IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 4, pp. 355–381. DOI:10.46586/tches.v2024.i4.355-381

pyecsca: Reverse engineering black-box elliptic curve cryptography via side-channel analysis

Jan Jancar[1], Vojtech Suchanek[1], Petr Svenda[1], Vladimir Sedlacek[2] and Łukasz Chmielewski[1]

[1] Masaryk University, Brno, Czechia
[2] Rutgers University, Piscataway, New Jersey

Abstract. Side-channel attacks on elliptic curve cryptography (ECC) often assume a white-box attacker who has detailed knowledge of the implementation choices taken by the target implementation. Due to the complex and layered nature of ECC, there are many choices that a developer makes to obtain a functional and interoperable implementation. These include the curve model, coordinate system, addition formulas, and the scalar multiplier, or lower-level details such as the finite-field multiplication algorithm. This creates a gap between the attack requirements and a real-world attacker that often only has black-box access to the target – i.e., has no access to the source code nor knowledge of specific implementation choices made. Yet, when the gap is closed, even real-world implementations of ECC succumb to side-channel attacks, as evidenced by attacks such as TPM-Fail, Minerva, the Side Journey to Titan, or TPMScan [MSE+20; JSS+20; RLM+21; SDB+24].

We study this gap by first analyzing open-source ECC libraries for insight into real-world implementation choices. We then examine the space of all ECC implementations combinatorially. Finally, we present a set of novel methods for automated reverse engineering of black-box ECC implementations and release a documented and usable open-source toolkit for side-channel analysis of ECC called **pyecsca**.

Our methods turn attacks around: instead of attempting to recover the private key, they attempt to recover the implementation configuration given control over the private and public inputs. We evaluate them on two simulation levels and study the effect of noise on their performance. Our methods are able to 1) reverse-engineer the scalar multiplication algorithm completely and 2) infer significant information about the coordinate system and addition formulas used in a target implementation. Furthermore, they can bypass coordinate and curve randomization countermeasures.

Keywords: elliptic curve cryptography · black-box implementations · reverse engineering · ECDH · ECDSA

1 Introduction

While elliptic curve cryptography (ECC) [Mil86; Kob87] is a popular choice for modern cryptosystems due to its performance and short keys, its implementations are usually quite complex. A developer needs to build the implementation from many blocks: big-integer arithmetic, finite-field arithmetic, addition formulas, coordinate system, curve model, scalar multiplier, and then finally, the cryptosystem itself. Each of these layers introduces choices to be made. For example, one implementation might pick Montgomery modular arithmetic [Mon85] with homogenous projective coordinates [Sil86], some addition formulas and a simple double-and-add-always scalar multiplier, while another implementation might pick the Barrett reduction with Jacobian coordinates [CC86] and a different combination of addition formulas and scalar multiplier. These two implementations, while significantly

Received: 2024-04-15 Accepted: 2024-06-15 Published: 2024-09-05

different on an internal level, can be fully interoperable with each other. Some implementation choices imply others, such as a coordinate system limiting the set of formulas available for that coordinate system, while others are essentially unconstrained, like the lower-level details of the arithmetic. Thus, given an implementation target like ECDH for prime-field curves like P-256, the space of possible implementation choices is large.

ECC has been the target of many side-channel attacks over the years, ranging from timing attacks and simple power analysis to complicated horizontal and vertical attacks [FGM+10; FV12; DGH+13]. Most of these require an attacker to have detailed information about the target implementation. For example, [BJP+15] state that 'The algorithm used for the hardware modular multiplication is assumed to be known to the attacker.' A survey of side-channel attacks on ECC by [DGH+13] lists the requirement for attackers to have 'full knowledge of all algorithms' for 4 out of 15 passive side-channel attacks, knowledge of the 'ECSM (Elliptic Curve Scalar Multiplier) and the elliptic curve formulas' is required by further 7 attacks, with 3 requiring just knowledge of the 'ECSM'.

In contrast to the attack assumptions presented in academic literature, real-world implementations of ECC are usually a black box from the perspective of an attacker. This holds especially when focusing on implementations in cryptographic hardware, such as smartcards, Hardware Security Modules, Trusted Platform Modules, or cryptocurrency wallets. A significant majority of these implementations are closed-source. The popular JavaCard platform [Ora23] is a clear example of this – it offers rich cryptographic APIs, including ECC [SKN+22]. The API implementation is left to vendors, who are motivated to reveal the least amount of information on their implementations to protect their intellectual property and satisfy certification requirements that encourage information hiding. In particular, Common Criteria [CC], a widely recognized international standard for evaluating and certifying the security of IT products, uses the Joint Interpretation Library attack rating [JIL] that encourages keeping knowledge about the implementation as secret as possible to increase required attack time. Open-sourcing a certified implementation would result in potential attacks being considered easier to execute and thus receiving a lower security rating.

As a result of these incentives, the secure hardware cryptography space is very different from the cryptographic theory space, as it violates Kerckhoffs's principle [Ker83] and partly bases its security on obscurity, inviting catastrophic issues. When obscurity dissipates – for example, by expert analysis or an information leak – security might be gone as well, as in the cases of the Infineon RSA library (ROCA [NSS+17]), Athena IDProtect smartcard (Minerva [JSS+20]) or NXP P5x secure elements (Side Journey to Titan [RLM+21]). This argument is especially clear in the case of the RSA library [NSS+17]: a flawed key generation algorithm allowed efficient factorization of public keys. While the researchers had to reverse-engineer this algorithm based solely on generated keypairs, it is reasonable to expect the vulnerability to be found promptly if the implementation was public.

In summary, side-channel attacks on ECC require knowledge of the implementation choices made by the target. Real-world implementations do not readily provide these choices but are often found vulnerable if reverse-engineered. We thus ask the following research questions:

RQ1: *What implementation choices are used in real-world open-source ECC libraries?*

RQ2: *How large is the space of all possible ECC implementations?*

RQ3: *Is it possible to automatically reverse-engineer black-box ECC implementations?*

Contributions. To answer our research questions, we contribute the following:

- Extensive analysis of 18 open-source ECC libraries, documenting a surprisingly wide variety of real-world ECC implementation choices (that we also expect from black-box implementations).
- Set of novel techniques for reverse engineering the coordinate system, addition formulas, and the scalar multiplier used in black-box ECC implementations – knowledge frequently necessary for a successful side-channel attack.
- Open-source toolkit **pyecsca**[1] for ECC side-channel analysis (including the above techniques), ECC implementation code generation and optimized trace processing.

Outline. In Section 2, we present a background on elliptic curve cryptography, side-channel attacks on ECC, and side-channel-based reverse engineering. A variety of implementation choices in open-source ECC libraries is documented in Section 3. We present automated techniques for reverse engineering of black-box ECC implementations in Section 4. Then, we showcase the functionality of our toolkit in Section 5. In Section 6 we evaluate the reverse-engineering techniques, and finally conclude in Section 7.

2 Background

In this section, we first give a background on elliptic curve cryptography, focusing on the broadness of implementations, and then describe selected side-channel attacks on ECC that we use for reverse engineering. Finally, we conclude with related work on reverse-engineering cryptographic implementations via side-channels.

2.1 Elliptic curve cryptography

For any prime $p \geq 5$, we consider an elliptic curve over $\mathbb{F}_p$ in three different models:

- Short-Weierstrass: $\mathcal{E}_{\mathrm{SW}} : y^2 = x^3 + ax + b$, $4a^3 + 27b^2 \neq 0$,
- Montgomery: $\mathcal{E}_{\mathrm{M}} : by^2 = x^3 + ax^2 + x$, $b(a^2 - 4) \neq 0$,
- Twisted Edwards: $\mathcal{E}_{\mathrm{TE}} : ax^2 + y^2 = 1 + dx^2y^2$, $ad(a - d) \neq 0$,

where the set of solutions $(x, y) \in \mathbb{F}_p^2$ together with a neutral point $\mathcal{O}$ form an additive group $E(\mathbb{F}_p)$. Most applications use curves with a large prime subgroup, i.e., $|E(\mathbb{F}_p)| = nh$, where n is prime and h is a small cofactor. Depending on the curve model and coordinate system used, the group operation can be defined in various ways using rational functions with coefficients from $\mathbb{F}_p$; many of these formulas are aggregated in the Explicit Formulas Database (EFD), curated by Bernstein and Lange [BL17].

The main ECC operation is scalar multiplication, denoted as $[k]P$ for a scalar k and a point P. It is the bottleneck for many ECC protocols, namely ECDH, X25519, ECDSA, and EdDSA, which are the main focus of this work. The Diffie-Hellman key exchange protocols, ECDH and X25519, consist of two subroutines. In the first subroutine, **KeyGen**, each party generates a random private key $0 < k < n$ and computes their public key $P = [k]G$, where G is a fixed generator of the prime subgroup. Algorithms for scalar multiplication can leverage precomputation on G (the so-called *fixed-base* approach). In the second subroutine, called **Derive**, each party computes $S = [k]P$, where P is the other party's public key, to obtain the shared key. Since the point P is unknown in advance, no precomputation is used (the so-called *variable-base* approach).

[1]Pronounced [pɪɛtska]. Code at https://github.com/J08nY/pyecsca, and documentation at https://pyecsca.org/.

The digital signature algorithms ECDSA and EdDSA use KeyGen, as described above, and two further subroutines. The **Sign** subroutine computes $[k]G$, where G is a generator and k is a random nonce in ECDSA and a hash output in EdDSA. Similarly to KeyGen, fixed-base techniques can be used. The **Verify** subroutine of both ECDSA and EdDSA requires *multi-scalar multiplication*: the computation of $[k]G + [l]P$, which is usually done either using two separate scalar multiplications or simultaneously (e.g., using the so-called Shamir's trick [Str64]). Precomputation on G can still be used.

The following paragraphs provide a summary of the most common types of techniques for scalar multiplication. Note that this categorization is intentionally vague, as algorithms might combine ideas from several types; for more, see [ACD+05; MVV18; HVM04].

- **Basic** scalar multipliers scan the scalar bit by bit in either left-to-right (LTR) or right-to-left (RTL) fashion, and are often called double-and-add scalar multipliers, akin to their square-and-multiply counterpart in multiplicative notation [HVM04].
- **Windowed** scalar multipliers split the scalar into windows (tuples of digits). These windows can be either fixed, meaning that they regularly divide the scalar, or sliding, in which case they skip over zero bits (there is really only one window sliding across the scalar). Windowed algorithms are parameterized by their window size $w \in \mathbb{N}$. The algorithms also perform precomputation, usually to compute all odd multiples of the point smaller than $2^w - 1$. However, this precomputation is cheap, so it can also be applied in a variable-base scalar multiplier [HVM04].
- **Comb** scalar multipliers perform precomputations and thus are intended for fixed-base scenarios. They are parameterized by a width $w \in \mathbb{N}$, as they iterate over the scalar in a comb-like fashion; each iteration processes all bits that are w bits apart. Combs are generally the fastest fixed-base scalar multipliers [Pip76; BHL+01].
- **Ladder** multipliers are very similar to basic scalar multipliers in that they scan the scalar bit by bit and do not perform any precomputation. However, in each iteration, they use a ladder formula exactly once, regardless of the inputs [Mon87].

