MASARYK UNIVERSITY

Faculty of Informatics

The Double-Edged Curve

Cryptographic Implications of Algebraic Structures in Elliptic Curves

Doctoral thesis of

Vojtěch Suchánek

Supervisor:

prof. RNDr. Václav Matyáš, M.Sc., Ph.D.

Brno, 2026

Declaration

I hereby declare that this thesis is my own original work, created independently and extensively using publications that I (co-)authored during my Ph.D. research. All sources, literature, and quoted materials used in its preparation are properly acknowledged and fully referenced.

Throughout the preparation of this thesis, I used the following tools:

- GPT-5 to rephrase drafts of existing paragraphs.
- Grammarly for proofreading the text.

I declare that I have used these tools in compliance with the principles of academic integrity and take responsibility for the content of this work.

Vojtěch Suchánek,
March 8, 2026.

Abstract

This thesis investigates how the algebraic structure of elliptic curves shapes the practical security and performance of real-world cryptographic systems. Rather than treating elliptic curves as abstract mathematical objects, we analyze how their structural properties interact with concrete implementations, side-channel attacks, and standardization processes.

We conduct a systematic, tool-based analysis of standardized elliptic curves, comparing them against large-scale simulations of "generic" curves to identify unexpected structural deviations. Our findings shed light on the trustworthiness and diversity of standardized parameters.

Focusing on side-channel resilience, we develop new variants of the zero-value point (ZVP) attack and introduce improved techniques for solving the dependent coordinates problem, particularly on curves with fast endomorphisms such as `secp256k1`. These advances enable significantly stronger key-recovery attacks against multiscalar multiplication implementations.

On the constructive side, we use fast endomorphisms to accelerate signature verification, achieving substantial performance improvements for Schnorr, BLS, and ECDSA signatures.

Bridging theory and practice, we present novel methods for reverse-engineering black-box ECC implementations using side-channel leakage and malformed inputs. We reveal widespread deficiencies in point validation and scalar randomization across real-world libraries and smartcards, and we show how to recover detailed information about their internal countermeasures.

Taken together, these results demonstrate that the rich algebraic structure of elliptic curves is neither purely a blessing nor a liability, but a double-edged sword that can both enable efficiency and expose vulnerabilities.

Acknowledgments

Although my name appears on the title page, this thesis is the result of collaborations with many people who have shaped my research and my experience throughout my Ph.D.

I am deeply grateful to Vashek, Petr, Marek, Jano, Łukasz, Tonda, Vlada, and Honza for the time, effort, and enthusiasm they invested in the work described in this thesis. I owe special thanks to Vashek for his guidance, support, and remarkable patience over the years, as well as for creating the inspiring and supportive environment that is CRoCS. The memories from our countless off-sites, conferences, and beer tours will stay with me for life.

I would also like to thank Jirka, Veronika, Adam, Tamara, and the rest of the lab. Even though we did not end up publishing a paper together, your influence on my Ph.D. journey has been just as meaningful as any publication.

Finally, I am deeply thankful to my friends and family, especially my parents, who raised me and supported me throughout my years of study.

The research was supported and funded by the following institutions and grants:

- Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development, and Innovations Infrastructures.
- Cisco University Research Program Fund, an advised fund of Silicon Valley Community Foundation.
- Ph.D. Talent Scholarship – funded by the Brno City Municipality.
- Tools for AI-enhanced Security Verification of Cryptographic Devices (AI-SecTools VJ02010010).
- Cyber-security Excellence Hub in Estonia and South Moravia (CHESS, 101087529) funded by European Union.

Contents

1 Introduction 1

2 What is the point? 7
- 2.1 Elliptic curve . . . 7
- 2.2 Scalar multiplication . . . 10
- 2.3 Protocols . . . 14
- 2.4 Point validation . . . 15
- 2.5 ZVP attack . . . 18
- 2.6 Fast endomorphisms . . . 21

3 Dissecting elliptic curve standards 27
- 3.1 Introduction . . . 27
- 3.2 Overview of standard curve generation . . . 29
- 3.3 Methodology . . . 32
- 3.4 Traits . . . 40
- 3.5 DiSSECT . . . 46
- 3.6 Conclusions . . . 47

4 Side-channel attacks on endomorphisms 49
- 4.1 Introduction . . . 49
- 4.2 Classification of formulas . . . 50
- 4.3 ZVP-GLV attack . . . 56
- 4.4 Conclusions . . . 67

5 Faster signature verification with endomorphism 69
- 5.1 Introduction . . . 69
- 5.2 Background . . . 71
- 5.3 Verification using 3-dimensional GLV . . . 76
- 5.4 The dependent coordinates problem . . . 89
- 5.5 Conclusions and future work . . . 90

6 Reverse-engineering ECC using side-channel attacks 91
- 6.1 Introduction . . . 91

6.2 Reverse-engineering . . . 94
6.3 Analysis of open-source libraries . . . 96
6.4 Reverse-engineering techniques . . . 101
6.5 Toolkit . . . 105
6.6 Results . . . 107
6.7 Conclusions . . . 113

7 Lying to ECC implementations 115
7.1 Introduction . . . 115
7.2 Related work . . . 117
7.3 Methodology . . . 118
7.4 Results . . . 122
7.5 Reverse-engineering scalar randomization . . . 127
7.6 Impact, limitations and discussion . . . 136

8 Conclusions 139

Appendices 141

A Academic publications 143

B Supplementary Algorithms, Tables, and Figures 147

Bibliography 153

1 Introduction

Miller [133] and Koblitz [114] independently proposed elliptic curves over finite fields for cryptographic protocols in 1985. In the introduction of [133], Miller explains:

> *"It is my intent to show that elliptic curves have a rich enough arithmetic structure so that they will provide a fertile ground for planting the seeds of cryptography."*

He then supports his proposition by instantiating the Diffie-Hellman key exchange protocol on an elliptic curve. The protocol works over any commutative group, but one needs to choose one where the discrete logarithm problem (DLP) is difficult and the group operation can be efficiently computed. Up to that point, the natural and popular choice was the multiplicative group of a finite field as in the digital signature standard [76]. However, the presumed hardness of this DLP was undercut with the index calculus attack [118] with its subexponential complexity. Miller argued using the Neron-Tate height function that it would be extremely unlikely for an index calculus attack on elliptic curves to ever work. The reason was, roughly speaking, that there are no 'small points' on an elliptic curve that could form a factor base for the attack. Hence, the point of Miller's seminal paper was not just to demonstrate a rich enough structure in elliptic curves. Rather, that elliptic curves provide just enough structure for the protocol and not more for the index calculus.

The discussion over the (lack of) algebraic structure in curves that could weaken the DLP then continued [113, 115]. The best known algorithms for the DLP on an elliptic curve are the baby-step giant-step and Pollard's ρ algorithms that require roughly $O(\sqrt{n})$ steps, where n is the size of the largest prime subgroup. These algorithms are generic in the sense that they operate over any group and ignore the fact that the group is defined by points on an elliptic curve. Shoup [193] showed that if we treat curves as generic groups we cannot do any better than $O(\sqrt{n})$. More precisely, he considered the formal generic group — an oracle that encodes the group elements as random strings and computes the group

operations without leaking any further information. Treating elliptic curves as generic groups requires precision and care, as the opposite might lead to problematic security proofs [42, 43] as pointed out by Stern et al. [200]. As opposed to the generic group, the representation of points on a curve is easily distinguishable from random strings since the coordinates satisfy a fixed Weierstrass equation. Similarly, the neutral element is special not only algebraically, but its representation is $(0 : 1 : 0)$. Finally, to compute the negation it is enough to negate the y-coordinate. The last property not only breaks the generic group assumptions, it can be used to speed up the Pollard's ρ algorithm by a factor of $\sqrt{2}$ [212].

