correlation between memory loads and stores of the unchanged sub-words, before and after swapping. We choose the more conservative option of storing conditionally swapped operands only after having them projectively re-randomized first. Therefore, we avoid 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

In this section we present benchmark results of our implementations in Subsection 5.1 and our side-channel evaluation results in Subsection 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 [52]. We use the gcc compiler, version 10.2.1 20201103 (release), GNU Arm Embedded Toolchain 10-2020-q4-major with the `-O2` optimization flag. The above details are important for performance numbers because while all low-level arithmetic functions are implemented in assembly, the high-level code is still in C. Compiler optimizations influencing ordering of instructions (e.g., unrolling or inlining) might influence FI resistance. 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 single-trace profiled attacks, 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.

Comparison. As expected, our countermeasures significantly reduce the speed. Still we believe our protected implementations to be highly competitive. For the purpose of comparison we have benchmarked several commonly used speed-optimized versions of unprotected off-the-shelf

TABLE 1. PERFORMANCE EVALUATION (IN CLOCK CYCLES).

Implementation / Countermeasure	**Cycles:**
Complete unprotected:	683 061
Complete ephemeral:	930 915
Complete static:	2 316 986
Cost per `cswaprr` 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

cryptographic libraries. We choose the libraries for their wide-spread use (boringSSL) or because they claim to be targeting smaller embedded systems (bearSSL, wolfSSL, ARM MBed TLS). We included both, publicly available commercial libraries and open source implementations. Table 2 reports the cycle counts, obtained using the same ARM Cortex-M4 and parameters as used in our work, and whether the libraries are constant time.

TABLE 2. PERFORMANCE OF RELATED CRYPTO LIBRARIES.

Library	**Constant Time**	**Cycles:**
wolfSSL [118] size:	yes	45 930 947
wolfSSL [118] speed:	yes	1 974 047
bearSSL i31 [107]:	yes	2 576 639
boringSSL [49]: fixed base	yes	1 591 407
boringSSL [49]: var. base	yes	2 516 476
ARM MBed TLS [6]	no	6 438 233

We see that the considered libraries require between 1 974k and 2 576k 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 while providing much less SCA and FI protections. A detailed security analysis of the considered cryptographic libraries is presented in Appendix A.

5.2. Side-channel Evaluation

This subsection presents our SCA setup and 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 [95] and record traces using the LeCroy WaveRunner 610Zi oscilloscope. For analysis of the traces we use the Inspector software by Riscure [96].

For all the results presented in this section we compile the code using the `-O2` optimization flag. However, we also performed every test without any optimizations (`-O0`), since using no optimizations often inflates existing leakage. The results are consistent for both optimization flags with respect to detected leakage.

We performed our leakage detection experiments in the following four scenarios:

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

- The ephemeral implementation, with all countermeasures 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 increased resistance to profiled attacks.

We compare power profiles of our standard unprotected, ephemeral, and static implementations in Appendix B.

We apply a commonly-used TVLA methodology [14], [48], [61], [110] to our implementations using the aforementioned measurement settings. Following this 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 t-test statistics measured between group 0 and 1, and between 0 and 2.

Initially [61], [110] the TVLA confidence threshold to detect leakage in the t-test statistic was set to 4.5, which lead to many false positives. Thus, we compute the confidence threshold for our experiments using the threshold formula by Ding, Zhang, Durvaux, Standaert, and Fei [33], following the approach by Papachristodoulou, Fournaris, Papagiannopoulos, and Batina [90]. The threshold computed this way is more accurate and ensures that false positives are avoided. For all our experiments the computed threshold is between 7 and 7.3; we therefore assume that peaks above 7 indicate leakage.

In our evaluation we concentrate not only the Montgomery ladder executions, 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 blinding 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 the traces by choosing a distinctive pattern from the first trace and matching it 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. Alignment at different locations shows similar results. The result for groups 0 and 2 is similar and is enclosed in Appendix B. Each group consists of 1000 traces.

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

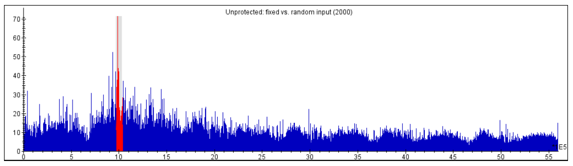

Figure 1. TVLA results for the unprotected implementation: fixed vs random point. The red color marks the alignment.

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

In Figure 2 we present the t-test results and we see that 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 vary between execution. As we see if the same scalar is used multiple times then the leakage can be detected.

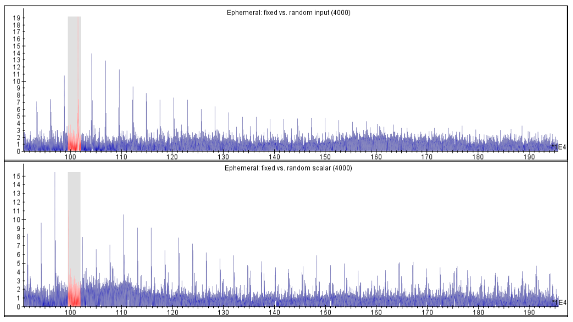

Figure 2. TVLA for the ephemeral implementation (0.9ms-2.0ms): fixed vs random point (top) and fixed vs random scalar (bottom).