Additionally, a few other modifications to the mentioned techniques are often considered to improve the speed or security of the algorithms, for example [Jan20]:

- Scalar multipliers that iteratively process a secret scalar might leak its bit length. To avoid this, the scalar multiplier can have a fixed number of iterations.
- Some formulas for group operations have exceptional inputs, such as the neutral point. The scalar multiplier can protect these formulas by short-circuiting (i.e., using the identities $P + \mathcal{O} = P$, $[2]\mathcal{O} = \mathcal{O}$, $P + P = [2]P$).
- Dummy group operations can be used to ensure that the sequence of operations is independent of the private scalar.
- Addition formulas are not necessarily symmetric in their input, i.e., the computations of $P + Q$ and $Q + P$ might differ. This is reflected in the sequence of the underlying arithmetic operations.

2.2 Side-channel attacks

This subsection presents a brief overview of selected side-channel attacks on ECC that we use in our work, namely *special-point-based attacks*, as coined by Sedlacek et al. [SCJ+21]. This term unifies three attacks: the *Refined Power Analysis* (RPA) [Gou03], the *Zero-Value Point* (ZVP) [AT03], and the *Exceptional Procedure Attack* (EPA) [IT03]. These attacks are similar. They target static ECDH and recover secret bits by adaptively sending public points to the target, thus producing an intermediate zero value conditionally on some secret key bits. The use of zeros in special points ensures that these attacks can bypass some (though not all) countermeasures based on randomization, such as coordinate randomizations [Cor99], because zero is invariant under a change of point representation.

For a complete overview of side-channel attacks on ECC, along with details about attacked implementations, see one of the surveys [FGM+10; FV12; DGH+13; ACL21] or a recent SoK on SCA-protected ECC implementation [BCH+23].

Refined Power Analysis (RPA) [Gou03] is the first special-point-based attack that we consider, with the special points being ones with a zero coordinate (i.e., $P_0 = (x, 0)$ or $(0, y)$). The attack assumes that point addition or doubling of one of the zero-coordinate points during scalar multiplication is detectable via SCA. This is fair since both point addition and doubling consist of many finite field operations using the point coordinates, each of which is implemented using many word-size instructions, which might leak if the zero operand is used. Given the detectability assumption and a zero coordinate point P_0, the attack is simple: construct a point $P = [d^{-1}]P_0$, where d is a hypothesis on some part of the secret key. If the special point is detected during execution, the d-th multiple of the input point was computed during the execution, confirming the hypothesis, as $[d]P = [d][d^{-1}]P_0 = P_0$. Iterating this leads to full secret key recovery. Several randomization countermeasures are ineffective against RPA as they preserve zero coordinates, namely projective coordinate randomization, randomization via curve isomorphism, or randomization via field isomorphism.

Zero-Value Point (ZVP) [AT03] attack can be seen as an extension of the RPA attack. The zero value can now appear in the intermediate computation steps during point doubling or addition rather than just in the coordinates. Zhang et al. [ZLL12] extended the attack to genus 2 curves, while Crépeau and Kazmi [CK12] did so for binary field extension curves. Edwards curves were targeted by Martínez et al. [MST+13], who analyzed the ZVP attack and showed that some addition formulas on Edwards curves are resistant to ZVP.

Compared to RPA, the construction of the input point P is more complex for ZVP. The idea is to express the targeted intermediate value, which we want to zero out, as a polynomial in the coordinates of the input points. For instance, if we denote the input points to **add** as P and Q, then an example polynomial might be $x_P y_Q + x_Q$ where $Q = [k]P$ for some scalar k. As explained in [SCJ+21], this is an instance of the dependent coordinates problem (DCP) that can be reduced to the problem of root-finding for univariate polynomials. The multiplication map $[k]P$ is expressed symbolically and substituted into x_Q in the intermediate polynomial $x_P y_Q + x_Q$, resulting in a polynomial only in x_P. The problem is qualitatively different for formulas with more than one input point (e.g., **add**) compared to formulas with one input point (e.g., **dbl**). In the former case, the degree of the multiplication-by-k map grows quadratically with k, putting a limit on the scalars that can be used. In the latter case, the intermediate values depend only on P, thus giving polynomials that are already univariate (since the coordinate y_P can be removed using the curve equation).

Exceptional Procedure Attack (EPA) [IT03] is the final special-point-based attack. Here, the attacker exploits an issue in the point addition formulas in which adding a specific pair of exceptional points degenerates and produces an invalid output. This invalid output point then propagates through scalar multiplication and results in a wrongly computed ECDH shared secret or an error raised by the target, both of which can be detected by the attacker. Similarly to previous special-point-based attacks, the attacker adaptively constructs the public point such that a pair of exceptional points is encountered conditionally on a part of the secret key.

2.3 Reverse engineering

In this subsection, we discuss two categories of side-channel-based reverse engineering (RE): works that build a side-channel-based instruction-level disassembler and works that focus on reverse engineering properties of cryptographic implementations through side-channel analysis (SCA) without necessarily recovering separate instructions.

Side-channel-based disassembly

Quisquater and Samyde [QS02] were the first to apply SCA to reverse engineering by recovering the sequence of instructions executed on a smartcard using electromagnetic (EM) radiation, a correlation-based classifier, and a neural network. Targeting JavaCard, a popular programmable smartcard platform supporting a subset of Java, Vermoen et al. [VWG07] reported a successful reverse engineering of bytecode using power analysis based on simple template classification. Moreover, they used certain properties of bytecode, like the impossibility of some bytecode sequences, to improve their RE technique further.

Several side-channel-based disassemblers were developed for various simple microprocessors: a disassembler utilizing a Bayesian classifier targeting the PIC microcontroller [EPW10], a disassembler based on k-Nearest Neighbor (kNN) classifier targeting the ATMega163 microcontroller [MMM14], and a disassembler based on hierarchical classification setup and principal component analysis targeting ARM Cortex-M3 [VMA20].

A more complex architecture, system-on-chip based on ARM Cortex-A9, was targeted by Maillard et al. [MHL+22]. Due to its significant complexity compared to previous work, they were faced with considerable challenges. They demonstrated perfect accuracy in functional unit recognition (e.g., recognizing ALU activity from memory loads) and displayed signs of leakage on the bit-level of instructions.

Strobel et al. [SBO+15] and Iyer et al. [ITO+24] took a different approach. Both works combine leakage coming from numerous probe locations to achieve high accuracy. The recovery of the instructions is done using dimensionality reduction and the kNN classifier in the former and using hierarchical classification in the latter case.

In contrast to the above disassemblers, which focused on classifying instruction opcodes, other works concentrated on also recovering operands: a disassembler that classifies the bit encoding of instructions of the PIC16F microcontroller [CLH19] and a disassembler targeting ATMega328P via hierarchical classification [PXJ+18]. A disassembler by Gao et al. [GOP22] recovered not opcodes but a micro-architectural leakage model of the target.

Machine learning (ML) was also used for side-channel-based RE of simple AVR microcontrollers: ATMega8A [NAH21] and ATMega328P [ASR+22], though the achieved instruction recognition rates were comparable with previous works. A slightly more complex target, ARM Cortex-M0, was also a target of ML RE [vGB22; BH22].

Side-channel-based reverse-engineering

SCA has also been used to reverse-engineer cryptographic primitive implementations. This method is sometimes called SCARE, standing for 'Side-Channel Analysis for Reverse-Engineering'. Performed for the first time by Clavier [Cla04], who recovered the substitution tables of the proprietary GSM A3/A8 ciphers, which were secret at the time. Daudigny et al. [DLM+05] analyzed an implementation of the Data Encryption Standard on a smartcard and were able to reverse-engineer constants used in the algorithm as well as some implementation details, such as which registers were used to store key material.

Several more works focused on RE of various implementations of symmetric-key schemes: hardware Feistel implementation [RDG+08], LFSRs and generic non-linear functions [GSM+10], and S-boxes in typical structures such as Substitution-Permutation Network (SPN), Feistel, and eXtended Feistel [TQP+14]. In contrast to previous techniques, [RR13] only assumed that an SPN is used, and demonstrated recovery of the full design.

Active side-channel attacks, such as fault injection, have also been used for RE. San Pedro et al. [SSG11] presented a technique called FIRE (Fault Injection for Reverse Engineering) targeting an unknown S-box in a known cipher. Fault injection, along with passive SCA, was used by Clavier et al. [CIM+15] to recover information about an AES-like cipher, with some SCA countermeasures implemented and without knowledge of the key.

Amiel et al. [AFV07] showed how to apply Correlation Power Analysis to recover the word size and modular multiplication algorithm used in an RSA implementation. They also hinted at possibly applying their RE against ECC as future work. Roche et al. [RLM+21] performed SCA against a black-box smartcard ECDSA implementation. To do that, they first manually reverse-engineered the scalar multiplier used before developing an attack.

3 Analysis of open-source libraries

To better understand the implementation choices made by real-world implementations of ECC and to answer **RQ1** we analyzed the sources of 18 open-source cryptographic libraries in their most recent released version (as of January 2024): BearSSL, BoringSSL, Botan, BouncyCastle, fastecdsa, Go crypto, Intel IPP cryptography, libgcrypt, LibreSSL, libsecp256k1, libtomcrypt, mbedTLS, micro-ecc, Nettle, NSS, OpenSSL, SunEC and Microsoft SymCrypt. Our analysis was restricted to code implementing ECDH, ECDSA on prime-field curves and X25519, Ed25519. By understanding the diversity observed in open-source libraries, we gained insights into the expected diversity in black-box implementations such as cryptographic smartcards or closed-source embedded systems.

For each library and cryptosystem operation (e.g., ECDH key generation or ECDSA signing), we documented the curve model, scalar multiplier, coordinate system, and addition formulas used. In case a library contained multiple implementations, for example, architecture or curve-specific ones, we documented them all. Due to the large amount of data collected, we only present a summary of it here[2]. We also contacted maintainers of the mentioned libraries and 8 of them confirmed or corrected our analysis of their library.

3.1 Specific implementations

Out of 18 libraries, more than half (10) have curve-specific or architecture-specific implementations in addition to a generic one that is used as a fallback, resulting in 64 total configurations. These implementations usually include optimized formulas (e.g., for Short-Weierstrass curves with $a = -3$) and scalar multipliers (e.g., precomputed multiples of the generator). As a result, it is not enough to consider only a specific library but also a specific hardware platform and curve on which the library is executed.

3.2 Curve models

It may seem that the curve model used in an implementation is determined by the cryptosystem implemented, which is either Short-Weierstrass, Montgomery, or Twisted Edwards. However, some curves can be transformed into birationally equivalent curves in a different curve model. This is supported by our analysis, which shows that 4 out of 13 implementations of X25519 use a Twisted Edwards model internally instead of the Montgomery model. All of the ECDH, ECDSA, and Ed25519 implementations were in the curve model that the cryptosystem specifies. Another practical example of a curve model transforming implementation is `ECCKiila` [BBC+20], a tool for generating ECC implementations that picks a Twisted Edwards model whenever able, even for Short-Weierstrass curves.

