IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 557–589. DOI:10.46586/tches.v2023.i1.557-589

SoK: SCA-secure ECC in software – mission impossible?

Lejla Batina[1] *, Łukasz Chmielewski[2,1] †, Björn Haase[3], Niels Samwel[1] and Peter Schwabe[4,1] ‡

[1] Radboud University, Nijmegen, The Netherlands
lejla@cs.ru.nl,nsamwel@cs.ru.nl
[2] Masaryk University, Brno, Czech Republic
chmiel@fi.muni.cz
[3] Endress+Hauser Liquid Analysis GmbH&Co. KG, Germany
bjoern.m.haase@web.de
[4] Max Planck Institute for Security and Privacy, Bochum, Germany
peter@cryptojedi.org

Abstract. This paper describes an ECC implementation computing the X25519 key-exchange protocol on the Arm Cortex-M4 microcontroller. For providing protections against various side-channel and fault attacks we first review known attacks and countermeasures, then we provide software implementations that come with extensive mitigations, and finally we present a preliminary side-channel evaluation. To our best knowledge, this is the first public software claiming affordable protection against multiple classes of attacks that are motivated by distinct real-world application scenarios. We distinguish between X25519 with ephemeral keys and X25519 with static keys and show that the overhead to our baseline unprotected implementation is about 37% and 243%, 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 at least as efficient as widely-deployed ECC cryptographic libraries, which offer much less protection.

Keywords: Elliptic Curve Cryptography · Side-Channel Analysis · Fault Injection

1 Introduction

Elliptic-curve cryptography (ECC) presents the current state of the art in public-key cryptography, both for key agreement and for digital signatures. The reason for ECC getting ahead of RSA is mainly 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 [MOV93, Sma99], 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 [Kob87] and Miller [Mil86]. The best known attack is still the Pollard rho algorithm [Pol78] and its parallel version by van Oorschot and Wiener [vOW99].

*This work was partially funded by PROACT (NWA.1215.18.014), a project financed by the Netherlands Organisation for Scientific Research (NWO).
†Part of this work was carried out while the author was employed by Riscure, Delft, The Netherlands.
‡This research was supported by Deutsche Forschungsgemeinschaft (DFG, German research Foundation) as part of the Excellence Strategy of the German Federal and State Governments – EXC 2092 CASA - 390781972; and by the European Commission through the ERC Starting Grant 805031 (EPOQUE).

Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

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 attack, was introduced in the late 90s [Koc96, KJJ99, BDL97], 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 [KJJ99], 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]. The most notable class of profiled attacks is called *Template Attacks* (TAs) [CRR02]. 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 on, but to name a few recent real-world examples of such attacks against ECC consider the "Minerva" timing attack against multiple implementations of ECDSA [JSSS20], the power-analysis attack recovering the secret key from a TREZOR hardware bitcoin wallet [Hoe15], or the side-channel attack against the Google Titan secure element [RLMI21]. 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 [FV12, FGM+10, DGH+13, AVL19].

Our goal in this paper is to go beyond reviewing and systematizing the state of the art in ECC side-channel and fault attacks, but to consolidate this knowledge through publicly available implementations. More specifically, we present two carefully optimized software implementations of X25519 elliptic-curve Diffie-Hellman key exchange [Ber06] with extensive state-of-the art SCA and FI countermeasures targeting the popular Arm Cortex-M4 microprocessor. The two implementations differ in their targeted use-case scenario and consequently their protection level: the first implementation is offering protections that we believe to be sufficient to protect *ephemeral* keys, i.e., in a scenario where attackers are not able to mount differential SCA or FI attacks. The second implementation aims to protect X25519 with static keys and thus adds protections against differential attacks. The overhead in terms of CPU cycles compared to a carefully optimized baseline unprotected implementation is about 37% for the implementation targeting the ephemeral use-case and and 239% for the static one. In absolute numbers even the more extensively protected implementation targeting the static use-case outperforms ECC implementations in widely deployed cryptographic libraries. 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.

To verify to what extent our protections are secure, we performed an experimental security evaluation. First we applied a commonly used test-vector leakage assessment (TVLA) [BCD+13] in various settings to detect different types of leakages. We additionally execute profiled single-trace attacks to confirm that our countermeasures also provide a certain level of protection against those more advanced side-channel attacks.

Surprisingly, despite the extensive body of literature on implementation attacks against ECC, we are not aware of a single paper describing a publicly available implementation with a similar level of SCA and FI protection. There are implementations of ECC that *claim* to offer extensive protection. 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*

[1] Nowadays DPA is rarely used in favor of an improved method called *Correlation Power Analysis* (CPA) [BCO04] that replaces difference of means with Pearson correlation.

all side channel and fault attacks as well as reverse engineering efforts" [NXP15]. However, like essentially all commercially available 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, for example, [SBB+18, GPPT16].

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 (see [FV12, Sec. 6.2]), 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.

Application scenario. 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 for both 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 EM and power leakage 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.

Most critical-infrastructure systems cannot use conventional smart-card circuits, specifically due to constraints imposed by power-budget, 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 [HL17] clearly exceeded by most off-the-shelf smart-card chips. Appliances for medical use or for food and drug production 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. Consequently, in this paper, we consider ECC running on embedded devices without dedicated cryptographic hardware[2]. This software-only approach has the additional advantage of encouraging independent evaluation of our implementations. Development boards featuring our target are usually available for around US$ 20, allowing any researcher to run and analyze our implementations. Furthermore, although we have used slightly more advanced equipment, we believe that a side-channel researcher equipped with a rather modest SCA equipment, for example, ChipWhisperer-Lite 32-Bit[3] and a middle-class oscilloscope[4] should be able to attack our implementations. As a specific target platform we chose the Arm Cortex-M4, because of its wide availability and popularity in research (see, e.g., [Len20, FA18, FA19, KRSS19] and industry (see, for example, [Shi21] stating that the number of Cortex-M CPUs sold per quarter is 4.4 billion;

[2]While we concentrate on software only, we believe that our implementations 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.

[3]https://www.newae.com/products/NAE-CWLITE-ARM

[4]For example, we used in the past Picoscope 3406D:
https://www.picotech.com/oscilloscope/3000/usb3-oscilloscope-logic-analyzer

this includes also other chips from the same family like the Cortex-M0 and M3).

Why target X25519? The X25519 elliptic-curve Diffie-Hellman key exchange, proposed in 2006 by Bernstein [Ber06, Ber14], is used in many state-of-the-art security protocols like in TLS [NJPG18], SSH [AJB20], the Noise Framework [Tre18, Sec. 12], the Signal protocol [Sys], and Tor [Mat16]. For a more extensive list see [IAN]. Moreover, X25519 is now also considered as a primitive for the OPC/UA protocol family [Fou08] by the OPC/UA security working group, specifically targeting IoT applications with both increased protection demands and low computational resources.

Besides the popularity of X25519, another reason why we chose it as a target is because we make an attempt to settle a dispute 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 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 [LMSS15, Sec. 3.1] leading to a significant increase of SCA-protection cost. 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 [GVY17]. As one way to alleviate these subgroup concerns it is often suggested to validate inputs [ABM+03, Duo15, Aum17]; however Bernstein's Curve25519 FAQ [Ber] 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 for certain malicious public keys in the ephemeral case.

Related work. There is a vast amount of literature on side-channel-protected ECC implementations in both, software and hardware. Most related work considers different types of curves than ours, but on the same platform, e.g., Four$\mathbb{Q}$ on Arm Cortex-M4 [LLP+17], or the same curve but without such comprehensive side-channel protections [FA19, SS16].

For various embedded devices a range of high-speed implementations [LLP+17] are proposed for the Four$\mathbb{Q}$ elliptic curve, including scalar multiplication, ECDH key exchange, and digital signatures. All implementations are constant-time and include side-channel 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 TVLA [BCD+13]. The authors 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 [FA19] present an X25519 implementation that is protected against timing attacks by constant-time execution, randomized projective coordinates, and constant-time conditional swaps. 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 hardware countermeasures, i.e., on an FPGA platform for Curve 25519 and Curve448 [SG17, SG15]. 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 30% with a similar increase in LUTs and FFs. The authors also add memory address scrambling to secure the memory accesses against side-channel attacks and correspondingly make DPA

Algorithm 1 The Montgomery ladder for x-coordinate-based X25519 scalar multiplication

Input: $k \in \{0, \ldots, 2^{255} - 1\}$ 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**
  $c \leftarrow k[i] \oplus p$; $p \leftarrow k[i]$ ▷ $k[i]$ denotes bit i of k
  $(X_1, X_2) \leftarrow \mathsf{cswap}(X_1, X_2, c)$
  $(Z_1, Z_2) \leftarrow \mathsf{cswap}(Z_1, Z_2, c)$
  $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}(x_P, X_1, Z_1, X_2, Z_2)$
return (X_1, Z_1)

more complex. For this purpose 2^6 different random addresses were used.