For groups 0 and 1 the highest peak reaches 19. This seems unexpected since we employ the projective re-randomization. However, this peak can be attributed to the repeated use of the non-randomized public x-coordinate in each invocation of ladderstep. Such "leakage" of public inputs is not exploitable but obviously generates a t-test peak, since the input is a TVLA parameter. Essentially, what we encounter here is 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` is 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 parameter `b1` to `fe25519_mul`, and therefore its output `b4` too, are randomized. The results produced by this implementation are incorrect, but we only use it for the purpose of evaluation. The result of t-test for this modified implementation shows that the leakage is not present anymore, because the highest peak does not reach 4.4; for the corresponding t-test plot see Appendix B. Therefore, we conclude that the first order leakage is not present.

The leakage present for groups 0 and 1 suggests that the ephemeral implementation might be vulnerable to single-trace profiled attacks. For example, it might be possible to learn some information during copying of the ephemeral scalar. However, the consequences of this attack would not be severe since the scalar is ephemeral and copying is done in 32-bit chunks, which is expected to leak relatively little information[4]. Note that we implement countermeasures against these attacks in the static case.

Observe that we were able to find out the aforementioned leakage only due to using the same scalar multiple times and it does not imply that the single-trace attack would be successful if the ephemeral scalar is not re-used.

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 corresponding t-test plot 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 note that the peaks in both experiments are below 4.3, and therefore, no leakage is detected. We also align at various other 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.

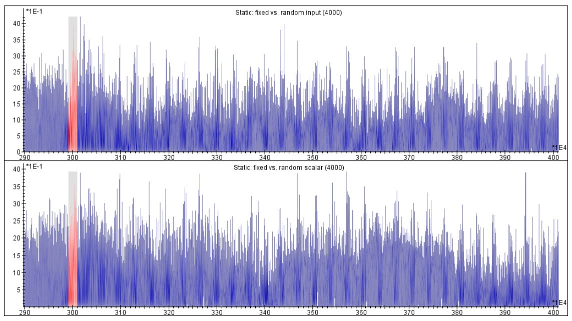

Figure 3. TVLA for the static implementation (2.9ms-4.0ms): fixed vs random point (top) and fixed vs random scalar (bottom).

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 have shown that the static implementation is resistant not only to standard attacks, like DPA and CPA, but also to multiple-trace attacks exploiting horizontal leakage, like cross-correlation and OTA.

Profiled attacks. The relevant profiling attack is a TA targeting the cswap [86]. Note that in our context this attack is already hard because both 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 verify the resistance of our implementation to this attack by performing a TVLA experiment on an implementation with all scalar randomizations (i.e., scalar and storage blinding) turned off. In this setting we check 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 implementation with the scalar and address randomizations turned off. The highest t-test peak is reaching 12.5. This leakage might be used by the aforementioned TA to recover the scalar from the cswaprr instructions. 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.

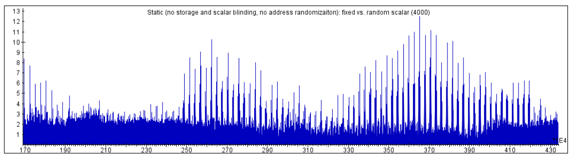

Figure 4. TVLA for static implementation (no blindings and address randomization) aligned at 1.65ms (1.7ms-4.3ms): fixed vs random scalar.

To validate whether the address randomization works correctly 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 usable for the TA [86] is present. The corresponding t-test plot is presented in Appendix B.

Although we have discovered that the leakage exploitable by the "standard" TA [86] is not present, a more complex profiled attack, DL-based for example, might still be theoretically feasible. However, we believe that the absence of the "classical" leakage suggests that even more complex profiled attacks would not achieve sufficient success rates (per single traces) to recover the scalar.

Another profiled attack, which we have not discussed here yet, is against the scalar transfer. 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 attack on key-transfer would recover the Hamming weight of 32-bit chunks of the blinded scalar. Furthermore, single traces need to be attacked, since a blinded scalar is stored together with the blinding and these values are updated for every scalar multiplication. Therefore, we conclude that it would be hard to mount a profiled attack on the scalar transfer.

4. Since 32-bit words of the scalar are being copied, we expect that only the corresponding 32-bit Hamming weights might be leaking.

6. Conclusions and Future Work

We have implemented and 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 while being at least as efficient as widely-deployed ECC libraries. 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, including deep-learning ones, and the corresponding protections for future work. Furthermore, formally proving that our implementations are side-channel resistant, including resistance against first-order DPA attacks, is a potential research direction.

Acknowledgements

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

References

[1] Abarzúa, R., Valencia, C., and López, J. 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] Adamantiadis, A., Josefsson, S., and Baushke, M. Secure shell (SSH) key exchange method using Curve25519 and Curve448. IETF RFC 8731, 2020. https://tools.ietf.org/html/rfc8731.