[2]Full report is available on https://pyecsca.org/libraries.html.

3.3 Scalar multipliers

The implementations often contained three types of scalar multipliers: a *fixed base* (for the generator), a *variable-base* (for any single point), and a *multi-scalar* one (for a pair of points). We do not present the results for the multi-scalar multiplier here due to their complexity and size, but they can be found in the full report[2].

In the Short-Weierstrass model, the fixed-base multipliers included: comb methods [HVM04, Alg. 3.44], fixed-window method [HVM04, Alg. 3.41], simple ladder method [Mon87] and double-and-add-always [Cor99]. In some cases, the fixed-window method was done with full pre-computation, meaning that all options of all windows were statically stored, and the scalar multiplier contained only addition. Similarly, scalar recoding techniques such as Booth [Boo51; Mac61] recoding were used. The width parameter of both comb and fixed-window methods varied from 4 to 7 bits. For variable-base multiplication, there were no combs, while the fixed-window methods remained, with the addition of some GLV [GLV01] and (regular) wNAF [HVM04] methods.

In the Montgomery model, the fixed base multiplier was either the Montgomery ladder [Mon87] or was done using the Twisted Edwards model and a different multiplier. The variable base multiplier was always (a variant of) the Montgomery ladder.

In the Twisted Edwards model, both the fixed- and variable-base multipliers varied between Pippenger's method [Pip76], comb methods, fixed-window methods, and double-and-add-always.

3.4 Coordinate systems

In the Short-Weierstrass model, Jacobian coordinates dominated as they were present in all but 5 of the 18 libraries. The remaining implementations chose predominantly homogeneous projective coordinates but also xz or Jacobian-modified coordinates.

The Montgomery model implementations all used xz coordinates, which is an obvious choice. Note, however, that some X25519 implementations used the Twisted Edwards model internally and thus are described in the next paragraph.

The case of the Twisted Edwards model was the most complex, with several implementations using and mixing several coordinate systems: projective, extended, completed, and 'Duif' or 'Niels'[3]. This complexity is likely due to the large influence of a few initial implementations of Ed25519, namely the SUPERCOP *ref10* implementation of Ed25519, which uses these coordinate systems. Other, simpler implementations in the Twisted Edwards model used projective coordinates. For an overview of Ed25519 implementations, see [GS21]. For the rest of this work, we will focus on single-coordinate systems, leaving aside these complicated implementations.

3.5 Formulas

We counted a total of 113 formula implementations in the analyzed libraries. Out of those, we could directly map 50 to formulas from the EFD database, meaning that the implementation either pointed at the EFD entry or that manual analysis of the formula led us to conclude that it is equal to an EFD formula. There were 21 distinct EFD formulas used. For 40 implementation formulas, we were not able to perform this mapping manually, and we analyze them below. The remaining $113 - 50 - 40 = 23$ formulas are out-of-scope of our analysis as they mix coordinate systems.

[3]Named after Niels Duif, one of the authors of the EdDSA signature scheme.

The unmatched 40 formulas showed similarities to EFD formulas to a certain extent, which we measured in the following way: As in Sedlacek et al. [SCJ+21], we unrolled each formula, expressing its intermediate values as polynomials in its inputs. The set of these polynomials then served as a representation for each formula.

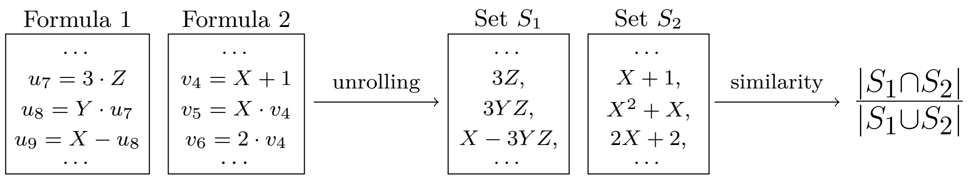


Figure 1: Measuring similarity of formulas by expressing intermediate values as polynomials and computing the ratio of the size of the intersection and the union.

We defined the similarity of formulas using their corresponding sets of polynomials. More precisely, the similarity between two sets S_1, S_2 was given as $0 \leq |S_1 \cap S_2|/|(S_1 \cup S_2)| \leq 1$ (see Figure 1). For each of the 40 formulas, we found an EFD formula with the highest similarity. For 9 formulas, an EFD match with similarity 1 was found, and for the rest, the largest similarity varied between 0.25 and 0.96.

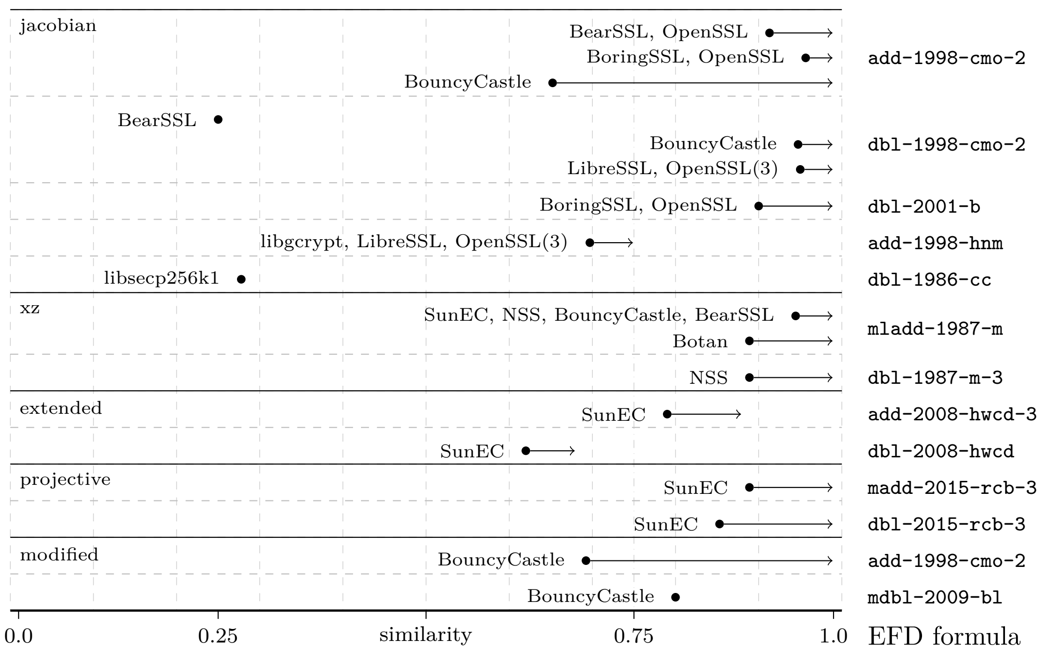


Figure 2: Formula similarity between unknown non-EFD formulas found in libraries (dots) and their closest EFD formulas (y-axis). The arrows represent the increase of similarity after our expansion of the EFD.

To bridge the gap between the unmatched library implementations of formulas and the EFD database, we investigated the differences between the formulas and implemented the following transformations:

- 'Fliperoo'[4] of operands: There are several ways to compute $x \cdot y \cdot z$ due to the associativity and commutativity of multiplication and similarly for addition/subtraction.

[4]We named this after the climbing move from [WR20].

- Sign switch: The sign of intermediate values containing subtraction can be switched by changing the order of operands. To preserve correctness, multiple signs need to be switched.
- Expansion and reduction of multiplications: Since $3 \cdot a = 2 \cdot a + a = a + a + a$, we can expand multiplications by small values to additions and vice versa. Some implementations act differently depending on whether the value is a power of 2 (and can be computed efficiently).

Applying these transformations, we were able to expand the EFD database to contain almost 20000 formulas[5]. This significantly improved the similarity matching as shown by Figure 2, in which the dots represent the unknown library formulas with their similarity match to the EFD on the x-axis. The arrows represent the change of similarity after our expansion of the EFD. For instance, although the BouncyCastle formulas were not that similar to any of the EFD formulas, after our expansion of EFD, we found a match for every one of them. In total, out of the $90 = 113 - 23$ library formulas in our scope, 80 were successfully matched to a transformed variant of an EFD formula, with only 3 formulas having the closest match below 0.75.

We investigated transforming the formulas to a canonical form instead of expanding them. However, we found no canonical form covering the transformations seen in the library formulas yet distinguishing the EFD formulas.

While the EFD database is quite extensive (nearly 200 formulas in the specified models), one should clearly not expect to find all real-world ECC implementations there. Our extended EFD database (almost 20000 formulas) shows that the space of possible implementations is much larger.

RQ1

The analyzed open-source libraries clearly leverage a wide range of possible scalar multipliers, coordinate systems, curve models, and formulas. What's more, they modify them, mix them, and transform them for more optimizations. This shows the diversity and the vast amount of real-world ECC implementations.

4 Reverse-engineering techniques

This section presents three reverse-engineering methods that utilize existing side-channel attacks – RPA, ZVP, and EPA. We use these side-channel attacks to build oracles, which we query about what is happening inside the target implementation during some computation. This allows us to distinguish different implementations based on their behavior under these oracles. Our methods focus on reverse engineering the coordinate system, formulas, and the scalar multiplier.

RPA, ZVP, and EPA attacks construct input points that invoke 'special' behavior during the computation on the device, detectable using a side-channel. This can be modeled as a boolean oracle that returns `True` if this special behavior is detected and `False` otherwise, possibly with some noise. The attacks use this oracle and the knowledge of the implementation configuration to verify hypotheses about the private scalar. We turn this around and make hypotheses about the configurations, and verify them using the oracle. Table 1 shows an overview of the methods and their capabilities. For example, the RPA method queries a side-channel oracle about an unknown multiplier while choosing the used curve, the scalar, and the input point.

In more detail, for each configuration from a set $\mathcal{C}$ of possible configurations, we simulate the oracle on a set of inputs $\mathcal{I}$ and record the oracle outputs in a decision table

[5]This dataset is available at https://doi.org/10.5281/zenodo.10908698.

Method	**Curve**	**Coordinates**	**Formulas**	**Multiplier**	**Scalar**	**Input point**
RPA-RE	chosen	any	any	**target**	known	chosen
ZVP-RE	chosen	**target**	**target**	known	known	chosen
EPA-RE	chosen	**target**	**target**	known	known	chosen

Table 1: Overview of our methods with parts of configurations they target for reverse-engineering. Some parts need to be either chosen or known.

as in Figure 3. The rows of the table correspond to configurations from $\mathcal{C}$ and columns correspond to the inputs from $\mathcal{I}$ with the oracle outputs in the table entries. Based on the table, we construct a decision tree for each attack. This tree is then used for an efficient search through the space of configurations $\mathcal{C}$ to minimize the number of oracle calls (and thus side-channel measurements).