The DLP in a large prime subgroup on a curve is known to be weaker than in the generic group only in two special cases. The first case is called anomalous and happens when the underlying prime p and the size of the subgroup n are equal. Here, the discrete logarithm problem is trivial. While the uniqueness of this case can be easily spotted by checking $p = n$, the same argument cannot be made for the second case. In the second case, the MOV attack [128] applies when the multiplicative order of p modulo n is small — a condition that we can see only when we know what we are looking for. After all, the curve that was proposed in Miller's original paper is vulnerable to this attack as it belongs to a class of supersingular curves. He correctly predicted in the conclusion that it may be prudent to avoid such curves because the extra structure might lead to a better DLP algorithm in the future. However, the extra structure he pointed out was not the supersingularity, but complex multiplication (CM) by $\sqrt{-1}$. This on its own is not a problem as there are curves with this CM that have strong DLP. Not only that, the class of curves with CM by $\sqrt{-1}$ or $\sqrt[3]{1}$, called GLV curves [82], have been shown to provide fast scalar multiplication. A prominent example of such a curve is the Bitcoin curve `secp256k1`. So while one special property (supersingularity) turned out to be a devastating weakness the other (CM) is present in every signature on the Bitcoin blockchain.

Nonetheless, it has been proven many times that the security of an elliptic curve protocol does not depend only on the number of steps required to solve the underlying DLP. To set up the simple Diffie-Hellman protocol, the developer needs to implement multiple layers of primitives including the coordinate system, addition formulas, scalar multiplication algorithm and the curve model. Elliptic curves offer many options for every layer (e.g., there are almost 300 addition formulas [28]). The literature contains decades of tricks and optimizations using every possible

useful property of elliptic curves. For instance, the quick negation map that can speed up the Pollard's ρ algorithm is often used to accelerate the scalar multiplication using signed representation. However, a fast but naive implementation without input validation would immediately fall prey to an invalid curve attack [31, 7], twist attack [78], the small subgroup attack [120] or the degenerate curve attack [147] as it happens in real-world libraries and servers [103]. In the invalid curve attack, the attacker steps outside the mathematical model and forces the implementation to operate with points outside the curve. Unfortunately, elliptic curves are flexible enough and often cooperate with such unexpected computations. Many addition formulas compute the sum of two points $P + Q$ over any elliptic curve. In return for these problems, elliptic curves offer several solutions including more strict formulas or a change of curve model as in the popular Curve25519 [24].

Implementations that correctly compute the shared secret over the correct subgroup can still be insecure as is often demonstrated with side-channel attacks [64]. By stepping outside the theoretical model even further, an attacker observes physical characteristics such as power consumption, timing behavior, or electromagnetic radiation. An unprotected implementation against these side-channel leaks any secret information including the private keys. The algebraic and geometric properties in elliptic curves again come with a multitude of solutions. As an example, one can freely protect the computation using randomization by multiplying coordinates of points with a random number. This all works thanks to the fact that projective coordinates are not random strings as in the generic group but correspond to lines in a projective space.

After forty years of elliptic curve cryptography (ECC), it is clear that the fertile ground of rich arithmetic structures described by Miller and Koblitz gave rise to a complex cryptographic area. This thesis investigates the diversity of elliptic curves in cryptographic standards, libraries and embedded devices by means of a systematic, tool-based analysis. To better understand the extent to which this diversity manifests itself in practice, we formulate the following research question:

RQ I
How rich is the landscape of elliptic curve properties in cryptographic implementations?

While identifying structural diversity is important, understanding its practical consequences is equally critical. This thesis therefore examines how selected properties influence practical aspects of implementations such as side-channel attacks, reverse-engineering, performance, or trust. To systematically examine these practical implications, we formulate the following research question:

> **RQ II**
> *What are the implications of these properties on the practical security?*

This thesis addresses these questions in five chapters, each backed by a publication presenting the following contributions:

- Chapter 3 addresses the challenge of trusting elliptic curves that have been standardized through non-transparent processes. To this end, we introduce a systematic, tool-based methodology for analyzing standardized elliptic curves and statistically comparing them against the expected behavior of generic curves, with the aim of identifying significant deviations. In the chapter, we present the largest publicly available database of standardized elliptic curves. We then simulate more than 250 000 curves by faithfully reproducing the generation processes of four major standards. For each curve, we compute several algebraic and structural properties and analyze them using automated techniques in order to detect deviations in standardized curves that may indicate unexpected structure or potential weaknesses.
- Chapter 4 examines the impact of elliptic curve properties on resistance to side-channel attacks, focusing in particular on the zero-value point (ZVP) attack. While many side-channel vulnerabilities in elliptic curve cryptography can be mitigated through coordinate randomization, this attack fundamentally bypasses such countermeasures. In this chapter, we design a new method for solving the dependent coordinates problem (DCP) on GLV curves, including the Bitcoin curve `secp256k1`, significantly outperforming previous approaches. This enables a new variant of the ZVP attack targeting multiscalar multiplication, allowing the recovery of twice as many key bits compared to the classical ZVP attack. We demonstrate a recovery of up to 63% of the private key for the interleaving algorithm used in multiscalar multiplication. Finally, we analyze the largest known database of elliptic curves and addition

formulas, comprising more than 14 000 combinations, and provide the first systematic classification of their resistance to the ZVP attack.

- Chapter 5 continues the exploration of GLV curves and their properties and shows how fast endomorphisms can accelerate the verification of Schnorr signatures. Our technique applies to single Schnorr signature verification, batch verification, and BLS signature verification in the Blind Diffie–Hellman key exchange. Experimental results demonstrate performance improvements of approximately 20%, 13%, and 27% for these three verification algorithms, respectively, on AMD x86 and ARM Cortex-M4 platforms. We further show that the technique can be extended to accelerate ECDSA signature verification, provided that one additional bit is appended to the signature. As a byproduct, we analyze the practical performance of three-dimensional lattice reduction algorithms, which are central to multi-scalar decompositions. We experimentally compare Semaev's algorithm, which offers the best asymptotic complexity, with the simpler Lagrange algorithm. Our results show that, despite its simplicity, Lagrange's algorithm is nearly twice as fast as Semaev's in practice. Finally, we close the study of the DCP from Chapter 4 by reducing the asymptotical complexity from $O(n)$ to $O(\sqrt{n}\log^3 n)$.
- Chapter 6 addresses the gap between side-channel attack models and real-world attackers, who often have only black-box access to elliptic curve cryptographic implementations. While many side-channel attacks assume detailed knowledge of implementation choices, practical ECC implementations involve a complex combination of curve models, coordinate systems, addition formulas, and scalar multiplication algorithms. In this chapter, we first analyze open-source ECC libraries to characterize real-world implementation choices and then study the combinatorial space of possible ECC implementations. Building on this analysis, we introduce novel methods for reverse-engineering black-box ECC implementations using ZVP attacks.
- Chapter 7 follows the motivation for reverse-engineering from Chapter 6. Using ECTester — a tool for testing black-box elliptic curve cryptographic implementations — we probe the behavior of implementations under known attacks as well as non-standard inputs and elliptic curve parameters. We show that several ECC libraries and most tested smartcards fail to verify the order of input points and incorrectly handle the point at infinity. Based on these findings, we

design new techniques for reverse-engineering scalar randomization countermeasures, allowing us to distinguish between group scalar randomization, additive, multiplicative, and Euclidean splitting. Our techniques require neither side-channel measurements nor source-code access; they rely solely on the ability to supply custom domain parameters. They recover not only the size but also the exact value of the random mask used. Using these methods, we successfully reverse-engineer countermeasures on 13 cryptographic smartcards from five major manufacturers. Finally, we discuss possible mitigations and assess whether such reverse engineering can be effectively prevented in practice.

2 What is the point?

This chapter serves as a brief introduction to *some* concepts in ECC that will be essential for the thesis. We will not pretend that this is a good introductory text on ECC for anyone with no prior knowledge of elliptic curves. The point of the chapter is to briefly recall concepts that would require several chapters to properly define and explain. For an excellent in-depth introduction to elliptic curves and ECC, we have to recommend [213], [194] and [95] — books without which this thesis would probably not exist.