[3] Akishita, T., and Takagi, T. Zero-value point attacks on elliptic curve cryptosystem. In *Information Security* (2003), C. Boyd and W. Mao, Eds., vol. 2851 of *LNCS*, Springer, pp. 218–233.

[4] Almeida, J. B., Barbosa, M., Barthe, G., Dupressoir, F., and Emmi, M. Verifying constant-time implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), CCS '17, ACM, pp. 53–70. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_almeida.pdf.

[5] Antipa, A., Brown, D., Menezes, A., Struik, R., and Vanstone, S. Validation of elliptic curve public keys. In *Public-Key Cryptography – PKC 2003* (2003), Y. G. Desmedt, Ed., vol. 2567 of *LNCS*, Springer, pp. 211–223. https://iacr.org/archive/pkc2003/25670211/25670211.pdf.

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

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

[8] Barthe, G., Belaïd, S., Cassiers, G., Fouque, P.-A., Grégoire, B., and Standaert, F.-X. maskVerif: Automated verification of higher-order masking in presence of physical defaults. In *Computer Security – ESORICS 2019* (2019), K. Sako, S. Schneider, and P. Y. A. Ryan, Eds., vol. 11735 of *LNCS*, Springer, pp. 300–318.

[9] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., and Strub, P.-Y. Verified proofs of higher-order masking. In *Advances in Cryptology – EUROCRYPT 2015* (2015), E. Oswald and M. Fischlin, Eds., vol. 9056 of *LNCS*, Springer, pp. 457–485.

[10] Batina, L., Chmielewski, Ł., Papachristodoulou, L., Schwabe, P., , and Tunstall, M. Online template attacks. In *Progress in Cryptology – INDOCRYPT 2014* (2014), W. Meier and D. Mukhopadhyay, Eds., vol. 21–36 of *LNCS*, Springer, p. 8885. http://cryptojedi.org/papers/#ota.

[11] Batina, L., Chmielewski, Ł., Papachristodoulou, L., Schwabe, P., and Tunstall, M. Online template attacks. *Journal of Cryptographic Engineering* (2017), 1–16. http://cryptojedi.org/papers/#ota, see also short version [10].

[12] Bauer, A., Jaulmes, E., Prouff, E., and Wild, J. Horizontal collision correlation attack on elliptic curves. In *Selected Areas in Cryptography – SAC 2013* (2014), T. Lange, K. Lauter, and P. Lisoněk, Eds., vol. 8282 of *LNCS*, Springer, pp. 553–570.

[13] Bayrak, A. G., Regazzoni, F., Novo, D., and Ienne, P. Sleuth: Automated verification of software power analysis countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2013* (2013), G. Bertoni and J.-S. Coron, Eds., vol. 8086 of *LNCS*, Springer, pp. 293–310.

[14] Becker, G. T., Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Kouzminov, T., Leiserson, A. J., Marson, M. E., Rohatgi, P., and Saab, S. Test vector leakage assessment (TVLA) methodology in practice. International Cryptographic Module Conference, 2013.

[15] Bernstein, D. J. A state-of-the-art diffie-hellman function. https://cr.yp.to/ecdh.html (accessed 2021-05-05).

[16] Bernstein, D. J. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography – PKC 2006* (2006), M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958 of *LNCS*, Springer, pp. 207–228. http://cr.yp.to/papers.html#curve25519.

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

[18] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y. High-speed high-security signatures. In *Cryptographic Hardware and Embedded Systems – CHES 2011* (2011), B. Preneel and T. Takagi, Eds., vol. 6917 of *LNCS*, Springer, pp. 124–142. see also full version [19].

[19] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y. 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] Biehl, I., Meyer, B., and Müller, V. Differential fault attacks on elliptic curve cryptosystems. In *Advances in Cryptology – CRYPTO 2000* (2000), M. Bellare, Ed., vol. 1880 of *LNCS*, Springer, pp. 131–146.

[21] Blömer, J., Otto, M., and Seifert, J.-P. Sign change fault attacks on elliptic curve cryptosystems. In *Fault Diagnosis and Tolerance in Cryptography* (2006), L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, Eds., vol. 4236 of *LNCS*, Springer, pp. 36–52.

[22] Boneh, D., DeMillo, R. A., and Lipton, R. J. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology – Eurocrypt '97* (1997), W. Fumy, Ed., vol. 1233 of *LNCS*, Springer, pp. 37–51.

[23] Brier, E., Clavier, C., and Olivier, F. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems – CHES 2004* (2004), M. Joye and J.-J. Quisquater, Eds., vol. 3156 of *LNCS*, Springer, pp. 16–29.

[24] Cabrera Aldaya, A., and Brumley, B. B. Online template attacks: Revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2021*, 3 (Jul. 2021), 28–59.

[25] Cagli, E., Dumas, C., and Prouff, E. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *Cryptographic Hardware and Embedded Systems – CHES 2017* (2017), W. Fischer and N. Homma, Eds., vol. 10529 of *LNCS*, Springer, pp. 45–68.

[26] Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology — CRYPTO' 99* (Berlin, Heidelberg, 1999), M. Wiener, Ed., Springer Berlin Heidelberg, pp. 398–412.