The complete formulas for Weierstrass curves [RCB16] were implemented and protected against SCA for an FPGA [CMV+17]. Besides a constant-time implementation, coordinate and scalar randomization were considered. The evaluation is done through timing analysis and TVLA and it shows that applying an increasing level of countermeasures leads to an improved SCA resistance. However, this work only considers a limited side-channel adversary that is only passive and not capable of profiling.

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 [Mon87]: $E : y^2 = x^3 + 486662x^2 + x$ over the finite field $\mathbb{F}_p$ with $p = 2^{255} - 19$. The group of $\mathbb{F}_p$-rational points on E has order $8 \cdot \ell$, where ℓ is a 252-bit prime. The central operation is scalar multiplication $\mathsf{smult}(k, x_P)$, which receives as input 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 $x_P \in \{0, \ldots, 2^{255} - 1\}$ and sets the least significant 3 bits of k and the most significant bit of k to zero, and the second-most significant bit of k to one (ensuring that $k \in 8 \cdot \{2^{251}, \ldots, 2^{252} - 1\}$). This operation on bits of the input k is often referred to as "clamping"; in our pseudocode we denote it by clamp. Subsequently, 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 [Mon87] using a projective representation $(X : Z)$ of an x-coordinate $x = X/Y$. Pseudocode for this ladder is given in Algorithm 1.

This algorithm uses the two subroutines cswap and ladderstep. The cswap ("conditional swap") routine swaps the first two inputs iff $c = 1$. The ladderstep routine 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 $\mathbb{F}_{2^{255}-19}$.

2.2 The Arm Cortex-M4 microcontroller

Our target platform is the Arm Cortex-M4 microprocessor. For us the most relevant features are: (1) 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 (2) 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 3-stage pipeline. Recent findings [YO19, SSB+21, MOW17] indicate that specific properties of the internal architecture are 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 [YO19] and own findings, we presume that the Cortex-M4 also features 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 in 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 [STM18]. 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 [BDL+11, BDL+12].

Fast X25519 on the Cortex-M4. Multiple earlier papers describe optimized implementations of X25519 on the Arm Cortex-M4 [FA19, FA18]. One of the fastest implementations is by Haase and Labrique [HL19]; which needs only 625 358 cycles for one scalar multiplication on an STM32F407 running at 16 MHz. The implementation is in the public domain and available from https://github.com/BjoernMHaase/fe25519. It serves as an unprotected baseline and starting point for our protected implementations. During our work on this paper, we were made aware of an even faster implementation by Lenngren [Len20]. Unlike the implementation by Haase and Labrique, this code by Lenngren uses access to memory at secret locations.

Implementation details. Our implementation combines low-level routines in assembly and high-level code in C. Thus, some compiler optimizations, especially ones influencing ordering of instructions (e.g., unrolling or inlining), might influence SCA and FI resistance. To avoid that we verify that the de-compiled C code contains all protections using a reverse-engineering tool called Ghidra[5].

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 point 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 (for a passive attacker) and fault injection capability (for an active attacker).

Passive side-channel attacker. We assume that an attacker can collect sufficiently many leakage traces together with the chosen input and corresponding output. In addition, we allow 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.

[5] https://ghidra-sre.org/

To be precise we consider an attacker according to level SL3 as defined in IEC-62443 [IEC13]. This covers capabilities of academic attackers with well-equipped lab, but no powerful state agencies (they are considered for level SL4).

As mentioned above, instead of proving the side-channel security of our implementations in a certain model (e.g., the noisy leakage model [PR13]), we experimentally evaluate their side-channel resistance. We first use TVLA (so-called 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 [ISO16, WO19]. Second, to confirm and extend TVLA results, we execute profiled multivariate single-trace attacks against various sensitive values. In doing so we show that our countermeasures also provide a certain level of protection against these attacks.

A limited fault attacker. Any software-only implementation, such as ours, 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 countermeasures. The fault attacker considered in this paper is thus 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 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, in particular [TSW16].

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

This section describes our X25519 implementation protected against side-channel and fault attacks when deployed in an *ephemeral* key-exchange. In this case a secret scalar is used only twice, once in public-key generation and once in the shared-secret computation. 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 [Kra03] or in combination with static key exchanges in, for example, 3DH [Mar13], 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.

Due to space constraints we do not list all possible attacks below, but we focus on the most relevant passive attacks in Subsection 3.1 and the active ones in Subsection 3.2; for a complete list of attacks see the surveys [FV12, KPF+16]. Finally, in Subsection 3.3 we explain our implementation and how it protects against the listed attacks.

3.1 Relevant passive attacks

Simple power analysis (SPA) [KJJ99] is one of the most common form 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 in the more classical sense as attacks that visually identify secret-data-dependent control flow typically based on instruction-dependent leakages.

Horizontal (collision) attacks. Single-trace attacks beyond SPA are often called *horizontal* attacks. These attacks trace back to the Big-Mac attack [Wal01], and were applied against ECC implementations [CFG+10, BJPW14, BJP+15, HKT15]. 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 [Wal01] and ECC [HKT15].

Horizontal correlation attacks [CFG+10] are based on predictions of intermediate arithmetic results within a single trace. This is effective against scalar-blinding countermeasures, but not against other mitigations, like coordinate or point randomization. Both collision and correlation horizontal attacks were also considered for RSA [BJPW13]. The correlation distinguisher was also successfully replaced with profiled attacks for ECC [PZS17].

Template attacks (TAs) [CRR02] require a profiling stage in which the adversary estimates the probability distribution of the leakage to use information present in each sample. Traditionally, TAs are considered the best approach from an information-theoretic point of view, but they make strong assumptions on the attacker, e.g., that they can collect an unbounded number of template traces and that the noise is close to Gaussian.

Medwed and Oswald [MO09] introduced a TA targeting the state of the ECC ladder after a few scalar bits were processed. In our context if a single-trace TA is successful then the few bits are recovered and the attack continues on the subsequent bits targeting the same trace. Another relevant profiling attack is the TA targeting the conditional move (cmov) instruction [NCOS17]. 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 of TAs targets key-transfer [OP11]. In most implementations the scalar is copied, from flash to RAM, just before the scalar multiplication is executed. This might allow an attacker to template the copy process and recover information about the scalar or the session key (i.e., the scalar-multiplication output) from a single trace.

Deep-learning attacks. A recent class of profiled attacks, similar to TAs, are deep-learning (DL) side-channel attacks [CDP17, KPH+19, MPP16]. 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 [WPB19, WCPB20, ZS19]. Deep learning was also used to attack ECC in the unsupervised setting [PCBP21].

Online template attacks (OTA) use horizontal techniques to exploit the fact that an internal state of a scalar multiplication depends only on the scalar and the known input [BCP+17, DPN+16, RSBD20] or output [AB21]. Advanced types of OTA need only one leakage trace and can defeat implementations protected with scalar blinding or splitting.

OTA traces back to and extends the doubling and collision attacks [FV03, HMA+08]. 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 [Cor99] or re-randomization [NLD15].

Horizontal cmov attacks [NC17] are similar to the single-trace TA [NCOS17], but are unsupervised. They combine a heuristic clustering approach with multiple-trace points-of-interest selection. Due to a more relaxed attack setting, these attacks have slightly lower success rate and their complexity is increased in comparison to the profiled approaches.

Soft analytical side-channel attack (SASCA) [VCGS14] combines SCA with the optimal data complexity of algebraic attacks and it was applied to ECC in the single-trace setting [ADP+20]. While this approach is promising, it has, as far as we are aware, so far only been experimentally executed on an 8-bit architecture in the single-trace setting. Therefore, we do not consider this attack in our evaluation and leave application to our 32-bit implementations as future work.

3.2 Relevant active attacks

Weak-curve attacks. The first weak-curve attack [BMM00] 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 provides a point P on an insecure curve E' and then solves the 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 [BMM00], invalid curves [CJ05], and twist curves [FLRV08] to hit a weak curve.