> *In mathematics you don't understand things.*
> *You just get used to them.* (*John von Neumann*)

2.1 Elliptic curve

An elliptic curve is an abelian variety of dimension one. While this definition enjoys the short length and full generality, it hides all the important details needed for cryptographic implementations — how do we represent the elliptic curve and compute the group operation. We will instead use a long descriptive definition restricted to curves over prime fields $\mathbb{F}_p$ of characteristic $p > 3$. We can leave out curves over other fields as they remain in the minority with respect to cryptographic applications. An elliptic curve over $\mathbb{F}_p$ is defined by the equation

$$E/\mathbb{F}_p \colon y^2z = x^3 + axz^2 + bz^3,$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0$. The set of projective points $(x : y : z) \in \mathbb{P}^3(\mathbb{F}_p)$ on E forms a finite commutative group $E(\mathbb{F}_p)$ with the neutral element $\mathcal{O} = (0 : 1 : 0)$. For any point $P \in E(\mathbb{F}_p)$, denote its coordinates as $P = (x_P : y_P : z_P)$. We will often use the affine form of

the curve $y^2 = x^3 + ax + b$ with the affine points $(x_P, y_P) = (x_P : y_P : 1)$. The addition $P + Q$ can be defined using the following formula [60]:

$$\begin{aligned}(x_{P+Q} : y_{P+Q} : z_{P+Q}) &= (vA : B - v^3 y_P z_Q : v^3 z_P z_Q) \\ v &= x_Q z_P - x_P z_Q, \quad A = u^2 z_P z_Q - v^3 - 2v^2 x_P z_Q, \\ u &= y_Q z_P - y_P z_Q, \quad B = u(v^2 x_P z_Q - A).\end{aligned} \tag{2.1}$$

The formula incorrectly outputs $(0 : 0 : 0)$ (not an element of the projective space) if $P = Q$ or one of P, Q is the neutral point. Unfortunately, this is not just a bad choice of formula. Every formula has a pair of exceptional points in $(E \times E)(\overline{\mathbb{F}}_p)$ on which the formula does not work. There are two approaches to solving this incompleteness. First, one can use more than one addition formula to cover all the necessary points, e.g., an add formula and a special (doubling) formula for $P = Q$, $P = \mathcal{O}$ and $Q = \mathcal{O}$. The second approach is to use formulas [170] that have been designed to have the exceptional points outside of the main subgroup of points that are commonly used in cryptography.

The Explicit Formulas Database (EFD) [28] collects an extensive list of formulas for addition on curves. It contains addition formulas for other coordinate systems apart from the standard projective coordinates as well as formulas for other curve models including the Montgomery model and the (Twisted) Edwards model. To illustrate, with the Montgomery model of an elliptic curve

$$M : By^2z = x^3 + Ax^2z + xz^2,$$

we can use the xz coordinate system $P = (x_P : z_P)$ without the y-coordinate. Since there are addition formulas on the Montgomery curve that do not need the y-coordinate, this loss of information is not a problem. On the contrary, the formulas are faster and the implementation is less prone to errors [24]. Apart from simple addition or doubling of points, there are formulas for more complicated operations such as the ladder operation that takes on input three points $P, Q, P - Q$ and outputs the two points $P + Q, 2P$. Overall, the decades of research and practical implementations in ECC produced a plethora of formulas, each optimized for specific goals such as speed, size, side-channel resistance, completeness, and generality.

To compute something meaningful for cryptography, we have to extend point addition into scalar multiplication (SM) kP, naturally as k-fold

addition $P + \cdots + P$. As we will see, scalar multiplication is the critical operation with respect to both the speed and the security. The scalar k is often secret, while the points P and kP are public. The secrecy of k is protected by the discrete logarithm problem (ECDLP): given P and kP, it should be difficult to find k. Elliptic curves are generally a good source of a hard ECDLP and a fast scalar multiplication. However, there are a few pitfalls one has to avoid (SafeCurves website [29] presents a good overview):

Large prime. The characteristic p of the underlying field $\mathbb{F}_p$ should be large enough. The Hasse theorem dictates that p is not far from the size of the whole group $\#E(\mathbb{F}_p)$. More precisely, the difference between these two quantities, the so-called trace of Frobenius $t = p + 1 - \#E(\mathbb{F}_p)$, satisfies $|t| \leq 2\sqrt{p}$. In practical terms, this means that p and $\#E(\mathbb{F}_p)$ will differ only in the lower half of binary bits. Hence, if we want to avoid the attacker successfully brute-forcing $E(\mathbb{F}_p)$ to solve ECDLP, we need to start with a large p. In practice, p has hundreds of bits[1], most commonly $\ell(p) = 256$.

Large order. The point P used for the scalar multiplication should lie in a large prime subgroup. This is a protection against the Pohlig-Hellman attack [163] which is able to break down the ECDLP into smaller subgroups. Naturally, a large prime subgroup does not have any (non-trivial) subgroups, making the attack ineffective. We will denote the size of the large subgroup as n and the remaining cofactor $\frac{\#E(\mathbb{F}_p)}{n}$ as h.

Trace. There are two values of the Frobenius trace t that cause problems: 0 and 1. When $t = 1$ the curve is called anomalous and the Semaev-Satoh-Araki-Smart additive transfer is able to reduce the ECDLP into a discrete logarithm problem in the additive group of a finite field [195, 188, 176] — problem that is trivial to solve. Similarly, if $t = 0$, then there is a transfer of the ECDLP into the multiplicative group $\mathbb{F}_p^\times$, $\mathbb{F}_{p^2}^\times$ or $\mathbb{F}_{p^3}^\times$. The multiplicative DLP is then easily solvable for the typical bit-lengths of p.

Pairings. The transfer into $\mathbb{F}_{p^e}^\times$ with $t = 0$ can be generalized for any curve [187, 79, 128]. If e is the order of p in $\mathbb{F}_n^\times$ (also called the embedding degree), the ECDLP can be transferred to $\mathbb{F}_{p^e}^\times$. The expected embedding degree for a random curve is of size $\approx n$, making the field too large to solve the multiplicative DLP. However, any curve with low embedding degree (i.e., $e < 20$) is vulnerable. The main tool for the transfer is the

[1]For any integer x, we will denote the bit-length of x as $\ell(x)$.

bilinear pairing which has found its use as a verification of BLS signatures [15] or in zero-knowledge proofs based on elliptic curves [88].

Large discriminant. This final recommendation is not connected to any real vulnerability. Bernstein et al. [29] suggest the CM discriminant (i.e., the absolute discriminant of the field $\mathbb{Q}(\sqrt{t^2-4p})$) to be at least 2^{100}. They warn against a possible speed-up to the rho method, but this is only applicable to binary curves [81]. Similarly, the Brainpool standard [70] recommends a large class number (quantity related to the discriminant). The standard argues that some works [107, 98] can be seen as an argument for this condition.

2.2 Scalar multiplication

Scalar multiplication is the main performance bottleneck in the classical ECC protocols. Hence, the computation of kP must not only ensure the protection of the secret scalar k, but also needs to be fast. This thesis deals with both of these aspects of scalar multiplication.

Similarly as for the formulas, the literature contains numerous algorithms for scalar multiplication (we will call these algorithms *multipliers* for short). Usually, multipliers iteratively update some accumulator point based on the digits of k in some representation. Each iteration updates the accumulator using formulas for point addition (add), doubling (dbl), ladder (ladd) or others.

Example 1. As an example of a basic multiplier, Algorithm 1 shows the double-and-add multiplier with the NAF representation[2]. The NAF represents scalars with $1, 0$ and -1, e.g., $31 = 2^5 - 1 = 10000\text{-}1_{\text{NAF}}$. The multiplier iterates over NAF of k and for each digit, one doubling operation dbl is computed and if the digit is non-zero one add as well. In the example with $k = 31$, the multiplier would perform six dbl and two add operations. One of the add operations technically corresponds to a subtraction ($\mathsf{add}(Q, -P)$), but a negation of point is essentially costless: $-P = -(x_P, y_P) = (x_P, -y_P)$. This quick negation is an important property of curves, very often used in multipliers. The reason for the

[2]Depending on the representation, we denote (signed) digits by upper indices in brackets, i.e., the least significant digit of d is $d^{(0)}$. By $d^{\geq i}$ we denote the prefix given by the upper digits of d down to $d^{(i)}$. In particular, for binary representation, $d^{(0)}$ is the least significant bit.