The decision tree is constructed from the table in the following way. From the set $\mathcal{I}$ of inputs in the columns, we select the input for which the oracle result divides the set $\mathcal{C}$ into the smallest possible subsets. In the example in Figure 3 with the binary oracle, we select the input point which divides the four multipliers into two groups of two (which is $[3^{-1}]P_0$). We repeat this process recursively for each subset of multipliers until we reach leaves containing either single configurations or configurations that cannot be distinguished using the method. This process is analogous to translating a decision table into a decision tree. We use heuristics-based methods [Pol65] instead of optimal ones [Lew78] as the general problem is NP-complete [HR76]. In subsections 4.1, 4.2, and 4.3, we describe the details of the individual attack methods, including the construction of the set of inputs $\mathcal{I}$.

$\mathcal{I}_{\text{RPA}}$:	$[2^{-1}]P_0$	$[3^{-1}]P_0$	$[4^{-1}]P_0$	$[5^{-1}]P_0$
LTR	`True`	`True`	`False`	`False`
RTL	`True`	`False`	`True`	`True`
Comb	`True`	`False`	`True`	`False`
Ladder	`True`	`True`	`True`	`False`

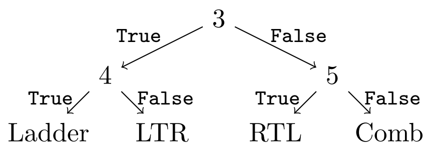


Figure 3: Small illustrative example simulation of the RPA oracle on different multipliers (table rows) and input points (table columns), where $P_0 = (0, y)$. The decision tree is constructed based on the decision table. The example scalar is 213.

Noise. Due to the possible noise in the side-channel measurements, the constructed binary oracles sometimes do not answer correctly. We address this by querying the oracle multiple times and doing a majority vote. For instance, each oracle answer (`True`/`False`) in the decision tree in Figure 3 is the winning majority result of several RPA oracle calls.

Target implementation. The described methods require control over the input point (Table 1) and can thus target ECDH Derive and ECDSA Sign, where the curve base point serves the role of the input point. The success of the method depends on the used curve and the scalar (e.g., a point with zero coordinate must lie on the curve for RPA), and so we assume control over the curve and the scalar. For ZVP, we assume that any requirements of the implementation on the domain parameters a, b (e.g., $a = -3$) are checked and can be therefore inferred from the (error) output of the implementation, based on our analysis of libraries. As was shown in [SJS20], computationally demanding checks (such as deterministic primality tests) are often ignored, so we exclude these for the EPA method.

Countermeasures. Scalar randomization thwarts the three considered side-channel attacks and, consequently, our RE methods [FGM+10]. However, the methods also inherit resistance against several other countermeasures from the corresponding attacks (see

Section 2.2), namely coordinate and curve randomization. Moreover, countermeasures are generally designed to stop attacks on the private key, and they are not necessarily effective against reverse engineering using our or other (future) methods. After all, the countermeasures were not intended for our strong model, in which we have full control over the domain parameters, the inputs, and the private key.

4.1 RPA

In RPA, the side-channel oracle is the detection of points with zero coordinates during the scalar multiplication $[k]P$, where k is a fixed private scalar and P is a general point. The space of configurations $\mathcal{C}$ is the set of all scalar multipliers, and the goal is determining the one being used. We assume the elliptic curve is fixed and has a point P_0 with a zero coordinate. For each scalar multiplier $C \in \mathcal{C}$, we compute the set $\{[k_{C,1}]P, \ldots, [k_{C,r}]P\}$ of multiples of a general input point P that appear during the computation of $[k]P$. The set of inputs is then

$$\mathcal{I}_{\mathrm{RPA}} = \bigcup_{C \in \mathcal{C}} \{[k_{C,1}^{-1}]P_0, \ldots, [k_{C,r}^{-1}]P_0\}.$$

Each of these input points $[k_{C,i}^{-1}]P_0$ causes the appearance of the zero coordinate point $[k_{C,i}][k_{C,i}^{-1}]P_0 = P_0$, if the scalar multiplier C is used.

4.2 ZVP

In ZVP, the side-channel oracle detects zero intermediate values in the computation of the formulas during the scalar multiplication. We will assume that the scalar multiplier is known (recovered using, e.g., the RPA technique), so the goal is to reverse-engineer the formulas and coordinate system used. The set of configurations $\mathcal{C}$ will be the combinations of formulas for all operations used in the scalar multiplier (e.g., (`add-2007-bl`, `dbl-2015-rcb`) $\in \mathcal{C}$ for a multiplier using **add** and **dbl**).

The general idea of the construction of inputs $\mathcal{I}_{\mathrm{ZVP}}$ is illustrated in Figure 4. Each layer corresponds to a choice of curve E_l and the used private scalar k_l. Columns describe the sequence of curve operations as defined by the scalar multiplier and the scalar k_l (e.g., **dbl**, **add**, **dbl**, **dbl**,... for the first layer). Rows correspond to all of the possibilities for the **add** and **dbl** formulas. For each row, we unroll the formula F for the given operation to create a set of intermediate polynomials, as described in Section 3.5. We solve the dependent coordinates problem for all the intermediate polynomials (see Section 2.2), which results in sets $\mathcal{P}_i(E_l, j)$ of pairs of points and curves. Since the complexity of the dependent coordinates problem grows with the size of the scalar used, we compute $\mathcal{P}_i(E_l, j)$ only for small j (but some instances of DCP can be easily solved without such restriction). By definition, each point from $\mathcal{P}_i(E_l, j)$ causes a zero value during the computation of the corresponding column operation (i.e., **add**(P, $[j]P$) or **dbl**($[j]P$)) as long as the i-th row is the correct choice of formulas. The set $\mathcal{I}_{\mathrm{ZVP}}$ of inputs is then the union of all triples of points, curves, and private scalars k_l.

As opposed to RPA, we consider more than one oracle for ZVP. The simplest is a binary oracle that indicates whether a zero intermediate value appeared anywhere during the scalar multiplication. The power consumption can also leak the number of zero intermediate values that appeared (count oracle) as well as their position in the power traces (position oracle). Furthermore, we consider a noisy count oracle, which we query multiple times, compute the average number of zeroes, and take the closest possible output.

4.3 EPA

The EPA attack leverages exceptional cases of addition formulas to induce an error during scalar multiplication. An oracle indicating whether an error happened is then used to find

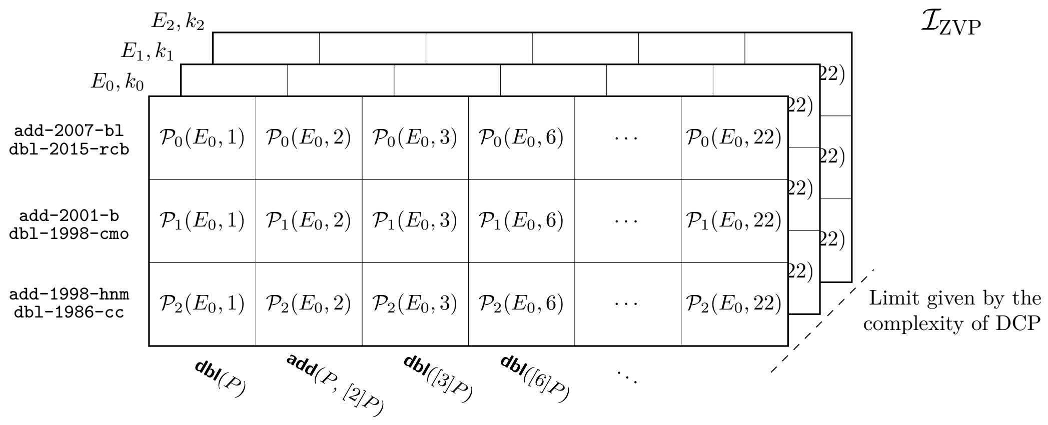

$\mathcal{I}_{\text{ZVP}}$
E_2, k_2
E_1, k_1
E_0, k_0
add-2007-bl dbl-2015-rcb
add-2001-b dbl-1998-cmo
add-1998-hnm dbl-1986-cc
$\mathcal{P}_0(E_0,1)$ $\mathcal{P}_0(E_0,2)$ $\mathcal{P}_0(E_0,3)$ $\mathcal{P}_0(E_0,6)$ $\cdots$ $\mathcal{P}_0(E_0,22)$
$\mathcal{P}_1(E_0,1)$ $\mathcal{P}_1(E_0,2)$ $\mathcal{P}_1(E_0,3)$ $\mathcal{P}_1(E_0,6)$ $\cdots$ $\mathcal{P}_1(E_0,22)$
$\mathcal{P}_2(E_0,1)$ $\mathcal{P}_2(E_0,2)$ $\mathcal{P}_2(E_0,3)$ $\mathcal{P}_2(E_0,6)$ $\cdots$ $\mathcal{P}_2(E_0,22)$
Limit given by the complexity of DCP
dbl(P)
add(P, [2]P)
dbl([3]P)
dbl([6]P)

Figure 4: The space of possible ZVP points for combinations of formulas for **add**/**dbl** and for different curves. Each set $\mathcal{P}_i(E_l, j)$ of ZVP points is computed from the intermediate values of the given formula (row) for the corresponding operation (column).

the secret scalar. We can use this oracle to reverse-engineer the addition formulas.

The limitation of this method is the number of formulas with non-trivial exceptional cases, with only two such formulas in the Short-Weierstrass model [SCJ+21]. This means that the method will not be able to distinguish most formulas. There might be a potential to increase the number of special cases by combining the EPA attack with the results of Sedlacek et al. [SJS20]. They found that several commercial JavaCards are able to work with curves in which the parameter p, specifying the finite field $\mathbb{F}_p$ over which the curve is defined, is not prime. In this case, the implementations compute over $\mathbb{Z}_n$ where elements not co-prime to n do not have multiplicative inverses. It is likely that if an implementation attempts to compute an inverse of a non-invertible element, it will raise an error. We leave the application of EPA for reverse engineering as future work.

5 Toolkit

In this section, we present the features of the **pyecsca** toolkit for side-channel analysis and reverse engineering of ECC implementations. The toolkit consists of three parts: a *core* repository providing the main functionality, a *codegen* package providing the ability to generate C implementations of ECC, and a *notebook* repository with Jupyter notebooks that provide an interface to the toolkit[6]. Our reverse-engineering methods are included in the toolkit, along with documentation and tutorials.

5.1 Configuration

An ECC implementation configuration is a central concept in the toolkit as it represents the implementation details that the toolkit aims to reverse-engineer. We use data from the Explicit-Formulas Database, to build parts of an ECC implementation configuration. Although the EFD is a website, it also contains data in raw text files in an undocumented but stable format, which we parse. We use data from the EFD to build the first three components of an implementation configuration: the curve model, the coordinate system, and the formulas. The remaining seven components are implemented manually.

The implementation configuration, as defined by the toolkit, has ten components, six of which are described below:

[6]We link to these notebooks where relevant using the icon.