Loop-abort attacks. A rather simple FI 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 is a simple attack aiming 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 output aims at enforcing predictable output to prevent copying of the final result or use a fault to directly write constant values, e.g., zeroes, to the output buffer.

3.3 Protected implementation

We extend the approach from Algorithm 1 with SCA and FI protections. Algorithm 2 presents pseudocode of our protected implementation of ephemeral X25519. Like most previous X25519 implementations it is running in "constant time", i.e., it avoids secret branch conditions and secretly indexed memory access. 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 swaps 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 non-zero 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.

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: $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ 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: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(\mathsf{montdouble}(\mathsf{montdouble}(x_p, 1)))$ ▷ 3 doublings to multiply by co-factor 8
7: `Increase(`ctr`)`
8: **if** $Z_P = 0$ **then**
9: **go to** Line 23 ▷ Early-abort if input point is in order-8 subgroup
10: $x_P \leftarrow X_P \cdot Z_P^{-1}$ ▷ Return to affine x-coordinate
11: $X_1 \leftarrow 1, Z_1 \leftarrow 0$
12: $Z_2 \xleftarrow{\$} \{0, \ldots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$ ▷ Initial randomization of projective representation
13: $k \leftarrow k \oplus 2k$ ▷ Precompute condition bits for cswap
14: `Increase(`ctr`)`
15: **for** i from 252 downto 1 **do** ▷ Main scalar-multiplication loop
16: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, k[i])$ ▷ Projective re-randomization merged with cswap
17: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}\ (x_P, X_1, Z_1, X_2, Z_2)$
18: `Increase(`ctr`)`
19: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ ((X_1, Z_1, X_2, Z_2), k[0])$
20: $x_P \leftarrow X_2 \cdot Z_2^{-1}$
21: `Increase(`ctr`)`
22: **if** `! Verify(`ctr`)` **then** ▷ Detected wrong flow, including iteration count
23: $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ Set output buffer to random bytes
24: **return** x_P

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 and M_2 having random data in bits 16 to 31. Bits 15 to 1 are zeroed and the condition (M_1) and negated condition (M_2) is held in bit 0. For inputs A and 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.

To protect against FI, e.g., loop-abort attacks, we employ a flow-counter [Wit18]. Specifically, we use a single counter monotonously incremented throughout the scalar multiplication to detect changes in the execution flow. If the counter value does not match the expected constant 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: while it is less protected than the static one, it achieves much faster running time. In particular, we do not employ address randomization, storage and scalar blinding (for details, see Section 4). However, it would be easy to add some of those countermeasures from the implementation for the static use-case. Of course, one can also simply use the static routine in the ephemeral scenario; this comes at the cost of slowing down the computation by a factor of ≈ 2.5 (for exact numbers, see Table 1).

Below we explain how our implementation addresses the attack threats 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) does not use conditional branches. In this context, using a ladder is a commonly recommended countermeasure against "classical" SPA [KJJ99].

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, SASCA, and online template attacks extract information about the scalar by identifying temporary values used in the computation through matching against templates. As recommended in [BCP+14, BCP+17], we employ a countermeasure against online-template attacks: projective randomization [NSS04] with a randomizer chosen uniformly at random from $\mathbb{F}_p$ and we return only affine output. This countermeasure also mitigates TAs exploiting leakage of the internal ladder. Note that our sampling routine for the randomizer can in principle return a zero value; however the probability of this happening is smaller than an attacker guessing the secret key.

Our protections for the ephemeral use case do not stop all single-trace attacks. Specifically, the following attacks might still be feasible: single-trace TA or DL targeting cswaprr (see, e.g., [NCOS17]) and single-trace TA or DL targeting key loading. Potentially also single-trace SASCA might be possible. Our static implementation adds additional protections against such attack; for details see Section 4.

Weak-curve attacks. Since we are implementing Curve25519, many weak-curve attacks are not possible; see the paragraph on small-subgroup attacks in [Ber06, 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 are thwarted by employing the flow-counter countermeasure [Wit18].

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 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 combine SCA and FI attacks, as presented in [FGV11], 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 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 attack 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 protected implementation of X25519 attacks when deployed in a *static* key-exchange scenario. In this scenario the secret scalar can be used an arbitrary number of times, contrary to the ephemeral scenario 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 and then stored in the memory of the library. The static masks are then 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 (Subsection 4.1) and active attacks (Subsection 4.2). Note that these lists are extensions to the lists from Section 3. Moreover, since the static implementation employs 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 [KJJ99] 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 ECDH-based 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 exists higher-order multivariate DPA [Mes00, CJRR99], which combines multiple samples coming from many traces to derive side-channel information.

Cross-correlation attacks exploit collisions in subsequent additions and doubling of a scalar multiplication algorithm using many traces [WvWM11]. The horizontal-correlation attacks from Section 4.2 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 [Gou02]. A zero-value point attack (ZPA) [AT03] 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 [FRVD08] do not attack the scalar multiplication itself but its countermeasures. They rely 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. Since k_i is fixed and $r_i\#E$ is random in different executions, the probability of the carry out $c = 1$ depends mainly on k_i when

adding k_i to $r_i \# E$. The attacker can estimate the probability of $c = 1$ by monitoring the outgoing carry bit of the adder. With this, the value of k_i can be guessed reliably.

Address-bit DPA [Cor99] exploits the link between the address and the key, and it was applied to ECC [IIT03a]. This attack is applicable if addresses of coordinates processed in a ladder step somehow depend on the corresponding scalar bit. Essentially, the scalar can be recovered if the attacker distinguishes 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 [NCOS17] (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 the cswap leakage in this context.

4.2 Relevant active attacks

Safe-error attacks were introduced by Yen and Joye [YJ00, JY03]. Two types of such attacks have been reported: C and M safe-error attacks. The C safe-error attack exploits dummy operations which are usually introduced to achieve SPA resistance, for example, in double-and-add-always. The adversary tries to induce temporary faults during the execution of the dummy operation. If the result is unaffected then it means that the fault hit a dummy operation; if the result changes it means that a real operation was affected. This is enough to learn a scalar bit in the attacked iteration. M safe-error attacks exploit the fact that faults in some memory blocks will be cleared. These attacks were first applied to RSA [JY03], but they can also be applied to scalar multiplication. The goal of the attack is to affect memory that is overwritten only if a scalar bit has a certain value.

Differential fault analysis (DFA) on scalar multiplication [BMM00, BOS06] uses the difference between correct and faulty outputs to deduce certain scalar bits. These attacks require multiple correct and incorrect outputs to learn the scalar. Since, as described later, we randomize the scalar, DFA is is not applicable, we do not discuss it in more detail here.

4.3 Protected implementation

The scalar multiplication with protections for the static use-case 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 non-zero 64-bit random blinding factor. 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 (EEA). We protect these computations by additional multiplicative blinding. We could alternatively use the constant-time EEA by Bernstein and Yang [BY19], but this would be less efficient, and our implementation anyway requires a source of randomness.

To limit address leakage we implement the address-randomization technique [IIT03b, HMH+12]. This countermeasure adds additional random cswaprr executions to make the

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

Input: the x-coordinate x_P of point P. **Output:** $x_{[k]P}$.
Secure Input: a 64-bit blinding f, blinded scalar $k_{f-1} = k \cdot f^{-1} \mod l$, and blinding points $R, S = [-k]R$.

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

cswaprr sequence in the scalar multiplication independent of the scalar itself. Although this countermeasure is called address-randomization for historical reasons, it can also thwart cswap-like leakage as shown in [HMH+12].

Below we explain how our implementation addresses the attack threats listed above.

First-order and higher-order DPA attacks. To prevent DPA, Coron [Cor99] suggested three countermeasures for an EC-based key-exchange: randomizing a point, randomizing projective coordinates, and randomizing 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 the we employ scalar randomization,

a higher-order multivariate DPA is mitigated since it is not possible to make an assumption about the scalar used in different executions of the scalar multiplication.

Cross-correlation attacks are reduced to horizontal-correlation attacks by scalar blinding. Horizontal-correlation attacks, in turn, are prevented by projective re-randomization.

Special-point attacks are prevented by the initial point blinding, i.e., computing $S = [-k]R$. 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 after every usage and therefore, no single guess about k or k_i can be made between multiple blinding computations.

Address-bit DPA is infeasible due to scalar blinding in every scalar multiplication.