NAF rather than simple binary is the fact that NAF has lower hamming weight (number of nonzero bits). As a result, the same multiplier with binary representation $31 = 11111_2$ would require five dbl but also five add operations. In total this is two more curve operations than with NAF. The details matter as the cost of one add is usually higher than one dbl.

Algorithm 1: Double-and-add.

Input: $P \in E$, m-digit scalar d in the NAF representation.
Output: kP

$Q \leftarrow \mathcal{O}$
for $i = m - 1$ down to 0 **do**
  $Q \leftarrow \mathsf{dbl}(Q)$
  if $d^{(i)} = 1$ **then** $Q \leftarrow \mathsf{add}(Q, P)$
  if $d^{(i)} = -1$ **then** $Q \leftarrow \mathsf{add}(Q, -P)$
return Q

The following paragraphs provide a summary of the most common types of multipliers. This categorization is intentionally vague, as multipliers might combine ideas from several types, for more, see [59, 129, 95].

- **Basic** scalar multipliers scan the scalar (signed) bit by bit in either left-to-right (LTR) or right-to-left (RTL) fashion. The double-and-add (Algorithm 1) is the prominent example.
- **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 multipliers are parameterized by their window size $w \in \mathbb{N}$. The multipliers also perform precomputation, usually to compute all odd multiples of the point smaller than $2^w - 1$. This precomputation is cheap, so it can also be applied in case of a variable point.
- **Comb** multipliers follow a similar idea as the window-based with the difference that windows are not consecutive subsequences of bits of k, but are distributed in a regular comb-like pattern across the whole binary representation of k. Combs are intended for fixed-point scenarios and are generally the fastest scalar multipliers.

- **Ladder** multipliers are very similar to basic scalar multipliers in that they scan the scalar bit by bit and do not perform any precomputation. In each iteration, they update two accumulators using the ladd operation.

2.2.1 Side-channel countermeasures

Even if the discrete logarithm on the curve is difficult, the physical implementation of the multiplier may reveal more than intended. Side-channel attacks (SCA) exploit measurable effects of computation, such as timing variations or power traces, to infer information about the scalar. Let us now briefly present several modifications to multipliers that serve as countermeasures against side-channel attacks. See [73, 16, 64] for a detailed overview of side-channel attacks on ECC implementations.

Dummy operations. Some scalar multipliers are not regular, i.e., the sequence of add and dbl formula applications depends on the processed scalar. The simple double-and-add multiplier (Algorithm 1) is an example of this. To protect these multipliers from simple power analysis, dummy operations can be used. This transforms the simple double-and-add multiplier to double-and-add-always. This has the effect of introducing dummy values in the scalar multiplication that are not used to compute the final result.

Complete formulas. While complete formulas are not a side-channel countermeasure as such, they allow simpler exception-less handling of the point at infinity. For example, avoiding Minerva-style [105] leakage of the bit-length of the scalar in a side-channel-free manner is much easier with complete formulas. Furthermore, using complete formulas, scalar multipliers can safely handle scalars larger than the curve order without issues due to the potential to reach the point at infinity.

Scalar randomization. There are several side-channel countermeasures that randomize the scalar used in scalar multiplication. They sample some additional randomness and expand a single scalar multiplication into several multiplications or a single larger one. Figure 2.1 shows an overview of the techniques. We consider four popular techniques: group scalar randomization (GSR) [61], additive [58], multiplicative [205], and Euclidean [55] splitting. Note that the countermeasures have different *expansion ratio*, the amount and size of scalar multiplications they expand a single l-bit scalar multiplication into. For GSR and multiplicative

splitting, assuming they sample b bits of randomness, the ratios are $l \to 1 \times (l + b)$ and $l \to (1 \times b) + (1 \times l)$, respectively. For additive splitting the ratio is $l \to 2 \times l$ (i.e., two l-bit scalar multiplications). For Euclidean splitting, the ratio is $l \to 3 \times \frac{1}{2}l$.

Group scalar randomization

function $\textsc{Mult}(G, k)$
  $r \xleftarrow{\$} \{0, 1, \ldots, 2^b\}$
  return $(k + rn)G$

Additive splitting

function $\textsc{Mult}(G, k)$
  $r \xleftarrow{\$} \mathbb{Z}_n^*$
  return $(k - r)G + rG$

Euclidean splitting

function $\textsc{Mult}(G, k)$
  $r \xleftarrow{\$} \{0, 1, \ldots, 2^{\lfloor \log_2(n)/2 \rfloor}\}$
  $S \leftarrow rG$
  $k_1 \leftarrow k \mod r$
  $k_2 \leftarrow \left\lfloor \frac{k}{r} \right\rfloor$
  return $k_1 G + k_2 S$

Multiplicative splitting

function $\textsc{Mult}(G, k)$
  $r \xleftarrow{\$} \{0, 1, \ldots, 2^b\}$
  $S \leftarrow rG$
  return $(kr^{-1} \mod n)S$

Figure 2.1: An overview of scalar randomization countermeasures, showing the multiplication of point G with scalar k while sampling a random mask of b bits.

Coordinate and curve randomization. Apart from randomizing the scalar, one can also randomize the point coordinates, as a single affine point can be represented by many projective ones. In standard projective coordinates, point $P = (X : Y : Z)$ can be randomized using $\lambda \xleftarrow{\$} \mathbb{F}_p^*$ as $(\lambda X : \lambda Y : \lambda Z)$. This works analogously in other coordinate systems. Similarly, one can randomize computation on a whole curve by taking an isogenous curve, transferring the computation there, and transferring the result back.

Point blinding. Finally, the point blinding countermeasure, of which there are several variants, modifies the scalar multiplication of kP by adding a random point R to point P before the start of the multiplication and then subtracts $S = kR$ at the end. The points R and S may be precomputed for a fixed keypair and adjusted after each multiplication. Some variants of this countermeasure subtract the point S directly during the main scalar multiplication loop and do not require an additional operation.

2.3 Protocols

This thesis focuses on classical ECC protocols including the Diffie-Hellman key-exchange protocols (ECDH and X25519) and the digital signature protocols (ECDSA, EdDSA, BLS and Schnorr signatures). Generally, the Diffie-Hellman protocol consists of two subroutines: `KeyGen` and `Derive` which compute the keypair and the shared secret, respectively. The digital signature protocols use `KeyGen` and two further subroutines: `Sign` and `Verify` that compute and verify the signature of a message, respectively. We will now briefly recall important details of the protocols.

Let E be an elliptic curve as defined above. Denote G the generator of the prime subgroup of order n, H a cryptographic hash function, and m any message. For all the protocols, `KeyGen` computes a private key $d \in [1, n]$ and the public key $P = dG$, though the details might differ.

ECDH. To derive the shared secret, each party computes $S = kP$, where P is the other party's public key, after which some key derivation or hash function is applied to x_S. **X25519** is essentially the same as ECDH, but optimized for one specific curve Curve25519 with some lower level differences. For example, the public key is only the x-coordinate of the P and the private key is always a multiple of 8.

Schnorr signature. Schnorr signature over E is a pair (R, s), where $R = rG$ for a random $r \in [1, n]$ and $s = r + H(R, m)d \pmod{n}$ computed in `Sign`. Here, R is encoded using only x_R for the hash function. By checking the equality $sG = R + H(R, m)P$ the signature is then verified. To save space, the signature (R, s) can contain only the coordinate x_R and one additional bit signaling the parity of y_R, which is then used for the recovery of $y_R = \sqrt{x_R^3 + ax_R + b}$ (another option is to force the bit during the generation of $R = rG$). Note that there is another variant of the Schnorr signature defined as (e, s), where $e = H(sG - eP, m)$. We only focus on the first, and more popular variant that supports batch verification and is part of the Bitcoin improvement proposal (BIP) 340 [218]. The **EdDSA** signature is a type of Schnorr signature optimized for twisted Edwards curves [20].