- **Curve model**. One of `Short-Weierstrass`, `Montgomery`, `Edwards`, `Twisted Edwards`. Note that some of these models have restrictions on classes of curves they represent.
- **Coordinate system**. The implementation choice of coordinates on the curve model. Different curve models have different coordinate systems.
- **Scalar multiplier** and **Formulas**. The scalar multiplication algorithm, along with the formulas used in it. Currently, there are fourteen multipliers implemented: `LTR`, `RTL`, `Coron`, `Ladder`, `SimpleLadder`, `DiffLadder`, `BinaryNAF`, `WindowNAF`, `Window`, `WindowBooth`, `SlidingWindow`, `FullPrecomp`, `BGMW`, and `Comb`. The different scalar multipliers require different types of formulas. Some multipliers are parameterized, with parameters like `direction` (left-to-right or right-to-left) or `width`.
- **Hash algorithm**. Used to compute the shared secret in ECDH and to hash the messages in ECDSA. One of `None` (identity function on the data), `SHA1`, `SHA224`, `SHA256`, `SHA384`, or `SHA512`.
- **Random sampling**. The technique used to generate uniformly random numbers modulo n. Can be `Sample` or `Reduce`. In the `Sample` case, numbers up to $\lceil \log_2(n) \rceil$ bits are sampled uniformly until one of them is less than n. In the `Reduce` case, a number up to $\lceil \log_2(n) \rceil + 40$ bits is sampled uniformly and reduced modulo n.

These configurations, namely the scalar multipliers, contain all of the scalar multipliers found in our analysis of libraries from Section 3. There is, thus, a high chance that other real-world implementations fall into our exhaustive set of configurations. The toolkit uses four components for finite-field operations, which are not the object of our reverse engineering.

Table 2: Number of configurations by their components.

Curve	Coords	#	Total
$\mathcal{E}_{\mathrm{SW}}$	jacobian	17 136	113 502
	jacobian-0	22 848	
	jacobian-3	28 560	
	modified	2 856	
	projective	9 520	
	projective-1	10 710	
	projective-3	16 660	
	w12-0	476	
	xyzz	1 428	
	xyzz-3	2 856	
	xz	452	
$\mathcal{E}_{\mathrm{M}}$	xz	132	132
$\mathcal{E}_{\mathrm{E}}$	inverted	2 856	14 431
	projective	11 424	
	yz	99	
	yzsquared	52	
$\mathcal{E}_{\mathrm{TE}}$	extended	2 856	11 424
	extended-1	5 712	
	inverted	1 428	
	projective	1 428	

(a) Configurations per coordinate system.

Scalar multiplier	#
LTR	9 328
RTL	9 328
Coron	1 166
Ladder	407
SimpleLadder	2 332
DiffLadder	328
BinaryNAF	4 664
WindowNAF	18 656
WindowBooth	18 656
Window	9 328
SlidingWindow	18 656
FullPrecomp	18 656
Comb	9 328
BGMW	18 656

(b) Configurations per scalar multiplier.

While our analysis of library formulas (Section 3.5) already hinted at the vast space of possible implementations, we can now look at the number of different configurations that we can enumerate, to answer our **RQ2:** *How large is the space of all possible ECC implementations?* We will restrict scalar multiplier parameters to those seen in our library analysis, as they have theoretically unbounded domains.

RQ2

As Table 2 shows, an enumeration of the space of ECC implementations yields 139 489 configurations. This considerable number comes from combinations of different coordinate systems, formulas, scalar multipliers, and their parameters. We believe that this shows that the space of implementation configurations of ECC is large enough to warrant reverse engineering. Furthermore, including low-level details like the finite-field arithmetic, this number increases to an astonishing 304 411 392 configurations.

5.2 Simulation

Many attacks, as well as our reverse-engineering methods, require the ability to compute intermediate values that appear in the target implementation given some inputs. To allow for this, the toolkit is able to simulate KeyGen, ECDH, or ECDSA down to finite-field intermediate values in the form of an execution tree. This tree can then be analyzed or a suitable leakage model can be applied to obtain simulated power traces.

5.3 Code generation

The **pyecsca** toolkit is capable of fully automatic generation of C implementations of ECC given any implementation configuration, for a set of supported micro-controllers and CPU architectures. The target implementation supports ECDH, ECDSA, and key generation, which are accessible through a simple serial interface. We currently target the `STM32F0` and `STM32F3` chips, which are based on ARM Cortex-M0 and ARM Cortex-M4F, respectively. We also target the host device (assumed `x86_64`) such that implementations can be generated, built, and run on the host device for testing and development purposes.

We use the Jinja2 templating language [Jinja2], to automatically generate implementation C sources from template files and the implementation configuration. Out of the whole implementation, only the high-level primitives like ECDH, ECDSA, key generation, and scalar multipliers had to be implemented (once) by hand, with the formulas auto-generated from the configuration. The libtommath library [LTM] is used to implement finite-field arithmetic due to its rich API and variety of algorithms.

This functionality allows users to validate and test attacks and reverse-engineering methods on hardware with known (generated) implementations.

5.4 Emulation

Executing generated implementations on real hardware and performing side-channel measurements on them may introduce unwanted technical difficulties. Thus, the toolkit also offers a middle step: emulation of generated implementations via a CPU emulator. It uses the Rainbow [Don19] CPU emulator by the Ledger-Donjon team, which is itself based on QEMU. The emulator functionality targets the ARM Cortex-M4 and is able to produce instruction-level traces with various leakage models as well as trace memory accesses or add custom hooks and breakpoints. This provides users with yet another way of validating their attacks, reverse-engineering techniques, or countermeasures.

5.5 Miscellaneous

The toolkit aims to provide a comprehensive suite of functions for trace alignment, filtering, pattern matching, signal-processing, and statistical tests.

Several methods for trace alignment are provided, based on cross-correlation, sum-of-absolute-differences, but also elastic alignment based on FastDTW [vWB11]. The toolkit is compatible with ChipWhisperer [OC14], a popular toolkit for side-channel analysis. Its UFO target boards constitute the targets of the code generation.

Trace acquisition is possible using both ChipWhisperer and PicoScope branded oscilloscopes, with a unified API offered. As the collected traces can be large, visualizing them in a responsive and efficient manner is done using HoloViews [HV] and Datashader [DS]. The toolkit supports the Hierarchical Data Format (HDF5) to allow for the storage of trace sets and for working with larger-than-memory trace sets. Some trace operations, like correlation coefficient computation, are GPU accelerated via CUDA.

6 Results

We evaluate our attack-based reverse-engineering methods on two *evaluation levels*.

Oracle simulation. On this level, we directly simulate an oracle answering our queries, potentially with some noise, by simulating the true target implementation and extracting the oracle answers from its execution. For example, in the RPA-RE case, we simulate the execution and output a (noisy) boolean on whether a zero-coordinate point occurred during the execution.

Method simulation. On this level, we still simulate the true target implementation. However, we then apply a leakage model (and noise) to the intermediate values to obtain simulated leakage traces, on which we then mount the reverse-engineering method. For example, in the RPA-RE case, we essentially mount the RPA attack on the simulated leakage traces to construct the oracle.

The two levels are complementary. By evaluating on the oracle simulation level, we see how the reverse-engineering method behaves in the presence of various levels of noise. By evaluating on the method simulation level, we see how noise in the traces and choices in the attack methodology influence the noise in the resulting oracle. On both levels, we iterate the true configuration that is simulated over the set of all possible configurations.

Noise. We model the noise in binary oracles in two ways: symmetric and asymmetric. In the symmetric case, there is a single error probability e of a flip in the oracle's output. In the asymmetric case, there are two error probabilities (e_0, e_1), corresponding to a flip in the oracle's output $0 \to 1$ and $1 \to 0$. The two models are equal when $e_0 = e_1 = e$. We chose $e \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$, note that $e = 0.5$ makes the oracle output completely random. We add noise to the count oracle by adding a zero-mean-shifted Binomial distribution $B(0.5, n)$ to the count, with $n \in \{0, 4, 10, 20, 40, 60\}$ which we picked heuristically based on the observed distribution of counts. In method simulation, we add zero-mean Gaussian noise with variable standard deviation to normalized simulated leakage traces.

Metrics. We consider four metrics in the evaluation: success rate, precision, result size, and query rate. As the RE methods output a set of configurations (possibly a singleton), we consider the result to be successful if the true configuration is in the output set and precise if the set is a singleton. We also measure the average result set size as it represents precision from a different perspective. We evaluate the oracle query rate because each oracle query amounts to performing a side-channel attack, which increases attacker effort.

The decision trees built by the methods offer insights into their behavior. Leaves represent indistinguishable sets of configurations. Random walks in the tree correspond to the reverse-engineering process in the random scenario with a noisy oracle (i.e., error probability = 0.5). Walks in the tree weighted by the leaf size model the reverse-engineering process with a uniform probability of each configuration and no errors. The expected leaf depth (🍃↕) then corresponds to the expected oracle query rate, and the expected leaf size (|🍃|) corresponds to the expected result size.

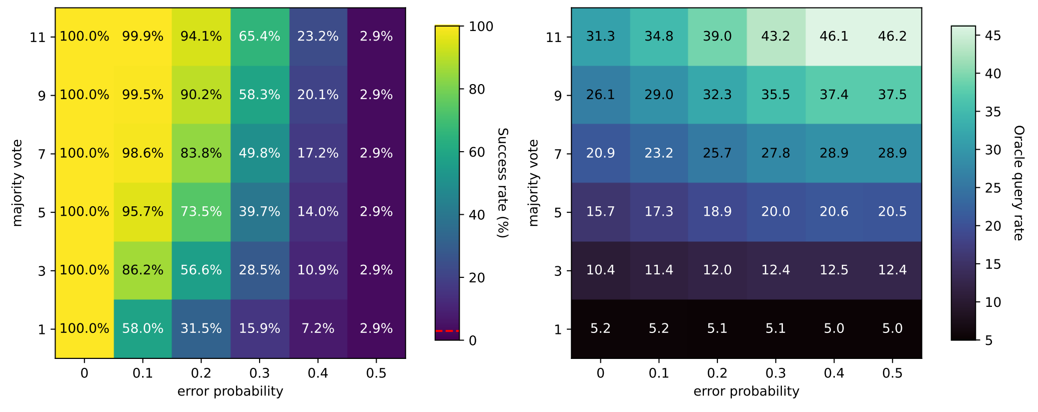
Table 3: Decision tree metrics of our reverse-engineering methods. The **Expected** case considers a uniform probability of each configuration and the **Random** case considers a random walk.

				Expected		Random	
Method	**Oracle**	$\lvert\mathcal{C}\rvert$	#🍃	|🍃|	🍃↕	|🍃|	🍃↕
RPA-RE	binary	34	34	1.0	5.0	1.0	5.0
ZVP-RE	binary	214	74	8.7	5.1	5.0	4.0
ZVP-RE	count	214	134	2.4	4.0	1.3	2.5
ZVP-RE	position	214	196	1.2	2.1	1.1	1.8

6.1 RPA-RE

We evaluated the RPA-RE method considering a set of 34 scalar multipliers to reverse-engineer. We chose the NIST P-256 curve as it is a commonly used curve that also has a zero-coordinate point. In this setting, the method achieves 100% precision as it constructs a decision tree that has single-element leaves (result sets). We used 10 000 runs per a combination of true configuration, error probability and majority vote parameters. In the figures, this amounts to 340 000 runs per heatmap cell.