Address template attacks. Only single-trace attacks are applicable due to the employed randomizations. The single-trace horizontal, template, and deep learning attacks can directly target whatever instructions involve the scalar bits. Therefore, these attacks are much harder to defeat. To limit their effectiveness we implement address randomization [IIT03b, HMH+12] and we evaluate this protection in Section 5.2.

Another effective countermeasure against these attacks was implemented for Curve25519 `cswap` [LLF20]. 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 DFA are not applicable due to the usage of scalar blinding.

5 Evaluation

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 the 24 MHz frequency for obtaining reproducible cycle counts that are not significantly affected by wait cycles of the memory subsystem [HL19]. For all our measurements (including libraries we compare our library to), we use the gcc compiler, arm-none-eabi-gcc version 9.2.1 20191025, 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. The performance evaluation results are presented in Table 1.

As expected, we see that all protections come at a cost. However, the protections for the ephemeral implementation, so mostly the projective re-randomization, only incur a small performance penalty. The additional protections included in the static implementation, mainly against single-trace profiled attacks, have a more significant cost even when implemented efficiently. The total overhead for the ephemeral implementation is about 37% and the overhead for the static one is 243% in comparison to the unprotected case.

Table 1: Performance Evaluation.

Implementation or Countermeasure	Cycles:
Complete unprotected [Len20]:	548 873
Our unprotected baseline [HL19]:	680 097
Complete ephemeral:	932 204
Complete static:	2 338 681
Single cswaprr (ephemeral & static):	1047
Randomized Addressing (static):	334 961
Update static blinding points (static):	153 071
Update static scalar (static):	211 468
Blinding of the scalar (static):	207 434
Additional 64-bit scalar multiplication (static):	332 347

Table 2: Performance of Related Crypto Libraries.

Library	Const. Time	Cycles:
wolfSSL [wol21] size:	yes	45 930 947
wolfSSL [wol21] speed:	yes	1 974 047
bearSSL i31 [Tho18]:	yes	2 576 639
BoringSSL [Goo21]: fixed base	yes	1 591 407
BoringSSL [Goo21]: var. base	yes	2 516 476
Arm MBed TLS [Arm21]	no	6 438 233

Comparison. Our countermeasures significantly reduce the speed, but we believe that our implementations are highly competitive; at least for users who are willing to sacrifice portability for SCA and FI resistance. Table 2 reports the cycle counts, obtained using the same Arm Cortex-M4 from several commonly used speed-optimized versions of unprotected off-the-shelf cryptographic libraries. As we can see, switching from any of these libraries to even our implementation with protections in the static use case will not result in a major decrease in performance. However, as these libraries are written in C, their code is more portable and easier to maintain than our implementation that makes heavy use of hand-optimized assembly.

Security properties of related crypto-libraries. None of the other libraries we benchmarked exceeds the protection level of conventional constant-time execution. We assess that Arm Mbed TLS even fails to provide constant execution timing as detailed below. Regarding the security properties, BoringSSL[6] and bearSSL are roughly comparable to our unprotected implementation, 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 find suitable for smaller embedded targets due to its code-size. All other implementations use the Montgomery ladder on the Montgomery curve in projective coordinates. The libraries differ in their implementation of field arithmetic, and in 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 are merged. 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 countermeasures against the types of SCA we consider. Arm Mbed TLS on the other hand integrates projective randomization also for public input points. As Arm MBed TLS uses a variable-time early-abort multiplication strategy we presume that this projective randomization was meant as a mitigation against simple timing-based SPA attacks. Unfortunately, as input points are not validated, this does not actually work. Simple timing attacks using the strategy used by Genkin, Valenta, and Yuval [GVY17] become feasible if the adversary inserts a low-order point, and then distinguishes multiplications by zero. 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 protecting against timing attacks.

We communicated our concerns to the developers of Arm Mbed TLS [Arm21] and suggested different mitigations. They acknowledged the issue and implemented countermeasures in version 3.0.0: https://github.com/ARMmbed/mbedtls/releases/tag/v3.0.0.

[6] We chose to analyze BoringSSL as one member of the of the OpenSSL library family. BoringSSL is a fork of OpenSSL and we did not observe any substantial differences between their X25519 implementations.

5.2 Side-channel Evaluation

This subsection presents our SCA setup and the evaluation of the unprotected, ephemeral, and static implementations. We also analyze the resistance to single-trace 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 [Ris18] and record traces using the LeCroy WaveRunner 610Zi oscilloscope. For analysis of the traces we use the Inspector software by Riscure [Ris21]. More practical considerations, like the signal-to-noise ratio of our setup, are presented in Appendix A.

We performed our leakage detection experiments in the following four scenarios: (1) the unprotected implementation, without any countermeasures except constant-time operations; (2) the ephemeral implementation, as in Algorithm 2; (3) the static implementation as in Algorithm 3, with all countermeasures enabled (projective re-randomization, randomized addressing, point, scalar, and storage blinding); and (4) a static implementation modified for the profiled-attacks evaluation with scalar and storage blinding disabled. We test setting 4 with and without randomized addressing to verify increased resistance to profiled attacks. Additionally, we perform several single-trace multivariate template attacks in setting 4 to check the resistance of our solution against the profiled attack[7].

We apply a commonly-used TVLA methodology [GJJR11, JRW11, BCD+13, TG16] with 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, the TVLA confidence threshold to detect leakage in the t-test statistic was set to 4.5, which lead to many false positives [JRW11, TG16]. To avoid this issue we use a formula to compute a confidence threshold [DZD+17] following an evaluation approach for ECC [PFPB19]. For all our experiments the computed threshold is between 7 and 7.3 and therefore, we assume that peaks above 7 indicate leakage.

In general, our evaluation concentrates not only on the Montgomery ladder executions, but also on the whole 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.

We usually detect leakage only around the area that we align on.[8] This is caused by jitter and, especially for the static case, by inverse blinding. Therefore, it might happen that if the traces are aligned at the beginning then the leakage is only detected there, even if it is present everywhere. To avoid that, we align the traces in multiple locations (e.g., 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 first analyze the difference between groups 0 and 1. The goal is to evaluate the correlation of the implementation leakage to an input point; such leakage, if present, can be used to mount CPA, for example. 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. Finally, we perform several single-trace multivariate template attacks against the static implementation.

TVLA of the unprotected implementation. Figure 1 depicts the results of the t-tests for groups 0 and 1, each consisting of 1000 traces. Alignment at different locations and the test for groups 0 and 2 show similar results. The highest peak is reaching 71 and thus, we conclude that our unprotected baseline implementation leaks significantly.

[7] In particular, we attack intermediate values, like masks and masked scalar bits, to see whether we are able to obtain enough information to recover the scalar itself.

[8] We align the traces matching a distinctive pattern from the first trace to all subsequent traces by shifting them horizontally; to measure matching quality we use Pearson correlation.

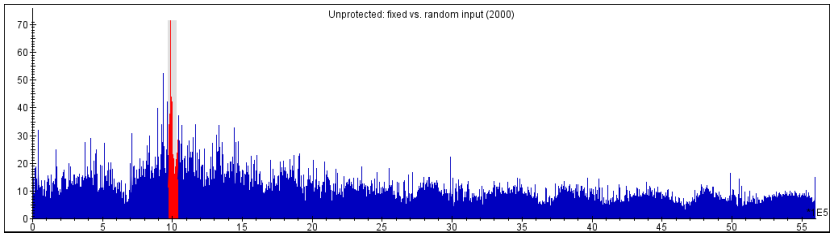

Figure 1: Unprotected imp. TVLA: fixed vs random point. Alignment is marked red.

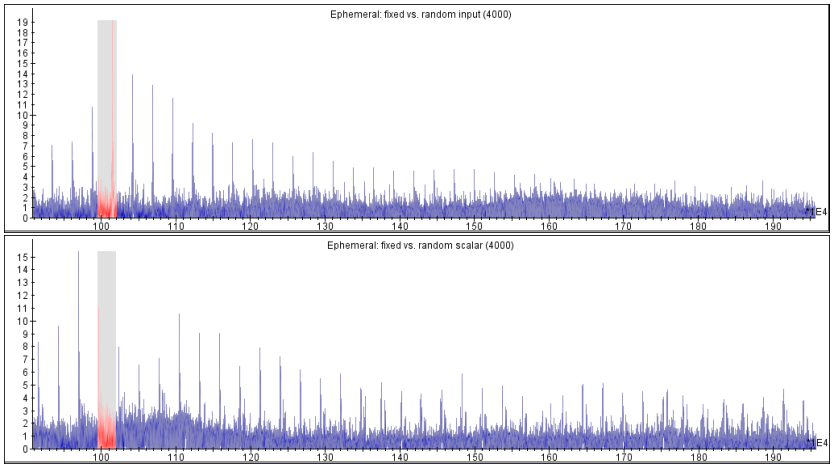

Figure 2: Ephem. imp. TVLA (0.9-2.0ms): fixed vs random point (top) & scalar (bottom).