[27] Chari, S., Rao, J. R., and Rohatgi, P. Template attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2002* (2002), B. S. K. Jr., C. K. Koc, and C. Paar, Eds., vol. 2523 of *LNCS*, Springer, pp. 13–28.

[28] Chmielewski, Ł., Massolino, P. M. C., Vliegen, J., Batina, L., and Mentens, N. Completing the complete ECC formulae with countermeasures. *Journal of Low Power Electronics and Applications 7*, 1 (2017). https://www.mdpi.com/2079-9268/7/1/3.

[29] Ciet, M., and Joye, M. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, Codes and Cryptography 36* (2005), 33–43.

[30] Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., and Verneuil, V. Horizontal correlation analysis on exponentiation. In *Information and Communications Security* (2010), M. Soriano, S. Qing, and J. López, Eds., vol. 6476 of *LNCS*, Springer, pp. 46–61. http://eprint.iacr.org/2003/237.

[31] Coron, J.-S. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems* (1999), Ç. K. Koç and C. Paar, Eds., vol. 1717 of *LNCS*, Springer, pp. 292–302.

[32] Danger, J.-L., Guilley, S., Hoogvorst, P., Murdica, C., and Naccache, D. A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. *Journal of Cryptographic Engineering 3*, 4 (2013), 1–25.

[33] Ding, A. A., Zhang, L., Durvaux, F., Standaert, F., and Fei, Y. Towards sound and optimal leakage detection procedure. In *Smart Card Research and Advanced Applications* (2017), T. Eisenbarth and Y. Teglia, Eds., vol. 10728 of *LNCS*, Springer, pp. 105–122.

[34] Dugardin, M., Papachristodoulou, L., Najm, Z., Batina, L., Danger, J.-L., and Guilley, S. Dismantling real-world ecc with horizontal and vertical template attacks. In *Constructive Side-Channel Analysis and Secure Design* (Cham, 2016), F.-X. Standaert and E. Oswald, Eds., Springer International Publishing, pp. 88–108.

[35] Duong, T. 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).

[36] Eldib, H., Wang, C., and Schaumont, P. Smt-based verification of software countermeasures against side-channel attacks. In *Tools and Algorithms for the Construction and Analysis of Systems* (2014), E. Ábrahám and K. Havelund, Eds., vol. 8413 of *LNCS*, Springer, pp. 62–77.

[37] Fan, J., Gierlichs, B., and Vercauteren, F. To infinity and beyond: Combined attack on ECC using points of low order. In *Cryptographic Hardware and Embedded Systems – CHES 2011* (2011), B. Preneel and T. Takagi, Eds., vol. 6917 of *LNCS*, Springer, pp. 143–159.

[38] Fan, J., Guo, X., Mulder, E. D., Schaumont, P., Preneel, B., and Verbauwhede, I. 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)* (2010), IEEE, pp. 76–87.

[39] Fan, J., and Verbauwhede, I. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security: From Theory to Applications* (2012), D. Naccache, Ed., vol. 6805 of *LNCS*, Springer, pp. 265–282.

[40] Foundation, O. Unified architecture, 2008. https://opcfoundation.org/about/opc-technologies/opc-ua/ (accessed 2021-05-05).

[41] Fouque, P.-A., Lercier, R., Réal, D., and Valette, F. Fault attack on elliptic curve with Montgomery ladder implementation. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography* (2008), IEEE, pp. 92–98. https://www.di.ens.fr/~fouque/pub/fdtc08.pdf.

[42] Fouque, P.-A., Réal, D., Valette, F., and Drissi, M. The carry leakage on the randomized exponent countermeasure. In *Cryptographic Hardware and Embedded Systems – CHES 2008* (2008), E. Oswald and P. Rohatgi, Eds., vol. 5154 of *LNCS*, Springer, pp. 198–213.

[43] Fouque, P.-A., and Valette, F. The doubling attack — Why upwards is better than downwards. In *Cryptographic Hardware and Embedded Systems – CHES 2003* (2003), C. D. Walter, Çetin K. Koç, and C. Paar, Eds., vol. 2779 of *LNCS*, Springer, pp. 269–280. www.ssi.gouv.fr/archive/fr/sciences/fichiers/lcr/fova03.pdf.

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

[45] Fujii, H., and Aranha, D. F. Curve25519 for the Cortex-M4 and beyond. In *Progress in Cryptology – LATINCRYPT 2017* (2019), T. Lange and O. Dunkelman, Eds., vol. 11368 of *LNCS*, Springer, pp. 109–127. http://www.cs.haifa.ac.il/~orrd/LC17/paper39.pdf.

[46] Genkin, D., Pachmanov, L., Pipman, I., and Tromer, E. ECDH key-extraction via low-bandwidth electromagnetic attacks on PCs, 2016. https://web.eecs.umich.edu/~genkin/papers/ecdh.pdf.

[47] Genkin, D., Valenta, L., and Yarom, Y. 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* (2017), ACM, pp. 845–858.

[48] Goodwill, G., Jun, B., Jaffe, J., and Rohatgi, P. 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.

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

[50] Goubin, L. A refined power-analysis attack on elliptic curve cryptosystems. In *Public Key Cryptography – PKC 2003* (2002), Y. G. Desmedt, Ed., vol. 2567 of *LNCS*, Springer, pp. 199–211.