ECDSA. ECDSA signature is the pair (r, s), where $r = x_{kG} \pmod{n}$ for a random $k \in [1, n]$ and $s = k^{-1}(H(m) + rd) \pmod{n}$ computed in `Sign`. In `Verify`, one then verifies (r, s) by computing $R = H(m)s^{-1}G + rs^{-1}P$ and checking that $x_R \pmod{n} = r$. ECDSA signatures can be verified in

a similar way as Schnorr signatures, i.e., checking that $R = H(m)s^{-1}G + rs^{-1}P$, assuming we have the point R (and not merely r).

BLS signature. BLS signature [34] is the point $\sigma = dH_E(m)$, where H_E is an appropriate hash-to-curve function, i.e., $H_E(m)$ is a point on E. The verification equation is $e(\sigma, G) = e(H_E(m), P)$. Here, e is a bilinear pairing. Hence, only pairing-friendly curves (with an efficiently computable pairing) can be used in practice. While the signing itself is simple and fast, the computation of the pairing can be a limiting factor, especially for resource-constrained devices.

To save space, protocols often store and transmit points (public keys or generators) in compressed form. For any point P, the x_P coordinate determines the y_P coordinate up to a sign through the curve equality $y_P^2 = x_P^3 + ax_P + b$. Hence, the point P can be represented using x_P and a single bit signaling the sign of y_P. The X9.62 standard defines the compressed format specifically as $PC|x_P$, where PC is a single byte with $PC = 2$ if the least significant bit of y_P is 0 and $PC = 3$ otherwise. The uncompressed form is represented as $4|x_P|y_P$, with 4 represented by a single byte and the coordinates zero-padded to the field length in bytes.

2.4 Point validation

Both parties in the ECDH protocol take as an input the public key P of the other party and perform a scalar multiplication dP using their private key d. To protect d, it is crucial to properly validate the point P. According to several major ECC standards (SECG SEC1 [181], NIST SP 800-56 [151, 152], ANSI X9.62 [5], IEEE P1363 [99]), the validation is composed of three steps:

P is affine. While every affine point can be represented using the x, y coordinates, for the infinity point $\mathcal{O}$ one needs projective coordinates $(X : Y : Z)$ in which $\mathcal{O}$ lies in the projective plane given by $Z = 0$. The precise representation depends on the coordinate system (e.g., $\mathcal{O} = (0 : 1 : 0)$ for the standard projective and $\mathcal{O} = (1 : 1 : 0)$ for the Jacobian coordinates). The standards SECG SEC1, NIST SP 800-56 and ANSI X9.62 define the zero byte string as $\mathcal{O}$ and any affine point has the first byte nonzero (indicating the potential compression). Thus, checking $P \neq \mathcal{O}$ is trivial.

P lies on the curve. The public key point contains the affine coordinates x, y in either compressed or uncompressed form. The implementation

should check that x, y represent elements of the finite field $\mathbb{F}_p$ (i.e., check that $x, y \in [0, p)$). Any pair (x, y) that then satisfies the curve equation $y^2 = x^3 + ax + b$ is an affine point on the curve. Failure to test this can lead to the invalid or twist curve attack (see Section 2.4.2).

P is in the correct subgroup. The point P should lie in the subgroup of order n. Otherwise the small subgroup attack is applicable for ECDH and a number of bits of the private key can be recovered depending on the size of the cofactor h. There are four common ways how implementations deal with this after receiving an affine point P:

- Compute $R = nP$ and reject the point if $R \neq \mathcal{O}$. This is recommended by several standards including NIST FIPS, SECG, X9.62, and P1363.
- Compute $R = (h^{-1} \pmod n)hP$ and proceed the protocol with R. This computation clears out the cofactor part of the point, i.e., if $P = G + P_h$, then $(h^{-1} \pmod n)h(G + P_h) = G + (h^{-1} \pmod n)\mathcal{O} = G$. Depending on the protocol, the details of this might differ. For instance, in the computation of the shared secret kP in the Diffie-Hellman protocol, we can replace k with $((h^{-1} \pmod n)hk)$. The costly inverse h^{-1} can be precomputed.
- Use a modified version of the protocol which deals with the cofactor as part of the protocol. This is the approach of the Cofactor Diffie-Hellman protocol. The shared secret is khP instead of kP. Similarly, the popular X25519 protocol assumes that the private key is a multiple of the cofactor $h = 8$, i.e., $k = 8k_0$ for some k_0.
- Ignore the subgroup check. This is not necessarily a sign of negligence. All of the above validation options are expensive (i.e., as expensive as the ECDH secret derivation). Moreover, the most popular NIST curves have cofactor $h = 1$. Finally, the small subgroup attack (as described in Section 2.4.1) that is possible because of the lack of validation can only recover a few bits of the private key. Note, however, that other variants of the attack [120] aim for key reuse rather than key recovery.

2.4.1 Small subgroup attacks

The small subgroup attacks target the ECDH protocol that works over a curve with a nontrivial cofactor $h \neq 1$. The attacker sends to the victim B their public key P_A as part of the protocol. The attacker assumes that the victim trusts that P_A lies in the subgroup of order n

and does not perform any check. If the point P_A has order h, and not n, then $d_B P_A \in \{P_A, \ldots, hP_A\}$, where $d_B P_A$ is computed by B with their private key d_B. As a result, the shared secret $\mathrm{KDF}(d_B P_A)$ can be then enumerated (and checked with the actual shared secret computed by victim) by the attacker which leaks $d_B \pmod h$ to them [120].

Contrary to the statement on the SafeCurves website [29, 30], it is not enough to reject points P_A with $hP_A = \mathcal{O}$, though it would be faster than computing and rejecting points with $nP_A \neq \mathcal{O}$. The attacker can send a mixed order point $P_A = G + P_h$, where P_h is a point of order h. The party B then computes $d_B P_A = d_B G + d_B P_h = P_B + d_B P_h$. The space of possible shared secrets $\{\mathrm{KDF}(P_B + P_h), \ldots, \mathrm{KDF}(P_B + hP_h)\}$ can be easily enumerated, which, again, leaks $d_B \pmod h$.

2.4.2 Invalid curve attacks

The invalid curve attack targets the ECDH protocol with the assumption that the victim implementation does not check that the point lies on the curve. The attack is then based on the observation that addition and doubling formulas often do not contain the Weierstrass parameter b (the addition formulas usually do not contain the parameter a either). This means that the multiplier using these formulas correctly computes kP for a point P on any curve with the same parameter a.

The victim expects a public key point on a curve $E : y^2 = x^3 + ax + b$. The attacker generates and sends a point P_A on a different curve $E' : y^2 = x^3 + ax + b'$, with a small subgroup of order h. The victim computes the shared secret on the curve E'. Similarly as in the small subgroup attack, since h is small, the possible values for the derived secret computed by the victim can be then enumerated. This gives the attacker $d_B \pmod h$. The attacker repeats this for several curves obtaining $d_B \pmod{h_i}$ for each h_i, until the whole private key d_B is recovered using the Chinese remainder theorem [31, 7].

The degenerate curve attack [147] also tricks the victim to perform the ECDH protocol on a different group. The difference from the invalid curve is that this group is not an elliptic curve group, rather the multiplicative group of the underlying base field.

2.5 ZVP attack

This section presents one special type of side-channel attack called zero-value point (ZVP) [1] Attack and its variants. The ZVP attack is one of the central primitives that will repeat throughout the thesis. We will make both practical and theoretical improvements on the attack in Chapter 4, Chapter 5 and use it for reverse engineering in Chapter 6.

The target of the ZVP attack [1] is a static private key d used in a ECDH implementation. The protocol takes a point P (public key) on input, which is multiplied by the secret key d, resulting in dP. The special assumption of the ZVP attack is that the attacker has a side-channel oracle (given by, e.g., power consumption) that tells them whether a zero intermediate value has occurred during the computation of the scalar multiplication. For example, during the computation of the add formula 2.1, it can be detected that the intermediate value $A = 0$. This is a reasonable assumption as the power consumption during the zero-value operations dramatically decreases. See also Figure 2.2.