TVLA of the ephemeral implementation. Similarly to the unprotected case we analyzed the ephemeral implementation, but this time due to the countermeasures, we expect less leakage. Therefore, we increased the size of each group to 2000. Figure 2 presents the t-test results and we observe less leakage than for the unprotected case. The highest peak reaches 15 for groups 0 and 2, but it is to be expected since the ephemeral scalar should vary between executions. However, as we see, if the same scalar is used multiple times, by mistake for example, then the leakage is present.

For groups 0 and 1 the highest peak reaches 19. This seems unexpected since we employ 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 TVLA limitation [BCD+13]. To validate the above hypothesis we 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 leakage of this operation is not exploitable since x_P is public and constant per execution and the other input `b1` is randomized. This modification is incorrect and we only used it for this experiment. The TVLA result for this modification shows no leakage as the highest peak does not reach 4.4. Therefore, we conclude that the first-order leakage is not present.

The leakage present for groups 0 and 2 suggests that the ephemeral implementation might be vulnerable to single-trace profiled attacks. Since we expected such attacks to be not easy, for the sake of efficiency we have decided not to protect the ephemeral implementations against these attacks. Instead we have protected only the static implementation against these attacks and we evaluate its resistance using TVLA and single-trace template attacks.

Furthermore, we do not evaluate the ephemeral implementation against single-trace

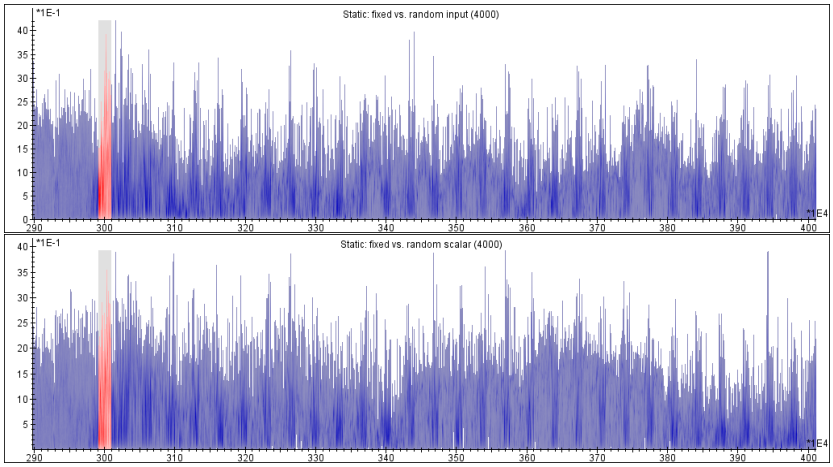

Figure 3: Static imp. TVLA (2.9-4.0ms): fixed vs random point (top) & scalar (bottom).

attacks since the static evaluation gives us a bound on the success rate also for the ephemeral case. Essentially the address-randomization mask in the static implementation is used similarly as the ephemeral scalar, but cswaprr is performed twice for the mask and not only once. Therefore, the single-trace TA for the ephemeral scalar should have lower success rate than the TA against the mask. This success rate, as shown below, is 64%.

TVLA of the static implementation. First we aligned the traces at the beginning of the execution. For groups 0 and 1 we notice leakage very early, when the input point is processed. For groups 0 and 2 we detect no leakage due to the scalar being stored blinded.

For the static traces we notice significant misalignments mainly due the used countermeasures. Thus, we performed another t-test for the traces aligned at the beginning of the main scalar-multiplication, as presented in Figure 3. The plots in the figure are zoomed in around the alignment moment (marked red). Since the peaks in both experiments are below 4.3, we conclude that no leakage is detected. We also aligned at various other locations during both scalar multiplications and in all cases no leakage is detected.

However, we detected leakage after the scalar multiplications are finished (around 14.5ms). This is caused by the computation of the final affine output. Afterwards, 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 trace we detect the leakage corresponding to sending the output. Such output leakage might be used for a single-trace attack to recover the output point (and effectively the session key). Although this may be possible, we cannot avoid that 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.

TVLA for profiled attacks. A relevant profiling attack in our context is a TA targeting the cswap [NCOS17]. In our context this attack is not easy because both, the blinded scalar and the additional 64-bit scalar blinding, would need to be recovered from a single trace. First we verified our implementation's resistance by performing TVLA on an implementation with all scalar randomizations turned off. In this setting we checked whether turning on the randomized addressing prevents scalar leakage from occurring. These experiments checked whether the profiling phase of TA detects leakage; in particular no leakage means that the attacker cannot determine points of interest for mounting a single-trace TA.

Figure 4 depicts the t-test results for groups 0 and 2 when the scalar and address

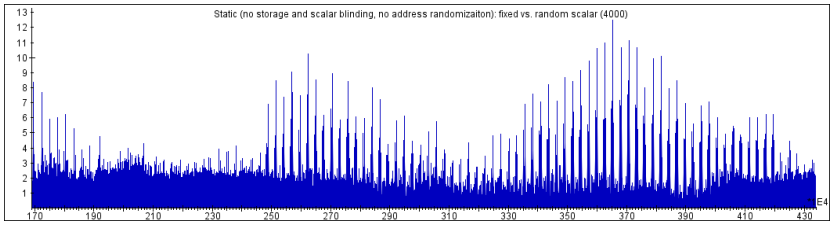

Figure 4: Static implementation TVLA (no blindings and address randomization) aligned at 1.65ms (1.7-4.3ms): fixed vs random scalar.

randomizations are turned off: the highest peak reaches 12.5. This leakage might be used by the aforementioned TA to recover the scalar from cswaprr. To validate whether the address randomization works correctly we turned it on and discovered no leakage.

Single-trace template attacks. Although we discovered that the leakage exploitable by the "standard" TA targeting single cswaprr to recover the scalar bit directly is not present, a more complex attack might still be possible. We evaluated this possibility by performing TAs targeting various sensitive intermediate values processed in our implementation: (1) address-randomization mask bits, (2) masked scalar bits, (3) plain scalar bits. If we can recover correctly (1) and (2) then we can xor these values to recover the scalar (3); we attack (3) to verify whether the implementation can be broken directly.[9] We perform multivariate TAs where we target multiple points in time corresponding to various operations on sensitive intermediates to maximize chance of success.[10] Since to the best of our knowledge there were no attacks like that performed against a 32-bit architecture, the results presented here are preliminary and we consider extending them to be future work.

We perform template attacks in the following setting: we use 50 000 traces for learning and 10 000 for attacking. Since leakage of the intermediate values often occurs in multiple iterations of scalar multiplications (this is explainable since the scalar and the mask are accessed in 32-bit words) or even outside scalar multiplication (e.g., during mask generation), we do not cut traces to pieces corresponding to iterations like in [NCOS17], but we learn and attack the whole traces depending on detected points of interests (POIs). We find POIs using a commonly used method, namely, sum-of-squared t-values (SOST)[11]. By repeating the template attack we discovered that the best result are obtained for 2500 POIs for (1), 10 for (2), and 100 for 3. We repeated the experiments for the first few bits of the targeted intermediates and the results were consistent.

Figure 5 presents the POIs detection plot for the most significant bit of the mask. The first region marked in red corresponds to the mask generation and the second black one to a part of the scalar-multiplication iteration. Since both regions contain high SOST peaks (i.e., 150 and 40 respectively), the single-trace attack is relatively successful: 63-64%. As mention before this is also an upper bound on a single-trace TA on the ephemeral scalar.

For the (2) case the success probability is only 50.5% while for (3) it is 52%. We were not able to increase the probabilities even when trying various alignments and TA parameters. This is due to the fact that the SOST peaks were relatively low: approximately

[9] Observe that when attacking an implementation in the wild we would need to recover a multiplicatively blinded scalar and the corresponding blinded 64-bit scalar from a single trace. From these two values, we can recover the secret scalar. Here for the sake of simplicity we omit this aspect.

[10] Note that we do not partition the traces to many segments and analyze them, since we aim at combining the leakage coming from various usages of sensitive values and it is unclear what such a partition would look like in such a setting. We simply collect more traces to amortize for not partitioning a single trace to multiple segments. This way we also avoid an issue of potential misalignment of the segments.