[51] Haase, B., and Labrique, B. Making password authenticated key exchange suitable for resource-constrained industrial control devices. In *Cryptographic Hardware and Embedded Systems – CHES 2017* (2017), W. Fischer and N. Homma, Eds., vol. 10529 of *LNCS*, Springer, pp. 346–364.

[52] Haase, B., and Labrique, B. 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.

[53] Hanley, N., Kim, H., and Tunstall, M. Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In *Topics in Cryptology – CT-RSA 2015* (2015), K. Nyberg, Ed., vol. 9048 of *LNCS*, Springer, pp. 431–448.

[54] Heyszl, J., Mangard, S., Heinz, B., Stumpf, F., and Sigl, G. Localized electromagnetic analysis of cryptographic implementations. In *Topics in Cryptology – CT-RSA 2012* (2012), O. Dunkelman, Ed., vol. 7178 of *LNCS*, Springer, pp. 231–244.

[55] Hoenicke, J. Extracting the private key from a trezor ... with a 70$ oscilloscope, 2015. https://jochen-hoenicke.de/crypto/trezor-power-analysis/ (accessed 2012-12-02).

[56] Homma, N., Miyamoto, A., Aoki, T., Satoh, A., and Shamir, A. Collision-based power analysis of modular exponentiation using chosen-message pairs. In *Cryptographic Hardware and Embedded Systems – CHES 2008* (2008), E. Oswald and P. Rohatgi, Eds., vol. 5154 of *LNCS*, Springer, pp. 15–29. http://www.aoki.ecei.tohoku.ac.jp/crypto/pdf/CHES2008_homma.pdf.

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

[58] Information technology – security techniques – testing methods for the mitigation of non-invasive attack classes against cryptographic modules. Standard, International Organization for Standardization, Geneva, CH, 2016.

[59] ITOH, K., IZU, T., AND TAKENAKA, M. Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. In *Cryptographic Hardware and Embedded Systems – CHES 2002* (2003), B. S. Kaliski, çetin K. Koç, and C. Paar, Eds., vol. 2523 of *LNCS*, Springer, pp. 129–143.

[60] ITOH, K., IZU, T., AND TAKENAKA, M. A practical countermeasure against address-bit differential power analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2003* (2003), C. D. Walter, Çetin K. Koç, and C. Paar, Eds., vol. 2779 of *LNCS*, Springer, pp. 382–396.

[61] JAFFE, J., ROHATGI, P., AND WITTEMAN, M. Efficient side-channel testing for public key algorithms: RSA case study. Tech. rep., Cryptography Research Inc. & Riscure, 2011. http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/09_Jaffe.pdf.

[62] JANCAR, J., SEDLACEK, V., SVENDA, P., AND SYS, M. 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.

[63] JOYE, M., AND YEN, S.-M. The Montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems – CHES 2002* (2003), B. S. Kaliski, çetin K. Koç, and C. Paar, Eds., vol. 2523 of *LNCS*, Springer, pp. 291–302.

[64] KASPER, M., PETRI, R., FELDHUSEN, D., GEBHARDT, M., ILLIES, G., LOCHTER, M., STEIN, O., THUMSERANG, W., AND WICKE, G. 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).

[65] KIM, J., PICEK, S., HEUSER, A., BHASIN, S., AND HANJALIC, A. 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.

[66] KOBLITZ, N. Elliptic curve cryptosystems. *Mathematics of Computation 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.

[67] KOCHER, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology – CRYPTO'96* (1996), N. Koblitz, Ed., vol. 1109 of *LNCS*, Springer, pp. 104–113. http://www.cryptography.com/public/pdf/TimingAttacks.pdf.

[68] KOCHER, P. C., JAFFE, J., AND JUN, B. Differential power analysis. In *Advances in Cryptology – CRYPTO '99* (1999), M. Wiener, Ed., vol. 1666 of *LNCS*, Springer, pp. 388–397.

[69] KRAWCZYK, H. SIGMA: The 'SIGn-and-MAc' approach to authenticated diffie-hellman and its use in the IKE protocols. In *Advances in Cryptology – CRYPTO 2003* (2003), D. Boneh, Ed., vol. 2729 of *LNCS*, Springer, pp. 400–425. https://webee.technion.ac.il/~hugo/sigma-pdf.pdf.

[70] LESLY-ANN DANIEL, SÉBASTIEN BARDIN, T. R. BINSEC/REL: Efficient relational symbolicexecution for constant-time at binary-level. In *2020 IEEE Symposium on Security and Privacy* (2020), pp. 1021–1038. https://people.csail.mit.edu/jgross/personal-website/papers/2019-fiat-crypto-ieee-sp.pdf.

[71] LIU, Z., LONGA, P., PEREIRA, G. C. C. F., REPARAZ, O., AND SEO, H. Four$\mathbb{Q}$ on embedded devices with strong countermeasures against side-channel attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2017* (2017), W. Fischer and N. Homma, Eds., vol. 10529 of *LNCS*, Springer, pp. 665–686. https://eprint.iacr.org/2017/434.