The success rate and oracle query rate for the oracle simulation level are visible in Figure 5. As it shows, it is possible to significantly improve the success rate of the reverse engineering by increasing the majority vote quorum at the cost of a higher query rate. Note that the baseline success rate of random guesses is 2.94%. The query rate changes with the noise as the majority voting aborts early if the majority is reached, thus lowering the query rate for less noisy scenarios. Our experiment with asymmetric error probabilities behaved symmetrically in all metrics, and we thus omit its visualization.



(a) Success rate (- - random guess). **(b)** Oracle query rate.

Figure 5: Results for the RPA-RE method with symmetric noise.

Method simulation. We evaluated the oracle building step of the RPA-RE method while varying the number of simulated traces per group in $\{10, 20, 30, \ldots, 100\}$ and standard deviation of noise $\sigma \in \{0, 1, 2, \ldots, 10\}$ averaging over 200 runs for each parameter choice.

The trace simulation included the coordinate randomization countermeasure. We implemented the RPA oracle by running simple peak-finding on a difference-of-means trace constructed from normalized traces. As Figure 6 shows, increasing the number of traces allows us to build an oracle with minimal error probabilities even under considerable noise.

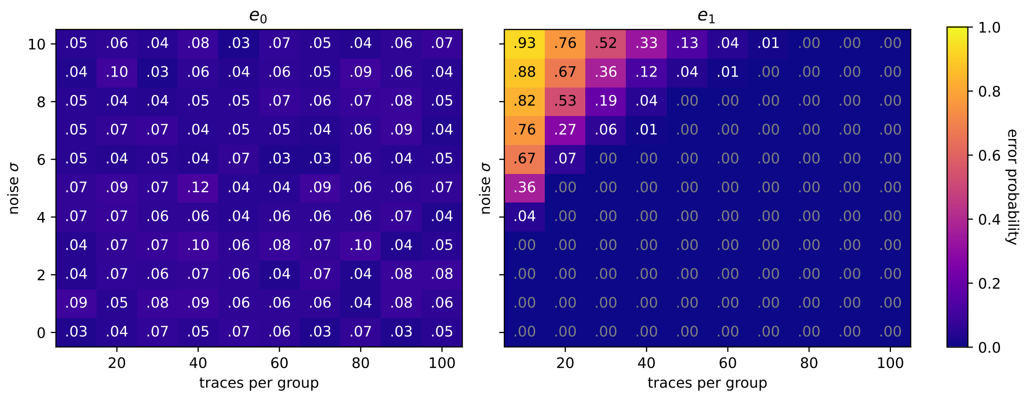


Figure 6: Error probabilities e_0 and e_1 in the RPA-RE method during method simulation. Traces per group refer to the two groups of traces collected in an RPA attack.

6.2 ZVP-RE

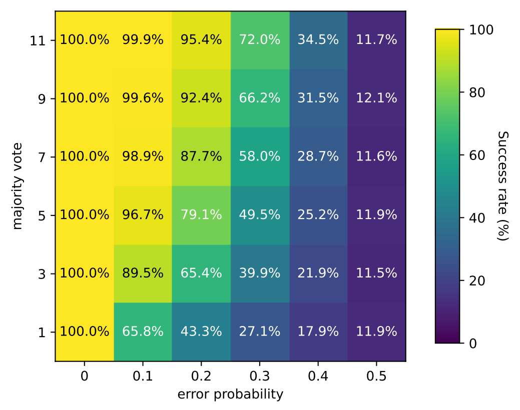
For evaluating the ZVP-RE method we chose a basic left-to-right scalar multiplier on the Short-Weierstrass curve model. Thus, all (**add**, **dbl**) formula pairs from compatible coordinate systems formed the target configurations, which amounted to 214 pairs. We used a total of 40 randomly generated curves, 10 for each of the coordinate system assumptions ($a = -3$, $a = -1$, $a = 0$, and generic), together with 10 small scalars. We bounded the ZVP point computation $\mathcal{P}_i(E_l, j)$ at $j \leq 100$. This amounted to 2212 distinct ZVP points. To build the decision trees in ZVP-RE, we first split the configurations based on the coordinate system assumption as explained in Section 4. Thus, the trees are one level deeper than suggested by the average oracle query rate (we do not count this configuration split as an oracle query).

We used 100 runs per a combination of true configuration, error probability, and majority vote parameters. In the figures, this amounts to 21 400 runs per heatmap cell.

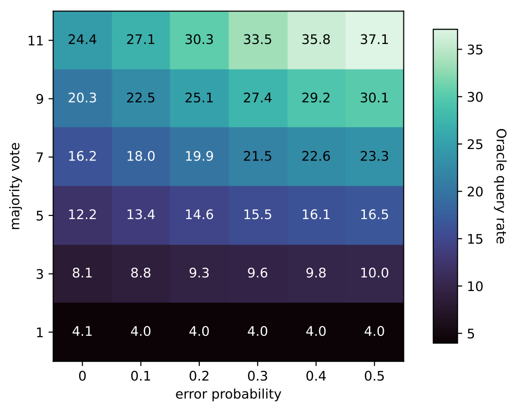
Binary oracle. In this setting, the ZVP-RE method builds a tree with 74 leaves. Its metrics are visible in Table 3. It is no longer always precise like RPA-RE as Figure 7 shows. However, its average result size of 8.7 configurations (formula pairs) demonstrates utility in narrowing the space of possible configurations. This directly corresponds to the expected leaf size, as reported in Table 3.

The leaves contain configurations that are indistinguishable from the perspective of the method. They contain formulas with the same set of ZVP points or, very often, the same intermediate polynomials (these configurations have the same rows in Figure 4). The leaves vary in size, with the largest leaf containing 30 configurations and 26 leaves containing one configuration each. This imbalance in the leaf sizes also explains why the high error oracle with a low success rate has higher precision than the low error oracle (see Figure 7c).

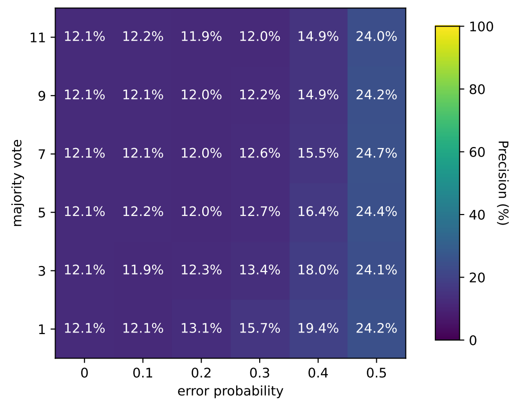
When considering the coordinate system only, of which there are 11, the method achieves a precision of 94%. Our experiment with asymmetric error probabilities did not behave symmetrically due to the decision tree being unbalanced. The success rate was slightly better in the prevalence of $0 \to 1$ errors over $1 \to 0$ errors. We omit this visualization due to space.



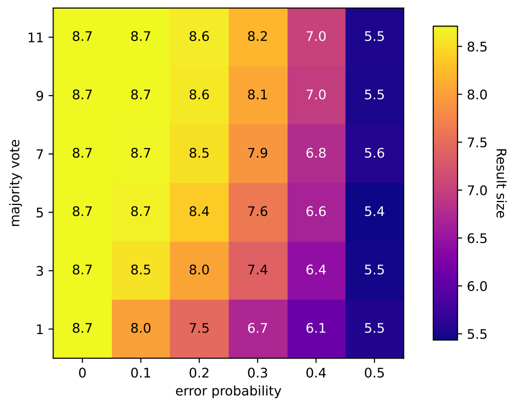
(a) Success rate.



(b) Oracle query rate.



(c) Precision.



(d) Average result size.

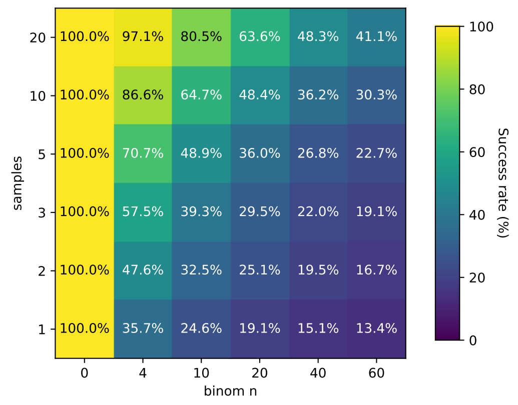
Figure 7: Results for the ZVP-RE method with a binary oracle with symmetric noise.

Count oracle. Using the count oracle (the number of zero intermediate values) improves the precision of the method significantly, to an average result size of 2.4 (precision of 41.6%). Figure 8 shows that even a large amount of noise can be handled by this method. Note that majority voting does not help increase the precision; it only increases the success rate. The tree remains unbalanced like in the case of the binary oracle, which leads to higher precision in higher noise scenarios.

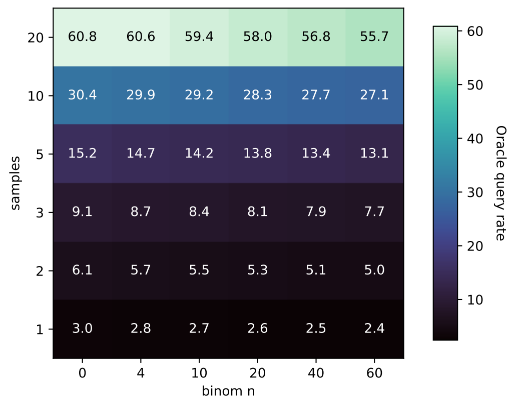
Position oracle. The position oracle outputs the position of zero intermediate values in the power traces. Using this oracle, the expected results size is 1.2 (Table 3), improving the precision of ZVP-RE to 85.6%. Additionally, since the arity of the decision tree increases with the oracle output size, the expected oracle query rate reduces to 1.1 (the first level of the tree is given by the coordinate system assumptions).

RQ3

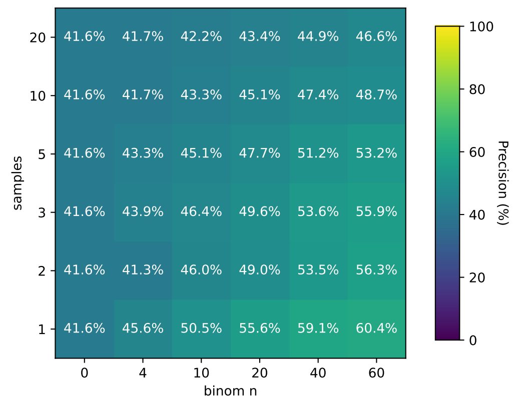
As our results show, the RPA-RE method is always precise in determining the scalar multiplier and is successful under considerable noise. While the ZVP-RE method has limits in its ability to distinguish individual formula pairs, its precision improves when considering the zero count (41.6%) or the zero position oracle (85.6%). Furthermore, when considering the coordinate system only, it achieves a precision of over 94% with the binary oracle.