[11] We also follow a common technique not to choose adjacent POIs. In our case, based on experiments, we discovered that it is the best to keep at least 3 samples distance.

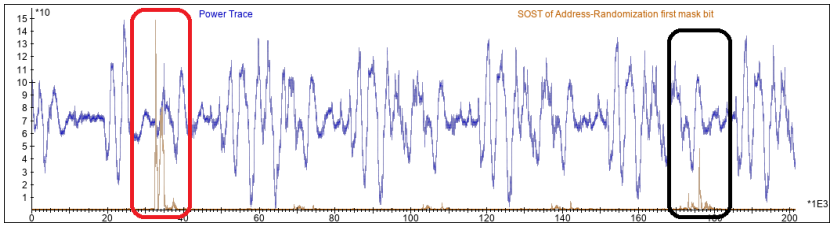

Figure 5: Power trace and the corresponding SOST for address-randomization mask.

20 for the (2) case and 30 for the (3) case. Slightly higher leakage for (2) can be explained by the fact that the scalar is processed by multiple instructions during masking of the scalar and the masked scalar is only saved once in 32-bit word.

The cswaprr operations alone seems to be leaking very little for our implementation and our setup; this is probably due to our non-optimal setup. Following [HMH+12], we envision that using electromagnetic probes, especially micro-probes and de-capped chips, would increase the quality of the signal and thus success rate of the single-trace attack. Furthermore, we believe that the countermeasure implemented in cswaprr, although imperfect, helps protecting against such TAs and it contributes to lower success rates.

The above results suggest that it is hard to attack the static implementation using a multivariate single-trace TA. In particular, attacking (3) directly has a success rate of only 52% and combining results for (1) and (2) gives an even worse result due to a low success rate for (2). Even if for some single traces the success rates for (1) and (2) would be significantly higher, similarly to the situation in [NCOS17], e.g., +20%, then the combined success rate would be low: $0.64 \approx (0.705 \cdot 0.84) + ((1 - 0.705) \cdot (1 - 0.84))$. This seems too low to recover the scalar, even if blinded scalars are somehow, similarly to [RIL19], combined to recover the static scalar.

6 Conclusions and Future Work

We have implemented software computing the X25519 key-exchange on the Arm Cortex-M4. This software comes with extensive protections against both side-channel and fault attacks while being at least as efficient as widely-deployed ECC libraries. Given the experimental evidence, we are carefully optimistic that the implementation is indeed secure within our attacker model, but we leave further investigation with an improved SCA setup (possibly with EM micro-probes) into applications of more complex single-trace profiled attacks[12] to future work. We encourage other researchers to attack our implementations in order to improve the community's understanding of SCA-protected ECC. Furthermore, we leave formal, ideally computer-verified proofs of SCA resistance of our implementations as an interesting direction of future research.

References

[AB21] Alejandro Cabrera Aldaya and Billy Bob Brumley. Online template attacks: Revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):28–59, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8967.

[12]We consider especially template attacks based on Linear Discriminant Analysis (LDA) [SA08] and deep-learning methods to be most promising.

[ABM+03] Adrian Antipa, Daniel Brown, Alfred Menezes, René Struik, and Scott Vanstone. Validation of elliptic curve public keys. In Yvo G. Desmedt, editor, *Public-Key Cryptography – PKC 2003*, volume 2567 of *LNCS*, pages 211–223. Springer, 2003. https://iacr.org/archive/pkc2003/25670211/25670211.pdf.

[ADP+20] Melissa Azouaoui, François Durvaux, Romain Poussier, François-Xavier Standaert, Kostas Papagiannopoulos, and Vincent Verneuil. On the worst-case side-channel security of ECC point randomization in embedded devices. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 205–227. Springer, 2020.

[AJB20] A. Adamantiadis, S. Josefsson, and M. Baushke. Secure shell (SSH) key exchange method using Curve25519 and Curve448. IETF RFC 8731, 2020. https://tools.ietf.org/html/rfc8731.

[Arm21] Mbed TLS, 2021. https://github.com/ARMmbed/mbedtls (accessed 2021-05-05).

[AT03] Toru Akishita and Tsuyoshi Takagi. Zero-value point attacks on elliptic curve cryptosystem. In Colin Boyd and Wenbo Mao, editors, *Information Security*, volume 2851 of *LNCS*, pages 218–233. Springer, 2003.

[Aum17] Jean-Philippe Aumasson. Should Curve25519 keys be validated?, 2017. https://research.kudelskisecurity.com/2017/04/25/should-ecdh-keys-be-validated/ (accessed 2020-12-02).

[AVL19] Rodrigo Abarzúa, Claudio Valencia, and Julio López. 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.

[BCD+13] Georg T. Becker, Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Timofei Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. Test vector leakage assessment (TVLA) methodology in practice. International Cryptographic Module Conference, 2013.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[BCP+14] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology – INDOCRYPT 2014*, volume 21–36 of *LNCS*, page 8885. Springer, 2014. http://cryptojedi.org/papers/#ota.

[BCP+17] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. *Journal of Cryptographic Engineering*, pages 1–16, 2017. http://cryptojedi.org/papers/#ota, see also short version [BCP+14].

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In Walter Fumy, editor, *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.

[BDL+11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, 2011. see also full version [BDL+12].

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012. http://cryptojedi.org/papers/#ed25519, see also short version [BDL+11].

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

[Ber06] Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006. http://cr.yp.to/papers.html#curve25519.

[Ber14] Daniel J. Bernstein. 25519 naming. Posting to the CFRG mailing list, 2014. https://www.ietf.org/mail-archive/web/cfrg/current/msg04996.html.

[BJP+15] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, Jean-René Reinhard, and Justine Wild. Horizontal collision correlation attack on elliptic curves – extended version –. *Cryptography and Communications*, 7(1):91–119, 2015. https://doi.org/10.1007/s12095-014-0111-8.

[BJPW13] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure rsa implementations. In Ed Dawson, editor, *Topics in Cryptology – CT-RSA 2013*, volume 7779 of *LNCS*, pages 1–17. Springer, 2013.

[BJPW14] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal collision correlation attack on elliptic curves. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *LNCS*, pages 553–570. Springer, 2014.

[BMM00] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *LNCS*, pages 131–146. Springer, 2000.

[BOS06] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. Sign change fault attacks on elliptic curve cryptosystems. In Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography*, volume 4236 of *LNCS*, pages 36–52. Springer, 2006.

[BY19] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):340–398, 2019. https://doi.org/10.13154/tches.v2019.i3.340-398.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *LNCS*, pages 45–68. Springer, 2017.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *Information and Communications Security*, volume 6476 of *LNCS*, pages 46–61. Springer, 2010. http://eprint.iacr.org/2003/237.

[CJ05] Mathieu Ciet and Marc Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, Codes and Cryptography*, 36:33–43, 2005.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO' 99*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.

[CMV+17] Łukasz Chmielewski, Pedro Maat Costa Massolino, Jo Vliegen, Lejla Batina, and Nele Mentens. 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.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Cetin K. Koc, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.

[DGH+13] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. *Journal of Cryptographic Engineering*, 3(4):1–25, 2013.

[DPN+16] Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, and Sylvain Guilley. Dismantling real-world ECC with horizontal and vertical template attacks. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design*, volume 9689 of *LNCS*, pages 88–108. Springer, 2016.

[Duo15] Thai Duong. 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).

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications*, volume 10728 of *LNCS*, pages 105–122. Springer, 2017.

[FA18] Hayato Fujii and Diego F. Aranha. Efficient Curve25519 implementation for ARM microcontrollers. In *Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 57–64. Sociedade Brasileira de Computaçcão, 2018. https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/4142/4071.

[FA19] Hayato Fujii and Diego F. Aranha. Curve25519 for the Cortex-M4 and beyond. In Tanja Lange and Orr Dunkelman, editors, *Progress in Cryptology – LATINCRYPT 2017*, volume 11368 of *LNCS*, pages 109–127. Springer, 2019. http://www.cs.haifa.ac.il/~orrd/LC17/paper39.pdf.

[FGM+10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. 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)*, pages 76–87. IEEE, 2010.

[FGV11] Junfeng Fan, Benedikt Gierlichs, and Frederik Vercauteren. To infinity and beyond: Combined attack on ECC using points of low order. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*, pages 143–159. Springer, 2011.