[72] LOCHTER, M., MERKLE, J., SCHMIDT, J.-M., AND SCHÜTZE, T. 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.

[73] LOISEAU, A., LECOMTE, M., AND FOURNIER, J. J. A. Template attacks against ECC: practical implementation against curve25519. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2020), IEEE, pp. 13–22.

[74] MAGHREBI, H., PORTIGLIATTI, T., AND PROUFF, E. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering* (2016), C. Carlet, M. A. Hasan, and V. Saraswat, Eds., vol. 10076 of *LNCS*, Springer, pp. 3–26.

[75] MARLINSPIKE, M. Simplifying OTR deniability, 2013. https://signal.org/blog/simplifying-otr-deniability/ (accessed 2021-05-05).

[76] MATHEWSON, N. Cryptographic directions in tor. Presentation at Real-World Crypto 2016, 2016. https://rwc.iacr.org/2016/Slides/nickm-rwc-presentation.pdf.

[77] MCCANN, D., OSWALD, E., AND WHITNALL, C. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In *Proceedings of the 26th USENIX Security Symposium* (2017), USENIX, pp. 199–216. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-mccann.pdf.

[78] MEDWED, M., AND OSWALD, E. Template attacks on ECDSA. In *Information Security Applications* (2009), K.-I. Chung, K. Sohn, and M. Yung, Eds., vol. 5379 of *LNCS*, Springer, pp. 14–27.

[79] MENEZES, A. J., OKAMOTO, T., AND VANSTONE, S. Reducing elliptic curve logarithms to logarithms in a finite field. *Transactions on Information Theory 39*, 5 (1993), 1639–1646.

[80] MESSERGES, T. S. Using second-order power analysis to attack dpa resistant software. In *Cryptographic Hardware and Embedded Systems — CHES 2000* (Berlin, Heidelberg, 2000), Ç. K. Koç and C. Paar, Eds., Springer Berlin Heidelberg, pp. 238–251.

[81] MILLER, V. S. Use of elliptic curves in cryptography. In *Advances in Cryptology – CRYPTO '85: Proceedings* (1986), H. C. Williams, Ed., vol. 218 of *LNCS*, Springer, pp. 417–426.

[82] MONTGOMERY, P. L. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation 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.

[83] MOSS, A., OSWALD, E., PAGE, D., AND TUNSTALL, M. Compiler assisted masking. In *Cryptographic Hardware and Embedded Systems – CHES 2012* (2012), E. Prouff and P. Schaumont, Eds., vol. 7428 of *LNCS*, Springer, pp. 58–75.

[84] NACCACHE, D., SMART, N. P., AND STERN, J. Projective coordinates leak. In *Advances in Cryptology – EUROCRYPT 2004* (2004), C. Cachin and J. Camenisch, Eds., vol. 3027 of *LNCS*, Springer, pp. 257–267. https://www.iacr.org/archive/eurocrypt2004/30270258/projective.pdf.

[85] NASCIMENTO, E., AND CHMIELEWSKI, Ł. Applying horizontal clustering side-channel attacks on embedded ECC implementations. In *Smart Card Research and Advanced Applications* (2017), T. Eisenbarth and Y. Teglia, Eds., vol. 10728 of *LNCS*, Springer, pp. 213–231.

[86] NASCIMENTO, E., CHMIELEWSKI, Ł., OSWALD, D., AND SCHWABE, P. Attacking embedded ECC implementations through cmov side channels. In *Selected Areas in Cryptology – SAC 2016* (2017), R. Avanzi and H. Heys, Eds., vol. 10532 of *LNCS*, Springer, pp. 99–119. https://cryptojedi.org/papers/#cmovsca.

[87] NIR, Y., JOSEFSSON, S., AND PEGOURIE-GONNARD, M. 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.

[88] NXP. 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.

[89] OSWALD, D., AND PAAR, C. Breaking Mifare DESFire MF3ICD40: Power analysis and templates in the real world. In *Cryptographic Hardware and Embedded Systems – CHES 2011* (2011), B. Preneel and T. Takagi, Eds., vol. 6917 of *LNCS*, Springer, pp. 207–222.

[90] Papachristodoulou, L., Fournaris, A. P., Papagianopoulos, K., and Batina, L. 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.

[91] Perin, G., Chmielewski, Ł., Batina, L., and Picek, S. 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.

[92] Pollard, J. M. Monte Carlo methods for index computation (mod p). *Mathematics of Computation 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.

[93] Prouff, E., and Rivain, M. Masking against side-channel attacks: A formal security proof. In *Advances in Cryptology – EUROCRYPT 2013* (Berlin, Heidelberg, 2013), T. Johansson and P. Q. Nguyen, Eds., Springer Berlin Heidelberg, pp. 142–159.

[94] Renes, J., Costello, C., and Batina, L. Complete addition formulas for prime order elliptic curves. In *Advances in Cryptology - EUROCRYPT 2016* (2016), M. Fischlin and J. Coron, Eds., vol. 9665 of *LNCS*, Springer, pp. 403–428.

[95] Riscure. Current probe. security test tool for embedded devices, 2018. https://www.riscure.com/product/current-probe/ (accessed 2021-05-05).