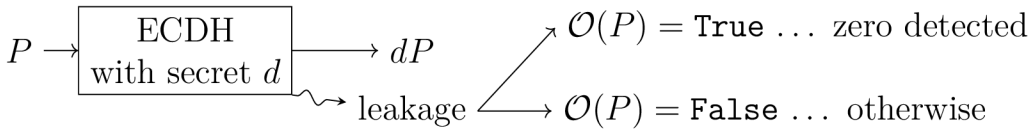


Figure 2.2: Construction of the ZVP oracle $\mathcal{O}$.

We will explain the attack in detail on the double-and-add multiplier (Algorithm 1). The attacker gradually recovers the prefixes $d^{\geq i}$ of d, digit by digit, starting from an empty prefix. Assuming that the attacker recovered the prefix $d^{\geq i}$, they take a guess on the next upper digit $d^{(i-1)}$. The idea of the ZVP attack is to verify this guess by constructing P such that a zero value appears during the computation of add. If the guess on $d^{(i-1)}$ is correct then $\mathcal{O}(P) = \texttt{True}$.

It remains to explain how to construct a point P that invokes the zero value. In a formula for add, any intermediate value can be expressed as a polynomial f in the coordinates of the input points Q and P (or $-P$) to the add (this can be seen in the example formula 2.1). The goal is, therefore, to find P such that the coordinates of $\pm P$ and Q are the roots of f, where Q is a known multiple of P. This problem is an instance of the so-called *dependent coordinates problem* (DCP):

Definition 1. *Let $E/\mathbb{F}_p : y^2 = x^3 + ax + b$ be a curve. Given a polynomial $f \in \mathbb{F}_p[x_1, x_2, y_1, y_2]$ and scalars $k, g \in \mathbb{Z}$, find a point $P \in E(\mathbb{F}_p)$ such that $f(x_Q, x_G, y_Q, y_G) = 0$, where $Q = kP$ and $G = gP$. We denote this problem[3] as $DCP_f(k, g)$ with the special instance $DCP_f(k, 1)$ as $DCP_f(k)$.*

To summarize the ZVP attack, the attacker makes a guess g on the digit $d^{(i-1)}$ of a private key d, with already recovered prefix $k = d^{\geq i}$, selects an intermediate polynomial f that appears during the add computation and solves $\mathrm{DCP}_f(k, g)$. The solution is a point P for which $\mathcal{O}(P) = \texttt{True}$ if the guessed k is correct, i.e., if $g = d^{(i-1)}$.

Remark 1. One might jump to the conclusion that if $\mathcal{O}(P) = \texttt{True}$, then the guess g must be correct. As a counterexample, if $f = x_1 + x_2$ and P is a solution to $\mathrm{DCP}_f(k, g)$ then $x_{gP} + x_{kP} = 0$. However, $x_{-gP} + x_{kP} = 0$ and so P would invoke a zero if g is correct as well as if $-g$ is correct. For such cases, we have to keep track of all the possibilities for g as we progress through the digits of d.

The ZVP attack has two issues, though. Firstly, solving DCP effectively is, in general, an open problem. Instances of DCP with smaller values of k (shorter prefixes $d^{\geq i}$) can be easily solved, but that allows the attacker to target only a few digits of d. Secondly, DCP does not have to have a solution, regardless of whether our guess is correct or not. We, therefore, cannot simply iteratively reveal each digit of d, as some digits might correspond to cases for which the DCP does not have a solution.

There are two attacks that can be considered special cases of the ZVP attack called refined power analysis (RPA) attack [87] and exceptional procedure attack (EPA) [102]. The RPA attack forces a coordinate of a point to be zero during the computation of the scalar multiplication. The DCP in this case is $\mathrm{DCP}_f(k)$ for either $f = y_1$ or $f = x_1$. The RPA attack is independent of the choice of the formula for add or dbl and relies purely on the existence of points with zero coordinates. All points with zero coordinates in the usual curve models have low order (≤ 4) and should not appear during the scalar multiplication — with the exception of points $(0, y)$ in the Weierstrass model [184]. The EPA attack takes a similar approach as RPA but also assumes that an incorrect computation

[3] The original formulation of DCP in [184] is with $g = 1$ for the simple binary representation. This is a generalization for more SM algorithms and scalar representations.

of add can be forced. This computation is induced using special points for which the used formula for add is not defined.

All ZVP attacks can be thwarted using scalar randomization, as the attacker cannot solve the DCP for an unknown randomized scalar and cannot prepare the input point. However, the RPA attack is resistant to projective coordinate randomization as $(0 : y : z) = (0 : ry : rz)$ for any $r \in \mathbb{F}_p$. ZVP and EPA attacks are resistant to coordinate randomization on the condition that the roots of the DCP polynomial f do not change under coordinate randomization.

2.5.1 Solving DCP

For a complete introduction of ZVP, it remains to explain how to solve DCPs that appear during the attack. Given a $\mathrm{DCP}_f(k, g)$ problem, one can replace the polynomial $f \in \mathbb{F}_p[x_1, x_2, y_1, y_2]$ with $f' \in \mathbb{F}_p[x_1, x_2]$, by removing any occurrences of y_1, y_2 using the curve equation [184]. We will, therefore, assume that already $f \in \mathbb{F}_p[x_1, x_2]$. In general, the only known approach [184, 1] for solving the $\mathrm{DCP}_f(k, g)$ is to express the maps for multiplication by k and g using rational functions $u_k, u_g \in \mathbb{F}_p(x)$ and substitute into f:

$$f(u_k(x), u_g(x)) = 0. \tag{2.2}$$

Taking just the numerator $\overline{f}$ of the rational function on the left, we get a univariate polynomial equation $\overline{f}(x) = 0$. We then test whether at least one root of $\overline{f}$ lifts to a point on our curve (i.e., there is a point with the root as x-coordinate).

Remark 2. All complexities are stated in an indeterminate $M(m)$ for the complexity of multiplying two degree-m polynomials. The complexity of root-finding of a degree-m polynomial is $O(M(m) \log m \log mp)$ operations in $\mathbb{F}_p$ [210].

Regarding the complexity of the just described method, the main bottleneck is the construction and root finding of $\overline{f}$, i.e., the numerator of (2.2). The rational function u_k can be computed using division polynomials in $O(M(k^2) \log k)$ time/space and satisfies $\deg(u_k) = k^2$ [213]. Similarly, $\deg(u_g) = g^2$ and so $\deg(\overline{f}) \leq 2 \max(k, g)^2 \deg f$. Taking

the Karatsuba complexity $m^{1.6}$ for $M(m)$, the overall time and space complexity is:

$$O(M(\delta^2)\log\delta\log\delta^2 p) \subseteq O(\delta^{3.2}\log\delta\log\delta^2 p), \quad \delta = 2\max(k,g)^2\deg f. \tag{2.3}$$

Due to this complexity in both time and space, we can solve the general DCP only for small scalars k and g. In the context of the ZVP attack, the scalar g is small, but the scalar k is a prefix of a private key, i.e., $k = d^{\geq i}$. Hence, the ZVP attack can recover only a few upper digits of the private key d.

Nevertheless, some instances of DCP have been solved in the literature for large k [1, 140, 184]. The reason is that these instances were defined over a special type of polynomials f. Specifically, they targeted $\mathrm{DCP}_f(k)$ in which the polynomial f depends on the coordinates of only one input point. To explain why such polynomials make the problem easy, consider $f = x_1 + 1$ as an example. We can simply first find a point Q that satisfies $x_Q + 1 = 0$ and then compute $P = k^{-1}Q$. As our analysis in Chapter 4 shows, most addition formulas do not contain such polynomials, and the attacker must use the general (hard) DCP.