[FLRV08] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. Fault attack on elliptic curve with Montgomery ladder implementation. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 92–98. IEEE, 2008. https://www.di.ens.fr/~fouque/pub/fdtc08.pdf.

[Fou08] OPC Foundation. Unified architecture, 2008. https://opcfoundation.org/about/opc-technologies/opc-ua/ (accessed 2021-05-05).

[FRVD08] Pierre-Alain Fouque, Denis Réal, Frédéric Valette, and Mhamed Drissi. The carry leakage on the randomized exponent countermeasure. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *LNCS*, pages 198–213. Springer, 2008.

[FV03] Pierre-Alain Fouque and Frederic Valette. The doubling attack — Why upwards is better than downwards. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 269–280. Springer, 2003. www.ssi.gouv.fr/archive/fr/sciences/fichiers/lcr/fova03.pdf.

[FV12] Junfeng Fan and Ingrid Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In David Naccache, editor, *Cryptography and Security: From Theory to Applications*, volume 6805 of *LNCS*, pages 265–282. Springer, 2012.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. 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.

[Goo21] BoringSSL, 2021. https://github.com/boringssl/boringssl (accessed 2021-05-05).

[Gou02] Louis Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In Yvo G. Desmedt, editor, *Public Key Cryptography – PKC 2003*, volume 2567 of *LNCS*, pages 199–211. Springer, 2002.

[GPPT16] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. ECDH key-extraction via low-bandwidth electromagnetic attacks on PCs, 2016. https://web.eecs.umich.edu/~genkin/papers/ecdh.pdf.

[GVY17] Daniel Genkin, Luke Valenta, and Yuval Yarom. 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*, pages 845–858. ACM, 2017.

[HKT15] Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In Kaisa Nyberg, editor, *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *LNCS*, pages 431–448. Springer, 2015.

[HL17] Björn Haase and Benoît Labrique. Making password authenticated key exchange suitable for resource-constrained industrial control devices. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *LNCS*, pages 346–364. Springer, 2017.

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384.

[HMA+08] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. Collision-based power analysis of modular exponentiation using chosen-message pairs. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *LNCS*, pages 15–29. Springer, 2008. http://www.aoki.ecei.tohoku.ac.jp/crypto/pdf/CHES2008_homma.pdf.

[HMH+12] Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. Localized electromagnetic analysis of cryptographic implementations. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *LNCS*, pages 231–244. Springer, 2012.

[Hoe15] Jochen Hoenicke. Extracting the private key from a trezor ... with a 70$ oscilloscope, 2015. https://jochen-hoenicke.de/crypto/trezor-power-analysis/ (accessed 2012-12-02).

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

[IEC13] Industrial communication networks – network and system security – part 3-3: System security requirements and security levels. Standard, International Electrotechnical Commission, 2013. https://webstore.iec.ch/preview/info_iec62443-3-3%7Bed1.0%7Db.pdf.

[IIT03a] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 129–143. Springer, 2003.

[IIT03b] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. A practical countermeasure against address-bit differential power analysis. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 382–396. Springer, 2003.

[ISO16] 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.

[JRW11] Josh Jaffe, Pankaj Rohatgi, and Marc Witteman. Efficient side-channel testing for public key algorithms: RSA case study. Technical report, Cryptography Research Inc. & Riscure, 2011. http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/09_Jaffe.pdf.

[JSSS20] Jan Jancar, Vladimir Sedlacek, Petr Svenda, and Marek Sys. 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):281–308, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8684.

[JY03] Marc Joye and Sung-Ming Yen. The Montgomery powering ladder. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer, 2003.

[KJJ99] Paul C. Kocher, Jushua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996. http://www.cryptography.com/public/pdf/TimingAttacks.pdf.

[KPF+16] Michael Kasper, Richard Petri, Dirk Feldhusen, Max Gebhardt, Georg Illies, Manfred Lochter, Oliver Stein, Wolfgang Thumserang, and Guntram Wicke. 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).

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. 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):148–179, 2019. https://doi.org/10.13154/tches.v2019.i3.148-179.

[Kra03] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated diffie-hellman and its use in the IKE protocols. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, 2003. https://webee.technion.ac.il/~hugo/sigma-pdf.pdf.

[KRSS19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. Workshop Record of the Second PQC Standardization Conference, 2019. https://cryptojedi.org/papers/#pqm4.

[Len20] Emil Lenngren. X25519 for arm cortex-m4 and other arm processors. GitHub repository, 2020. https://github.com/Emill/X25519-Cortex-M4.

[LLF20] Antoine Loiseau, Maxime Lecomte, and Jacques J. A. Fournier. Template attacks against ECC: practical implementation against curve25519. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 13–22. IEEE, 2020.

[LLP+17] Zhe Liu, Patrick Longa, Geovandro C. C. F. Pereira, Oscar Reparaz, and Hwajeong Seo. Four$\mathbb{Q}$ on embedded devices with strong countermeasures against side-channel attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *LNCS*, pages 665–686. Springer, 2017. https://eprint.iacr.org/2017/434.

[LMSS15] Manfred Lochter, Johannes Merkle, Jörn-Marc Schmidt, and Torsten Schütze. 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.

[Mar13] Moxie Marlinspike. Simplifying OTR deniability, 2013. https://signal.org/blog/simplifying-otr-deniability/ (accessed 2021-05-05).

[Mat16] Nick Mathewson. Cryptographic directions in tor. Presentation at Real-World Crypto 2016, 2016. https://rwc.iacr.org/2016/Slides/nickm-rwc-presentation.pdf.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology – CRYPTO '85: Proceedings*, volume 218 of *LNCS*, pages 417–426. Springer, 1986.

[MO09] Marcel Medwed and Elisabeth Oswald. Template attacks on ECDSA. In Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors, *Information Security Applications*, volume 5379 of *LNCS*, pages 14–27. Springer, 2009.

[Mon87] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987. http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866113-7/S0025-5718-1987-0866113-7.pdf.

[MOV93] Alfred J. Menezes, Tatsuaki Okamoto, and Scott Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *Transactions on Information Theory*, 39(5):1639–1646, 1993.

[MOW17] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In *Proceedings of the 26th USENIX Security Symposium*, pages 199–216. USENIX, 2017. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-mccann.pdf.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering*, volume 10076 of *LNCS*, pages 3–26. Springer, 2016.

[NC17] Erick Nascimento and Łukasz Chmielewski. Applying horizontal clustering side-channel attacks on embedded ECC implementations. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications*, volume 10728 of *LNCS*, pages 213–231. Springer, 2017.

[NCOS17] Erick Nascimento, Łukasz Chmielewski, David Oswald, and Peter Schwabe. Attacking embedded ECC implementations through cmov side channels. In Roberto Avanzi and Howard Heys, editors, *Selected Areas in Cryptology – SAC 2016*, volume 10532 of *LNCS*, pages 99–119. Springer, 2017. https://cryptojedi.org/papers/#cmovsca.

[NJPG18] Y. Nir, S. Josefsson, and M. Pegourie-Gonnard. 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.

[NLD15] Erick Nascimento, Julio López, and Ricardo Dahab. Efficient and secure elliptic curve cryptography for 8-bit AVR microcontrollers. In Rajat Subhra Chakraborty, Peter Schwabe, and Jon Solworth, editors, *Security, Privacy, and Applied Cryptography Engineering*, volume 9354 of *LNCS*, pages 289–309. Springer, 2015.

[NSS04] David Naccache, Nigel P. Smart, and Jacques Stern. Projective coordinates leak. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 257–267. Springer, 2004. https://www.iacr.org/archive/eurocrypt2004/30270258/projective.pdf.

[NXP15] 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.

[OP11] David Oswald and Christof Paar. Breaking Mifare DESFire MF3ICD40: Power analysis and templates in the real world. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*, pages 207–222. Springer, 2011.

[PCBP21] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek. Keep it unsupervised: Horizontal attacks meet deep learning. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):343–372, 2021. https://doi.org/10.46586/tches.v2021.i1.343-372.

[PFPB19] Louiza Papachristodoulou, Apostolos P. Fournaris, Kostas Papagianopoulos, and Lejla Batina. Practical evaluation of protected residue number system scalar multiplication. *IACR Transactions on Cryptographic Hardware*

and Embedded Systems, 2019(1):259–282, 2019. https://doi.org/10.13154/tches.v2019.i1.259-282.