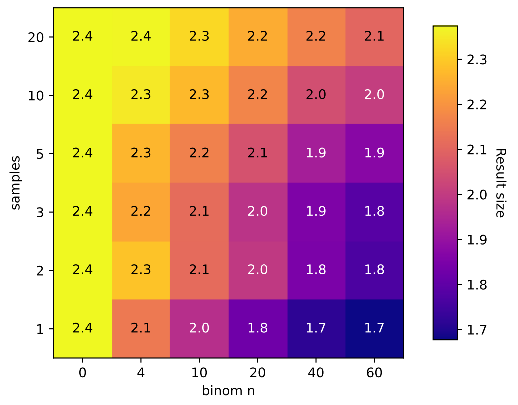
(a) Success rate.



(b) Oracle query rate.



(c) Precision.



(d) Average result size.

Figure 8: Results for the ZVP-RE method with a zero count oracle with binomial noise.

7 Conclusions

Implementing elliptic curve cryptography is a non-trivial process, requiring a range of implementation choices – especially when considering fast yet side-channel and fault-induction resilient implementations. The space of ECC implementation configurations has not been systematically studied so far, we arrived at a conservative estimate of 139 489 possibilities. By analyzing 18 open-source ECC libraries, we indeed documented a surprisingly wide variety of implementation decisions, likely to be also present in black-box implementations in secure hardware or smartcards. This variety is an obstacle to an attacker mounting a side-channel attack, as knowing the implementation details is frequently crucial for a successful attack. However, this knowledge is often treated as a mandatory assumption in the literature describing attacks on ECC.

This work provides a novel approach for systematic analysis and reverse engineering of black-box ECC implementations that is able to obtain the used scalar multiplier, the coordinate system, and the addition formulas. We take known side-channel attacks (RPA, ZVP, and EPA) and their dependence on the implementation configurations and turn them into reverse-engineering methods. We implement these methods as part of **pyecsca** – the first toolkit for automatic reverse engineering of ECC implementations. The toolkit is well-documented and tested, providing extensive functionality, including the enumeration of millions of ECC implementations, their synthesis for embedded devices, as well as the acquisition, processing, and visualization of traces. We hope that our toolkit will be used in future research into side-channel analysis and elliptic curve cryptography.

The results of the experimental evaluation demonstrate that RPA-RE and ZVP-RE can be used to automatically reverse-engineer the scalar multiplier and the coordinate system of a black-box ECC implementation. Furthermore, the ZVP-RE method is able to considerably reduce the space of possible addition formulas. Our reverse-engineering methods are resistant to coordinate and curve randomization. We leave the adaptation of the methods against more countermeasures (e.g., scalar randomization) as future work. Furthermore, the application of our methods to real-world black-box implementations, such as JavaCards or TPMs, is interesting for future research.

Acknowledgements

We would like to thank Jan Kvapil, Vashek Matyas as well as anonymous reviewers for their insightful reviews. Thanks to Tomáš Jusko and Andrej Bátora for their contributions to the toolkit. Jan Jancar was supported by Red Hat Czech. J. Jancar, V. Suchanek, P. Svenda, and Ł. Chmielewski were supported by the AI-SecTools (VJ02010010) project. V. Sedlacek was supported by the NSF grant CNS-2124692.

References

[ACD+05] Roberto M. Avanzi, Henri Cohen, Christophe Doche, Gerhard Frey, Tanja Lange, Kim Nguyen and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005. ISBN: 978-1-4398-4000-9.

[ACL21] Rodrigo Abarzúa, Claudio Valencia Cordero and Julio Cesar López-Hernández. Survey on performance and security problems of countermeasures for passive side-channel attacks on ECC. *Journal of Cryptographic Engineering*, 11(1):71–102, April 2021. DOI: 10.1007/s13389-021-00257-8.

[AFV07] Frederic Amiel, Benoit Feix and Karine Villegas. Power analysis for secret recovering and reverse engineering of public key algorithms. In *SAC 2007*, volume 4876 of *LNCS*, pages 110–125. Springer, Heidelberg, August 2007. DOI: 10.1007/978-3-540-77360-3_8.

[ASR+22] Cesar N. Arguello, Hunter Searle, Sara Rampazzi and Kevin R. B. Butler. A practical methodology for ML-based EM side channel disassemblers. *CoRR*, abs/2206.10746, 2022. DOI: 10.48550/arXiv.2206.10746.

[AT03] Toru Akishita and Tsuyoshi Takagi. Zero-value point attacks on elliptic curve cryptosystem. In *ISC 2003*, volume 2851 of *LNCS*, pages 218–233. Springer, Heidelberg, October 2003. DOI: 10.1007/10958513_17.

[BBC+20] Dmitry Belyavsky, Billy Bob Brumley, Jesús-Javier Chi-Domınguez, Luis Rivera-Zamarripa and Igor Ustinov. Set it and forget it! Turnkey ECC for instant integration. In *ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*, pages 760–771. ACM, 2020. DOI: 10.1145/3427228.3427291.

[BCH+23] Lejla Batina, Lukasz Chmielewski, Björn Haase, Niels Samwel and Peter Schwabe. SoK: SCA-secure ECC in software - mission impossible? *IACR TCHES*, 2023(1):557–589, 2023. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i1.557-589.

[BH22] Daehyeon Bae and JaeCheol Ha. Implementation of disassembler on microcontroller using side-channel power consumption leakage. *Sensors*, 22(15):5900, 2022. DOI: 10.3390/S22155900.

[BHL$^+$01] Michael Brown, Darrel Hankerson, Julio López and Alfred Menezes. Software implementation of the NIST elliptic curves over prime fields. In *Topics in Cryptology—CT-RSA 2001: The Cryptographers' Track at RSA Conference 2001 San Francisco, CA, USA, April 8–12, 2001 Proceedings*, pages 250–265. Springer, 2001. DOI: 10.1007/3-540-45353-9_19.

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

[BL17] Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. 2017. URL: https://hyperelliptic.org/EFD/ (visited on 10/07/2024).

[Boo51] Andrew D. Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, January 1951. ISSN: 0033-5614. DOI: 10.1093/qjmam/4.2.236.

[CC] Common Criteria. ISO/IEC 15408 Information technology — Security techniques — Evaluation criteria for IT security. In *ISO/IEC 15408-1:2022*. ISO/IEC, 2022.

[CC86] David V. Chudnovsky and Gregory V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. 7(4):385–434, December 1986. ISSN: 0196-8858. DOI: 10.1016/0196-8858(86)90023-0.

[CIM$^+$15] Christophe Clavier, Quentin Isorez, Damien Marion and Antoine Wurcker. Complete reverse-engineering of AES-like block ciphers by SCARE and FIRE attacks. *Cryptogr. Commun.*, 7(1):121–162, 2015. DOI: 10.1007/s12095-014-0112-7.

[CK12] Claude Crépeau and Raza Ali Kazmi. An analysis of ZVP-Attack on ECC cryptosystems. Cryptology ePrint Archive, Report 2012/329, 2012. https://eprint.iacr.org/2012/329.

[Cla04] Christophe Clavier. Side channel analysis for reverse engineering (SCARE) – an improved attack against a secret A3/A8 GSM algorithm. Cryptology ePrint Archive, Report 2004/049, 2004. https://eprint.iacr.org/2004/049.

[CLH19] Valence Cristiani, Maxime Lecomte and Thomas Hiscock. A bit-level approach to side channel based disassembling. In *CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *LNCS*, pages 143–158. Springer, 2019. DOI: 10.1007/978-3-030-42068-0_9.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems*, pages 292–302, Berlin, Heidelberg. Springer, 1999. DOI: 10.1007/3-540-48059-5_25.

[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):241–265, November 2013. DOI: 10.1007/s13389-013-0062-6.

[DLM$^+$05] Rémy Daudigny, Hervé Ledig, Frédéric Muller and Frédéric Valette. SCARE of the DES. In *ACNS 05*, volume 3531 of *LNCS*, pages 393–406. Springer, Heidelberg, June 2005. DOI: 10.1007/11496137_27.

[Don19] Ledger Donjon, 2019. URL: https://github.com/Ledger-Donjon/rainbow (visited on 01/12/2023).

[DS] Datashader: Accurately render even the largest data. URL: https://datashader.org/ (visited on 10/04/2024).

[EPW10] Thomas Eisenbarth, Christof Paar and Björn Weghenkel. Building a side channel based disassembler. *Trans. Comput. Sci.*, 10:78–99, 2010. DOI: 10.1007/978-3-642-17499-5_4.

[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 *HOST 2010, Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 13-14 June 2010, California, USA*, pages 76–87. IEEE, 2010. DOI: 10.1109/HST.2010.5513110.

[FV12] Junfeng Fan and Ingrid Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, volume 6805 of *LNCS*, pages 265–282. Springer, 2012. DOI: 10.1007/978-3-642-28368-0_18.

[GLV01] Robert P Gallant, Robert J Lambert and Scott A Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *Annual International Cryptology Conference*, pages 190–200. Springer, 2001. DOI: 10.1007/3-540-44647-8_11.

[GOP22] Si Gao, Elisabeth Oswald and Dan Page. Towards micro-architectural leakage simulators: reverse engineering micro-architectural leakage features is practical. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part III*, pages 284–311, Trondheim, Norway. Springer-Verlag, 2022. ISBN: 978-3-031-07081-5. DOI: 10.1007/978-3-031-07082-2_11.

[Gou03] Louis Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In *PKC 2003*, volume 2567 of *LNCS*, pages 199–210. Springer, Heidelberg, January 2003. DOI: 10.1007/3-540-36288-6_15.

[GS21] Cesar Pereida Garcıa and Sampo Sovio. Size, speed, and security: An Ed25519 case study. In *NordSec 2021, Virtual Event, November 29-30, 2021*, volume 13115 of *LNCS*, pages 16–30. Springer, 2021. DOI: 10.1007/978-3-030-91625-1_2.

[GSM$^+$10] Sylvain Guilley, Laurent Sauvage, Julien Micolod, Denis Réal and Frédéric Valette. Defeating any secret cryptography with SCARE attacks. In *LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 273–293. Springer, Heidelberg, August 2010. DOI: 10.1007/978-3-642-14712-8_17.

[HR76] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Inf. Process. Lett.*, 5(1):15–17, 1976. DOI: 10.1016/0020-0190(76)90095-8.

[HV] Holoviews: Stop plotting your data – annotate your data and let it visualize itself. URL: https://holoviews.org/ (visited on 10/04/2024).

[HVM04] Darrel Hankerson, Scott Vanstone and Alfred Menezes. *Guide to elliptic curve cryptography*. Springer Professional Computing. Springer, New York, NY, January 2004. ISBN: 978-0-387-95273-4.