[96] Riscure. Side channel analysis security tools, 2021. https://www.riscure.com/security-tools/inspector-sca/.

[97] Roelofs, N., Samwel, N., Batina, L., and Daemen, J. Online template attack on ecdsa:. In *Progress in Cryptology - AFRICACRYPT 2020* (Cham, 2020), A. Nitaj and A. Youssef, Eds., Springer International Publishing, pp. 323–336.

[98] Samwel, N., Batina, L., Bertoni, G., Daemen, J., and Susella, R. Breaking Ed25519 in WolfSSL. In *Topics in Cryptology – CT-RSA 2018* (2018), N. P. Smart, Ed., vol. 10808 of *LNCS*, Springer, pp. 1–20. https://eprint.iacr.org/2017/985.

[99] Santis, F. D., and Sigl, G. Towards side-channel protected x25519 on arm cortex-m4 processors, 2016.

[100] Sasdrich, P., and Güneysu, T. Implementing Curve25519 for side-channel-protected elliptic curve cryptography. *Transactions on Reconfigurable Technology and Systems 9*, 1 (2015), 1–15.

[101] Sasdrich, P., and Güneysu, T. 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* (2017), IEEE, pp. 1–6.

[102] Schwabe, P., and Stoffelen, K. All the aes you need on cortex-m3 and m4. In *Selected Areas in Cryptology – SAC 2016* (2017), R. Avanzi and H. Heys, Eds., vol. 10532 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, pp. 180–194. Document ID: 9fc0b970660e40c264e50ca389dacd49, https://cryptojedi.org/papers/#aesarm.

[103] Shelton, M. A., Samwel, N., Batina, L., Regazzoni, F., Wagner, M., and Yarom, Y. Rosita: Towards automatic elimination of power-analysis leakage in ciphers. In *Network and Distributed System Security Symposium (NDSS)* (2022 (to appear)), Internet Society. https://www.ndss-symposium.org/ndss-paper/rosita-towards-automatic-elimination-of-power-analysis-leakage-in-ciphers/.

[104] Smart, N. P. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology 12*, 3 (1999), 193–196.

[105] STMicroelectronics. Reference manual – STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced arm-based 32-bit MCUs. Tech. Rep. RM0090 Rev 17, STMicroelectronics, 2018. 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.

[106] Systems, O. W. Signal protocol. https://signal.org/ (accessed 2021-05-05).

[107] Bearssl, 2018. https://bearssl.org/ (accessed 2021-05-05).

[108] Timmers, N., Spruyt, A., and Witteman, M. Controlling PC on ARM using fault injection. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC* (2016), IEEE, pp. 25–35.

[109] The Noise protocol framework (revision 34), 2018. https://noiseprotocol.org/noise.pdf.

[110] Tunstall, M., and Goodwill, G. Applying tvla to public key cryptographic algorithms. Cryptology ePrint Archive, Report 2016/513, 2016. http://eprint.iacr.org/2016/513.

[111] van Oorschot, P. C., and Wiener, M. J. Parallel collision search with cryptanalytic applications. *Journal of Cryptology 12*, 1 (1999), 1–28. http://www.scs.carleton.ca/~paulv/papers/JoC97.pdf.

[112] Walter, C. D. Sliding windows succumbs to big mac attack. In *Cryptographic Hardware and Embedded Systems – CHES 2001* (2001), Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *LNCS*, Springer, pp. 286–299.

[113] Weissbart, L., Chmielewski, L., Picek, S., and Batina, L. 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.

[114] Weissbart, L., Picek, S., and Batina, L. One trace is all it takes: Machine learning-based side-channel attack on EdDSA. In *Security, Privacy, and Applied Cryptography Engineering* (2019), S. Bhasin, A. Mendelson, and M. Nandi, Eds., vol. 11947 of *LNCS*, Springer, pp. 86–105.

[115] Whitnall, C., and Oswald, E. 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* (2019), S. D. Galbraith and S. Moriai, Eds., vol. 11923 of *LNCS*, Springer, pp. 256–284.

[116] Witteman, M. Secure application programming in the presence of side channel attacks. Tech. rep., Riscure, 2018. https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf.

[117] Witteman, M. F., van Woudenberg, J. G. J., and Menarini, F. Defeating RSA multiply-always and message blinding countermeasures. In *Topics in Cryptology – CT-RSA 2011* (2011), A. Kiayias, Ed., vol. 6558 of *LNCS*, Springer, pp. 77–88. https://www.riscure.com/benzine/documents/rsacc_ctrsa_final.pdf.

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

[119] Yan, Y., and Oswald, E. Examining the practical side channel resilience of arx-boxes. In *Proceedings of the 16th ACM International Conference on Computing Frontiers* (2019), ACM, pp. 373–379. https://eprint.iacr.org/2019/335.

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

[121] Zhou, Y., and Standaert, F.-X. 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.