2.6 Fast endomorphisms

Gallant et al. [82] introduced a technique to accelerate scalar multiplication using *fast endomorphisms* — the GLV method. An endomorphism $\varphi : E \to E$ is a group homomorphism defined by rational maps r_1, r_2: $(x, y) \mapsto (r_1(x, y), r_2(x, y))$. Coefficients of r_1, r_2 lie in $\mathbb{F}_p$, and r_1 can be expressed purely in the x-coordinate (in the Weierstrass model). Any such endomorphism naturally acts as a scalar multiplication by $\lambda \in \mathbb{Z}$ on the (cyclic) subgroup of prime order n. Any scalar $k \in \mathbb{Z}$ can be then decomposed using the extended Euclidean algorithm as

$$k = k_0 + k_1\lambda \pmod{n}, \quad k_0, k_1 \approx \sqrt{n}.$$

The benefit of this decomposition is the size of the scalars k_0, k_1 when compared to the expected size of k. Generally, the number of iterations in multipliers is proportional to the size of the used scalar. A random scalar has the expected size $\ell(k) \approx n$, while the decomposed scalars have roughly half the size, i.e., $\ell(k_0) \approx \ell(k_1) \approx \ell(n)/2$. However, now there are two scalars, and so the number of iterations is halved only if the

two multiplications $k_0P + k_1\varphi(P)$ are carried out simultaneously using a special multiscalar multiplier (MSM). The two main types of MSM either use the interleaving method (as in the libsecp256k1 library [217]) or the Straus-Shamir [18] trick. The difference between the two methods is that at the beginning, the Straus-Shamir trick precomputes small linear combinations of points $P, \varphi(P)$, while the interleaving precomputes small multiples of the points P and $\varphi(P)$ separately. Details differ depending on the representation of the scalars k_0, k_1.

Algorithm 2 is the interleaving method combined with the regular w-NAF representation as designed in [109] and used in libsecp256k1 for $w = 5$ (also proposed by Brumley [44] for OpenSSL). In the regular w-NAF representation, the digits of the scalar take the odd values from $\{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$. Since none of the digits of the private key are ever zero in this representation, the number of add and dbl operations is fixed, providing a SCA protection (e.g., against cache timing attacks). Note that in each iteration of Algorithm 2, computing $Q + d_0^{(i)}P + d_1^{(i)}\varphi(P)$ requires two additions. There are two natural options to order them, either $(Q + d_0^{(i)}P) + d_1^{(i)}\varphi(P)$ or $Q + (d_0^{(i)}P + d_1^{(i)}\varphi(P))$. The second option requires temporary space to store $d_0^{(i)}P + d_1^{(i)}\varphi(P)$, but it can use faster formulas for addition since both the precomputed points $d_0^{(i)}P, d_1^{(i)}P$ can be in the affine form. For instance, this approach with the modified `add-2009-bl` addition formula can save one finite field squaring compared to the first option [28]. The first variant is used in libsecp256k1, but we will consider both variants of the interleaving algorithm, as their distinction will be relevant in the following chapters.

Algorithm 2: Interleaving Multiscalar Multiplication.

Input: $P \in E$, endomorphism φ, l-digit w-NAF scalars d_0, d_1
Output: $d_0 + d_1\varphi(P)$

Store kP, $k\varphi(P)$ for $k \in \{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$
$Q \leftarrow \mathcal{O}$
for $i = l - 1$ down to 0 **do**
  for $j = 0$ *to* $w - 1$ **do**
    $Q \leftarrow \mathsf{dbl}(Q)$
  $Q \leftarrow \mathsf{add}(Q, d_0^{(i)}P)$ `/* Alternatively:` $R \leftarrow \mathsf{add}(d_0^{(i)}P, d_1^{(i)}\varphi(P))$ `*/`
  $Q \leftarrow \mathsf{add}(Q, d_1^{(i)}\varphi(P))$ `/*` $Q \leftarrow \mathsf{add}(Q, R)$ `*/`
return Q

Algorithm 3 is the Straus-Shamir trick for a general representation of the scalars d_0, d_1. We will analyze the algorithm for two types of representations in this work. First, the GLV-SAC was designed by Faz-Hernández et al. [74] to achieve a constant number of dbl and add operations as a protection against side-channel attacks. The GLV-SAC representation of two scalars d_0, d_1 is a signed representation satisfying $(d_0^{(i)}, d_1^{(i)}) \in \{(1,0), (-1,0), (1,1), (-1,-1)\}$. In particular, $(d_0^{(i)}, d_1^{(i)}) \neq (0,0)$, which means that add is computed in each iteration. We will study the resistance of the GLV method with GLV-SAC to side-channel attacks in Chapter 4.

Algorithm 3: GLV method with Straus-Shamir trick.

Input: $P \in E$, endomorphism φ, l-digit scalars d_0, d_1
Output: $d_0 + d_1\varphi(P)$

Precompute $d_0^{(i)}P + d_1^{(i)}\varphi(P)$ for all possible values of $d_0^{(i)}, d_1^{(i)}$
$Q \leftarrow \mathcal{O}$
for $i = l - 1$ *down to* 0 **do**
    $Q \leftarrow \mathsf{dbl}(Q)$
    $Q \leftarrow \mathsf{add}(Q, d_0^{(i)}P + d_1^{(i)}\varphi(P))$
return Q

In Chapter 5 we will focus on the speed of the GLV method (rather than SCA protection) using the Joint Sparse Form (JSF) [197, 167]. In general, JSF is a joint representation of two or more scalars that represents each pair of bits (one bit for each scalar) using $0, 1$ or -1 such that the number of non-zero pairs (i.e., the joint Hamming weight) is minimal. Since the joint Hamming weight then equals the number of additions during the Straus-Shamir trick, the number of additions is also minimal. To be precise, the size of the JSF form is larger than the binary form, but only by at most one (signed) bit and so the number of additions only increases by at most one due to this property. The expected joint Hamming weight of two s-bit scalars in JSF is $0.5s$, whereas, in simple binary, it would be $0.75s$. We will also consider the JSF of three and four scalars for which the expected weight is $0.59s$ and $0.64s$, respectively. In the case of GLV, the two $\ell(n)/2$-bit scalars k_0, k_1 have joint Hamming weight $\ell(n)/4$ resulting in $\ell(n)/2$ doublings and $\ell(n)/4$ additions. For comparison, the simple double-and-add with the non-adjacent form (NAF)

representation performs $\ell(n)$ doublings and $\ell(n)/3$ additions, i.e., twice as many doublings and $\ell(n)/12$ more additions.

Naturally, the GLV method makes sense only if $\varphi(P)$ can be computed much more efficiently than the scalar multiplication by λ. Elliptic curves in the form $y^2 = x^3 + b$ (i.e., $a = 0$), where $p = 1 \pmod 3$, are equipped with an endomorphism which can be computed using only one multiplication in $\mathbb{F}_p$: $(x, y) \mapsto (\beta x, y)$, where $\beta \in \mathbb{F}_p^\times$. The corresponding scalar λ satisfies the quadratic equation $\lambda^2 + \lambda + 1 = 0 \pmod n$. Popular examples of such curves is the secp256k1 curve used in Bitcoin and several of the popular pairing-friendly curves like BLS12-381 [37].

2.6.1 Scalar decomposition

The final part of this section explains how exactly can we compute the decomposition $k = k_0 + k_1\lambda \pmod n$ in the GLV method. The usual use case of GLV curves (e.g., ECDH protocol) assumes that decomposed scalar $k = k_0 + k_1\lambda \pmod n$ is not fixed and so the scalar decomposition needs to be done efficiently on the fly. The common approach is to consider the lattice

$$\mathcal{L} = \langle (z_0, z_1) \mid z_0 + z_1\lambda = 0 \pmod n \rangle$$

and find a close vector $v \in \mathcal{L}$ to the vector $(k, 0)$. The decomposition (k_0, k_1) is the short vector $(k, 0) - v$. Assuming we have a short basis of $\mathcal{L}$, with vector norms $\approx n^{1/2}$, Babai's technique finds v resulting in $k_0, k_1 \approx n^{1/2}$ [11].