[IT03] Tetsuya Izu and Tsuyoshi Takagi. Exceptional procedure attack on elliptic curve cryptosystems. In *PKC 2003*, volume 2567 of *LNCS*, pages 224–239. Springer, Heidelberg, January 2003. DOI: 10.1007/3-540-36288-6_17.

[ITO+24] Vishnuvardhan V. Iyer, Aditya Thimmaiah, Michael Orshansky, Andreas Gerstlauer and Ali E. Yilmaz. A hierarchical classification method for high-accuracy instruction disassembly with near-field EM measurements. *ACM Trans. Embed. Comput. Syst.*, 23(1):10:1–10:21, 2024. DOI: 10.1145/3629167.

[Jan20] Jan Jancar. *PYECSCA: Reverse-engineering black-box Elliptic Curve Cryptography implementations via side-channels.* Master's thesis, Masaryk University, Brno, Czechia, 2020. URL: https://is.muni.cz/th/fjgay/.

[JIL] Senior Official Group Information Systems Security. Application of Attack Potential to Smartcards and Similar Devices. Joint Interpretation Library, November 2022. URL: https://www.sogis.eu/documents/cc/domains/sc/JIL-Application-of-Attack-Potential-to-Smartcards-v3.2.pdf.

[Jinja2] The Pallets Projects. Jinja. URL: https://jinja.palletsprojects.com/en/2.11.x/ (visited on 10/04/2024).

[JSS+20] Jan Jancar, Vladimir Sedlacek, Petr Svenda and Marek Sys. Minerva: The curse of ECDSA nonces. *IACR TCHES*, 2020(4):281–308, 2020. ISSN: 2569-2925. DOI: 10.13154/tches.v2020.i4.281-308.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:161–191, 1883.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987. ISSN: 0025-5718. DOI: 10.2307/2007884.

[Lew78] Art Lew. Optimal conversion of extended-entry decision tables with general cost criteria. *Commun. ACM*, 21(4):269–279, 1978. DOI: 10.1145/359460.359469.

[LTM] Tom St Denis and Contributors. Libtommath. URL: https://www.libtom.net/LibTomMath/ (visited on 10/04/2024).

[Mac61] Olin Lowe Macsorley. High-speed arithmetic in binary computers. *Proceedings of the IRE*, 49(1):67–91, 1961. DOI: 10.1109/JRPROC.1961.287779.

[MHL+22] Julien Maillard, Thomas Hiscock, Maxime Lecomte and Christophe Clavier. Towards fine-grained side-channel instruction disassembly on a system-on-chip. In *25th Euromicro Conference on Digital System Design, DSD 2022, Maspalomas, Spain, August 31 - Sept. 2, 2022*, pages 472–479. IEEE, 2022. DOI: 10.1109/DSD57027.2022.00069.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, Heidelberg, August 1986. DOI: 10.1007/3-540-39799-X_31.

[MMM14] Mehari Msgna, Konstantinos Markantonakis and Keith Mayes. Precise instruction-level side channel profiling of embedded processors. In *ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings*, volume 8434 of *LNCS*, pages 129–143. Springer, 2014. DOI: 10.1007/978-3-319-06320-1_11.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. ISSN: 00255718, 10886842. DOI: 10.2307/2007970.

[Mon87] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987. ISSN: 00255718, 10886842. DOI: 10.2307/2007888. (Visited on 01/12/2023).

[MSE$^+$20] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth and Nadia Heninger. TPM-FAIL: TPM meets timing and lattice attacks. In *USENIX Security 2020*, pages 2057–2073. USENIX Association, August 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm.

[MST$^+$13] Santi Martínez, Daniel Sadornil, Juan Tena, Rosana Tomàs and Magda Valls. On Edwards curves and ZVP-attacks. *Appl. Algebra Eng. Commun. Comput.*, 24(6):507–517, 2013. DOI: 10.1007/s00200-013-0211-2.

[MVV18] Alfred J. Menezes, Paul C. Van Oorschot and Scott A. Vanstone. *Handbook of applied cryptography*. CRC press, 2018. ISBN: 9780429466335.

[NAH21] Pouya Narimani, Mohammad Ali Akhaee and Seyedamin Habibi. Side-channel based disassembler for AVR micro-controllers using convolutional neural networks. In *ISCISC 2021, Isfahan, Iran, September 1-2, 2021*, pages 75–80. IEEE, 2021. DOI: 10.1109/ISCISC53448.2021.9720466.

[NSS$^+$17] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec and Vashek Matyas. The return of Coppersmith's attack: Practical factorization of widely used RSA moduli. In *ACM CCS 2017*, pages 1631–1648. ACM Press, October 2017. DOI: 10.1145/3133956.3133969.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In *COSADE 2014, Paris, France, April 13-15, 2014.* Volume 8622 of *LNCS*, pages 243–260. Springer, 2014. DOI: 10.1007/978-3-319-10175-0_17.

[Ora23] Oracle. Oracle Java Card technology, 2023. URL: https://www.oracle.com/java/java-card/ (visited on 20/10/2023).

[Pip76] Nicholas Pippenger. On the evaluation of powers and related problems. In *17th Annual Symposium on Foundations of Computer Science (SFCS 1976)*, pages 258–263. IEEE, 1976. DOI: 10.1109/SFCS.1976.21.

[Pol65] Solomon L. Pollack. Conversion of limited-entry decision tables to computer programs. *Commun. ACM*, 8(11):677–682, 1965. DOI: 10.1145/365660.365681.

[PXJ$^+$18] Jungmin Park, Xiaolin Xu, Yier Jin, Domenic Forte and Mark M. Tehranipoor. Power-based side-channel instruction-level disassembler. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, 119:1–119:6. ACM, 2018. DOI: 10.1145/3195970.3196094.

[QS02] Jean-Jacques Quisquater and David Samyde. Automatic code recognition for smart cards using a Kohonen neural network. In *Proceedings of the 5th Conference on Smart Card Research and Advanced Application Conference - Volume 5*, CARDIS'02, page 6, San Jose, CA. USENIX Association, 2002. DOI: 10.5555/1250988.1250994.

[RDG$^+$08] Denis Réal, Vivien Dubois, Anne-Marie Guilloux, Frédéric Valette and M'hamed Drissi. SCARE of an unknown hardware Feistel implementation. In *CARDIS 2008, London, UK, September 8-11, 2008*, volume 5189 of *LNCS*, pages 218–227. Springer, 2008. DOI: 10.1007/978-3-540-85893-5_16.

[RLM$^+$21] Thomas Roche, Victor Lomné, Camille Mutschler and Laurent Imbert. A side journey to Titan. In *USENIX Security 2021*, pages 231–248. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/roche.

[RR13] Matthieu Rivain and Thomas Roche. SCARE of secret ciphers with SPN structures. In *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 526–544. Springer, Heidelberg, December 2013. DOI: 10.1007/978-3-642-42033-7_27.

[SBO+15] Daehyun Strobel, Florian Bache, David F. Oswald, Falk Schellenberg and Christof Paar. Scandalee: A side-channel-based disassembler using local electromagnetic emanations. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, pages 139–144. ACM, 2015. DOI: 10.5555/2755753.2755784.

[SCJ+21] Vladimir Sedlacek, Jesús-Javier Chi-Domínguez, Jan Jancar and Billy Bob Brumley. A formula for disaster: A unified approach to elliptic curve special-point-based attacks. In *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 130–159. Springer, Heidelberg, December 2021. DOI: 10.1007/978-3-030-92062-3_5.

[SDB+24] Petr Svenda, Antonin Dufka, Milan Broz, Roman Lacko, Tomas Jaros, Daniel Zatovic and Josef Pospisil. TPMScan: A wide-scale study of security-relevant properties of tpm 2.0 chips. In *IACR Transactions on Cryptographic Hardware and Embedded Systems*, volume 2024, No. 2, pages 714–734, 2024. DOI: 10.46586/tches.v2024.i2.714-734.

[Sil86] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, number 106 in Graduate Texts in Mathematics. Springer-Verlag, 1986. ISBN: 978-0-387-09493-9.

[SJS20] Vladimir Sedlacek, Jan Jancar and Petr Svenda. Fooling primality tests on smartcards. In *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 209–229. Springer, Heidelberg, September 2020. DOI: 10.1007/978-3-030-59013-0_11.

[SKN+22] Petr Svenda, Rudolf Kvasnovsky, Imrich Nagy and Antonin Dufka. JCAlgTest: Robust identification metadata for certified smartcards. eng. In *19th International Conference on Security and Cryptography*, pages 597–604, Lisabon. INSTICC, 2022. ISBN: 978-989-758-590-6. DOI: 10.5220/0000163500003283.

[SSG11] Manuel San Pedro, Mate Soos and Sylvain Guilley. FIRE: Fault injection for reverse engineering. In *WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011*, volume 6633 of *LNCS*, pages 280–293. Springer, 2011. DOI: 10.1007/978-3-642-21040-2_20.

[Str64] Ernst Gabor Straus. Problems and solutions: Addition chains of vectors. *American Mathematical Monthly*, 71(806-808), 1964.

[TQP+14] Ming Tang, Zhenlong Qiu, Hongbo Peng, Xiaobo Hu, Yi Mu and Huanguo Zhang. Toward reverse engineering on secret S-boxes in block ciphers. *Sci. China Inf. Sci.*, 57(3):1–18, 2014. DOI: 10.1007/S11432-013-5053-9.

[vGB22] Jurian van Geest and Ileana Buhan. A side-channel based disassembler for the ARM-Cortex M0. In *ACNS 2022 Satellite Workshops, Rome, Italy, June 20-23, 2022*, volume 13285 of *LNCS*, pages 183–199. Springer, 2022. DOI: 10.1007/978-3-031-16815-4_11.

[VMA20] Shahram Vafa, Massoud Masoumi and Amir Amini. An efficient profiling attack to real codes of PIC16F690 and ARM cortex-m3. *IEEE Access*, 8:222520–222532, 2020. DOI: 10.1109/ACCESS.2020.3043395.

[vWB11] Jasper G. J. van Woudenberg, Marc F. Witteman and Bram Bakker. Improving differential power analysis by elastic alignment. In *CT-RSA 2011*, volume 6558 of *LNCS*, pages 104–119. Springer, Heidelberg, February 2011. DOI: 10.1007/978-3-642-19074-2_8.

[VWG07] Dennis Vermoen, Marc F. Witteman and Georgi Gaydadjiev. Reverse engineering Java Card applets using power analysis. In *WISTP 2007, Heraklion, Crete, Greece, May 9-11, 2007*, volume 4462 of *LNCS*, pages 138–149. Springer, 2007. DOI: 10.1007/978-3-540-72354-7_12.

[WR20] Pete Whittaker and Tom Randall. Wide Boyz. 2020. URL: https://www.youtube.com/@WideBoyz (visited on 19/12/2023).

[ZLL12] Fangguo Zhang, Qiping Lin and Shengli Liu. Zero-value point attacks on Kummer-based cryptosystem. In *ACNS 12*, volume 7341 of *LNCS*, pages 293–310. Springer, Heidelberg, June 2012. DOI: 10.1007/978-3-642-31284-7_18.