Appendix A. Security properties of related crypto-libraries

None of the other implementations we benchmarked exceeds 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 arithmetic on the Edwards curve and large precomputed tables, a strategy which we do not believe suitable for smaller embedded targets due to code-size limitations. All other implementations use the Montgomery ladder strategy on the Montgomery curve in projective coordinates. The implementations differ in the way field elements are represented and reduction is carried out, and how aggressively they optimize for code size and speed. 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. BearSSL uses a 9-limb representation with 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 used by Genkin, Valenta, and Yuval [47] become feasible if the adversary inserts a low-order point, because multiplications by 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 attacks.

We communicated our concerns to the developers of ARM Mbed TLS [6] and suggested different strategies for mitigating the problem. They acknowledged the issue and implemented countermeasures in version 3.0.0: https://github.com/ARMmbed/mbedtls/releases/tag/v3.0.0.

Appendix B. Supplementary Material for SCA Evaluation

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

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 performed 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.

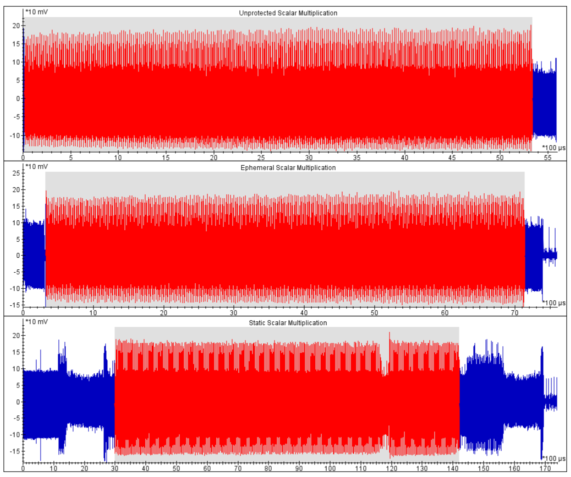

Figure 5. Power profiles of unprotected (top), ephemeral (middle), and static (bottom) implementations.

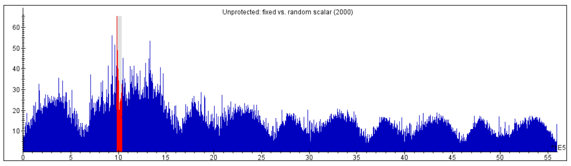

Figure 6. Unprotected implementation TVLA: fixed vs random scalar.

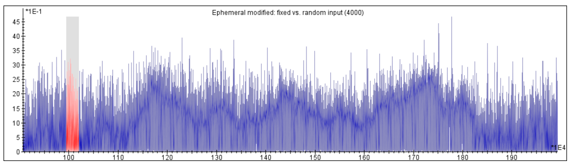

Figure 7. Modified ephemeral implementation TVLA: (0.9ms-2.0ms): fixed vs random point.

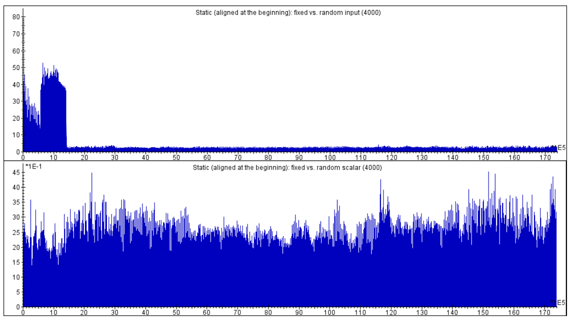

Figure 8. Static implementation TVLA: fixed vs random point (top) and fixed vs random scalar (bottom).

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 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.

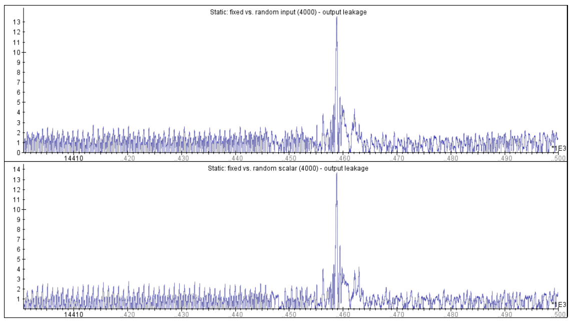

Figure 9. Static implementation TVLA (14.4-14.5ms), aligned at 13.0ms: fixed vs random point (top) and fixed vs random scalar (bottom).

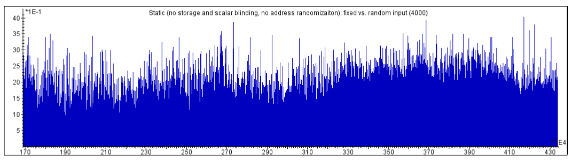

Figure 10. Static impl. with blindings and address randomization disabled TVLA (1.7-4.3ms), aligned at 1.65ms: fixed vs random point.

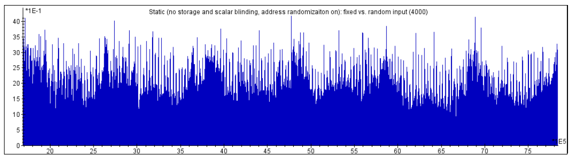

Figure 11. Static impl. with blindings turned off and the address rand. turned on TVLA (1.7-7.8ms), aligned at 1.65ms: fixed vs random scalar.