[Pol78] John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978. http://www.ams.org/journals/mcom/1978-32-143/S0025-5718-1978-0491431-9/S0025-5718-1978-0491431-9.pdf.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, 2013.

[PZS17] Romain Poussier, Yuanyuan Zhou, and François-Xavier Standaert. A systematic approach to the side-channel analysis of ECC implementations with worst-case horizontal attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *LNCS*. Springer, 2017.

[RCB16] Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 403–428. Springer, 2016.

[RIL19] Thomas Roche, Laurent Imbert, and Victor Lomné. Side-channel attacks on blinded scalar multiplications revisited. In Tim Güneysu Sonia Belaïd, editor, *CARDIS 2019 - 18th Smart Card Research and Advanced Application Conference*, volume 11833 of *LNCS*, pages 95–108. Springer, 2019. https://hal-lirmm.ccsd.cnrs.fr/lirmm-02311595/file/blinded_scalar.pdf.

[Ris18] Riscure. Current probe. security test tool for embedded devices, 2018. https://www.riscure.com/product/current-probe/ (accessed 2021-05-05).

[Ris21] Riscure. Side channel analysis security tools, 2021. https://www.riscure.com/security-tools/inspector-sca/.

[RLMI21] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to Titan. In *30th USENIX Security Symposium, (USENIX Security 2021)*, pages 231–248. USENIX Association, 2021.

[RSBD20] Niels Roelofs, Niels Samwel, Lejla Batina, and Joan Daemen. Online template attack on ecdsa: Extracting keys via the other side. In Abderrahmane Nitaj and Amr Youssef, editors, *Progress in Cryptology – AFRICACRYPT 2020*, volume 12174 of *LNCS*, pages 323–336. Springer, 2020.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *LNCS*, pages 411–425. Springer, 2008.

[SBB+18] Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. Breaking Ed25519 in WolfSSL. In Nigel P. Smart, editor, *Topics in Cryptology – CT-RSA 2018*, volume 10808 of *LNCS*, pages 1–20. Springer, 2018. https://eprint.iacr.org/2017/985.

[SG15] Pascal Sasdrich and Tim Güneysu. Implementing Curve25519 for side-channel-protected elliptic curve cryptography. *Transactions on Reconfigurable Technology and Systems*, 9(1):1–15, 2015.

[SG17] Pascal Sasdrich and Tim Güneysu. 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*, pages 1–6. IEEE, 2017.

[Shi21] Anton Shilov. 842 chips per second: 6.7 billion arm-based chips produced in q4 2020, 2021. https://www.tomshardware.com/news/arm-6-7-billion-chips-per-quarter.

[Sma99] Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12(3):193–196, 1999.

[SS16] Fabrizio De Santis and Georg Sigl. Towards side-channel protected x25519 on arm cortex-m4 processors, 2016.

[SSB+21] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. ROSITA: Towards automatic elimination of power-analysis leakage in ciphers. In *Network and Distributed System Security Symposium (NDSS)*, pages 1–17. Internet Society, 2021. https://www.ndss-symposium.org/ndss-paper/rosita-towards-automatic-elimination-of-power-analysis-leakage-in-ciphers/.

[STM18] STMicroelectronics. Reference manual – STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced arm-based 32-bit MCUs. Technical Report 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.

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

[TG16] Michael Tunstall and Gilbert Goodwill. Applying tvla to public key cryptographic algorithms. Cryptology ePrint Archive, Report 2016/513, 2016. http://eprint.iacr.org/2016/513.

[Tho18] Bearssl, 2018. https://bearssl.org/ (accessed 2021-05-05).

[Tre18] The Noise protocol framework (revision 34), 2018. https://noiseprotocol.org/noise.pdf.

[TSW16] Niek Timmers, Albert Spruyt, and Marc Witteman. Controlling PC on ARM using fault injection. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC*, pages 25–35. IEEE, 2016.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 282–296. Springer, 2014.

[vOW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999. http://www.scs.carleton.ca/~paulv/papers/JoC97.pdf.

[Wal01] Colin D. Walter. Sliding windows succumbs to big mac attack. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 286–299. Springer, 2001.

[WCPB20] Leo Weissbart, Lukasz Chmielewski, Stjepan Picek, and Lejla Batina. Systematic side-channel analysis of curve25519 with machine learning. *Journal of Hardware and Systems Security.*, 4:314–328, 2020. https://doi.org/10.1007/s41635-020-00106-w.

[Wit18] Marc Witteman. Secure application programming in the presence of side channel attacks. Technical report, Riscure, 2018. https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf.

[WO19] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of iso 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, volume 11923 of *LNCS*, pages 256–284. Springer, 2019.

[wol21] The wolfSSL crypto library as accessed at april 24th, 2021), 2021. https://github.com/wolfSSL/wolfssl/tree/master/wolfcrypt.

[WPB19] Léo Weissbart, Stjepan Picek, and Lejla Batina. One trace is all it takes: Machine learning-based side-channel attack on EdDSA. In Shivam Bhasin, Avi Mendelson, and Mridul Nandi, editors, *Security, Privacy, and Applied Cryptography Engineering*, volume 11947 of *LNCS*, pages 86–105. Springer, 2019.

[WvWM11] Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. Defeating RSA multiply-always and message blinding countermeasures. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *LNCS*, pages 77–88. Springer, 2011. https://www.riscure.com/benzine/documents/rsacc_ctrsa_final.pdf.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *Transactions on Computers*, 49(9):967–970, 2000.

[YO19] Yan Yan and Elisabeth Oswald. Examining the practical side channel resilience of arx-boxes. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 373–379. ACM, 2019. https://eprint.iacr.org/2019/335.

[ZS19] Yuanyuan Zhou and François-Xavier Standaert. Simplified single-trace side-channel attacks on elliptic curve scalar multiplication using fully convolutional networks. Proceedings of the 40th WIC Symposium on Information Theory in the Benelux, 2019. https://perso.uclouvain.be/fstandae/PUBLIS/219.pdf.

A Supplementary Material for the SCA Setup

For our SCA experiments we used the Piñata board by Riscure[13], an STM32F407IGT6 development board[14] featuring an Arm Cortex-M4F core working at a clock speed of 168 MHz that has been physically modified to improve the quality of the measured current signal. In particular, multiple capacitors close to the main processor were removed. A frequently used technique is to lower frequency of the chip, but we did not attempt that, because this lowers the signal amplitude and that might decrease the amount of leakage.

For the sake of reproducibility, we include here the signal-to-noise ratio (SNR) of our setup. To avoid effects of our countermeasures we compute the SNR for the unprotected implementation for fixed input and key. To compute the SNR we used the following formula: $SNR[i] = |\mu[i]/\sigma[i]|$, where $\mu[i]$ and $\sigma[i]$ are the mean and the standard deviation at a sample index i of the traces. The resulting SNR is presented in Figure 6; the figure is zoomed at the beginning of the scalar multiplication where we aligned at $100\mu s$. As we can see the SNR is high (≈ 50), but gets lower when we are further from the aligned offset. Although we were aligning at various locations simultaneously, we believe that this effect might have affected the chance of multivariate attacks to succeed in our evaluation.

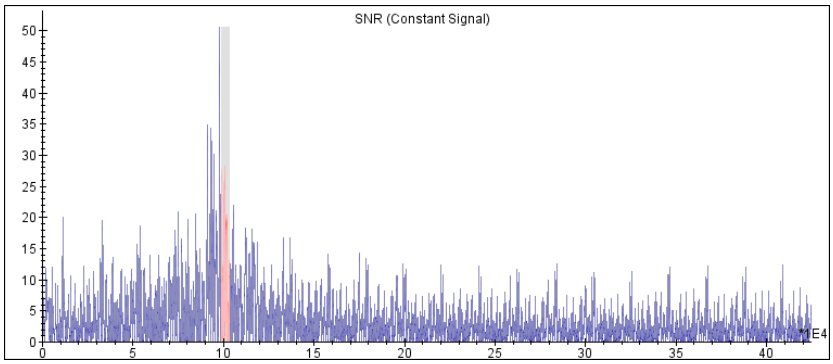

Figure 6: SNR for the unprotected implementation with fixed input and key.

As mentioned before, we used the `-O2` compilation flag in our experiments. Additionally, we performed every TVLA test without any optimizations (`-O0`), since that might inflate the existing leakage. In both cases the results were consistent with respect to detected leakage.

[13] https://www.riscure.com/product/pinata-training-target
[14] https://www.st.com/en/microcontrollers-microprocessors/stm32f407ig.html