The bottleneck is finding the short basis for $\mathcal{L}$, but this can be done once for the curve using the following early abort version of the Extended Euclidean Algorithm (EEA): Recall that for any two integers, (n, λ) in our case, EEA produces a sequence of equations

$$s_i n + t_i \lambda = r_i, \quad \text{where } \; s_0 = t_1 = 1, \; s_1 = t_0 = 0, \; r_0 = n, \; r_1 = \lambda.$$

The sequence r_i is strictly decreasing, starting from n until it reaches the greatest common divisor of λ and n. The sequence $|t_i|$ is strictly increasing, starting from 0. Heuristically, the sequences are expected to meet in the middle when $r_i \approx |t_i| \approx n^{1/2}$, in which case the algorithm is aborted, and two shortest vectors $(r_i, -t_i)$ are taken as the basis of $\mathcal{L}$. This does not work in general. For example, if $\lambda < n^{1/2}$, then no short

basis is found since the algorithm terminates at $i = 1$ with $(\lambda, -1)$, $(n, 0)$ as the shortest two vectors.

Since one endomorphism enables us to decompose k into two smaller scalars, one would like to use more endomorphisms for larger decompositions. This is the idea of the GLS method [80], which uses curves over extension fields that are equipped with two fast endomorphisms φ and ψ:

$$k = k_0 + k_1\lambda_\varphi + k_2\lambda_\psi + k_3\lambda_\varphi\lambda_\psi \pmod{n}, \quad k_i \approx \sqrt[4]{n}.$$

Similarly to the 2-dimensional case, the decomposition (k_0, k_1, k_2, k_3) is the vector $(k, 0, 0, 0) - v$, where v is the closest vector to $(k, 0, 0, 0)$ in the lattice

$$\mathcal{L} = \langle (z_0, z_1, z_2, z_3) \mid z_0 + z_1\lambda_\varphi + z_2\lambda_\psi + z_3\lambda_\varphi\lambda_\psi = 0 \pmod{n} \rangle.$$

The part that does not generalize easily is the search for the short basis of $\mathcal{L}$. Usually, the short basis is found by applying some reduction algorithm (e.g., LLL [119]) to the (long) basis:

$$(\lambda_\varphi, -1, 0, 0), (\lambda_\psi, 0, -1, 0), (\lambda_\varphi\lambda_\psi, 0, 0, -1), (n, 0, 0, 0).$$

The short basis can be precomputed again once for the given curve, so this does not have any effect on the performance. The main limitation of the GLS method is that it is only supported by curves over extension fields. This work focuses on prime field curves that are used in cryptographic protocols and so the GLS method is out of scope. However, we will use the idea of a large decomposition in Chapter 5.

3 Dissecting elliptic curve standards

This chapter is based on the following publication:

[VS2] V. Sedlacek, V. Suchanek, A. Dufka, M. Sys, and V. Matyas. "DiSSECT: Distinguisher of Standard and Simulated Elliptic Curves via Traits". In: *Progress in Cryptology – AFRICACRYPT 2022*. Ed. by L. Batina and J. Daemen. Springer Nature Switzerland, Cham, pp. 493–517. ISBN: 978-3-031-17433-9

3.1 Introduction

Many prominent cryptographers [177, 121, 26, 62, 179] criticize the selection of curves in all major elliptic curve standards [5, 182, 108, 191]. The lack of explanation of parameters used in cryptographic standards provides a potential space for weaknesses or inserted vulnerabilities.

This is a serious matter, considering that the popular NIST curves have been designed by the NSA, which has been involved in a number of incidents: the Dual EC DRBG standard manipulation [91, 23, 51], Clipper chip backdoor [204], or the deliberate weakening of the A5 cipher [32]. Bernstein et al. [26] justify such skepticism, showing that the degrees of freedom in the generation of elliptic curves offer a means to insert a backdoor. Relying on the supposed implausibility [116] of weak curves escaping detection for such a long time may prove catastrophic. As the usage of NIST curves increases with time [207], the impact of any vulnerability would be extensive at this moment.

Even though newer, more rigidly generated curves like Curve25519 [24], Ed448-Goldilocks [94] or NUMS curves [33] are on the rise, Lochter et al. [121] argue that *"perfect rigidity, i.e., defining a process that is accepted as completely transparent and traceable by everyone, seems to be impossible."* Thus, a thorough wide-scale analysis of the standard curves is important to establish trust in elliptic curve cryptosystems.

There is no clear way of looking for unknown vulnerabilities, especially within elliptic curve cryptosystems with such a rich and complex theory behind them. Our main idea is that if a hidden weakness is present in a curve, it can manifest itself via a statistical deviation when we compare the curve to a large number of generic curves. We consider two cases: either the deviation stems from the generation method of the curve, or from the choice of its parameters. Consequently, we compare the curve to two types of curves — random curves in the former case and curves generated according to the same standard in the latter case.

Contributions:

- We assemble the first public database of all standard[1] curves (to the best of our knowledge), with comprehensive parameter details.
- We develop an open-source, extensible framework DiSSECT[2] for generating curves according to known standards, and for statistical analysis of the standards. This includes a large number of test functions and visualization tools.
- We find properties of the standard GOST curves [164, 196] that are inconsistent with the claimed [2] generation method and present unreported properties of the BLS12-381 [15] curve.

In addition, we offer a new methodology for testing properties and potential weaknesses of standard curve parameters and their generation methods. By a systematic analysis of standard curves, we rule out multiple types of problems, raising the level of trust in most of the curves.

This chapter is organized as follows: In Section 3.2, we briefly survey the major elliptic curve standards. Section 3.3 introduces our database and explains our methodology for simulating and distinguishing curves. Section 3.4 gives an overview of our proposed trait functions and presents the findings of the outlier detection. We report on the technicalities of our tool in Section 3.5 and draw conclusions in Section 3.6. Appendix B.1 contains descriptions of individual traits.

[1] In this work, we use the adjective "standard" for widely used curves (from actual standards and even that are/were a de facto standard).

[2] Available at `https://dissect.crocs.fi.muni.cz`.

3.2 Overview of standard curve generation

The main approach for finding a curve that satisfies all known security conditions (see Section 2.1), including a large CM discriminant, is repeatedly selecting curve parameters until the conditions hold. The bottleneck here is computing the group size $\#E(\mathbb{F}_p)$. We can use the polynomial SEA algorithm [178], but in practice, it is not fast enough to allow on-the-fly ECC parameter generation.

Several standardization organizations have therefore proposed elliptic curve parameters for public use. To choose a curve, one has to first pick the appropriate base field. Several standards also choose primes of special form for more efficient arithmetic, e.g., generalized Mersenne primes [198] or Montgomery-friendly primes [62]. When the field and the curve form are fixed, it remains to pick two[3] coefficients, one of which is often fixed.

In this subsection, we survey the origin of all the standard curves. For easier orientation, we divide the generation methods of standard curves into three rough categories. Curves of unknown or ambiguous origin are hard to trust, while verifiably pseudorandom ones make use of one-way functions in hopes of addressing this. Other rigid methods go even further in attempts to increase their transparency. Note that the following survey focuses on standards for classical ECC and leaves out pairing friendly curve[4].

Unknown or ambiguous origin. The following is a list of curves we analyze and whose method of generation is either completely unexplained or ambiguous.

- The 256-bit curve `FRP256v1` has been recommended in 2011 by the French National Cybersecurity Agency (ANSSI) in the Official Journal of the French Republic [108]. The document does not specify any method of generation.
- The Chinese SM2 standard [191] was published in 2010 [68] by the Office of State Commercial Cryptography Administration (OSCCA). The standard recommends one 256-bit curve but does not specify the generation method.
- In addition to the Russian GOST R 34.10 standard from 2001, six standardized curves were provided in [164] and [196]. Again, no method

[3] At least for short Weierstrass, Montgomery and twisted Edwards curves.
[4] The website [89] provides a good overview for pairing-friendly curves.

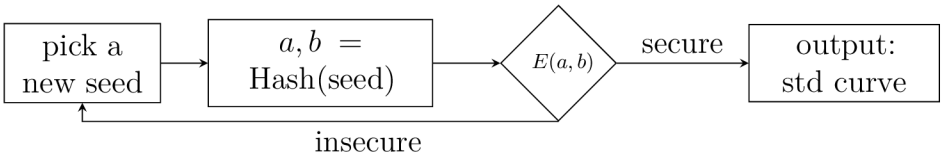


Figure 3.1: A simplified template for generating verifiably pseudorandom curves over a fixed field $\mathbb{F}_p$.

of generation was specified in the original standard, although there were some attempts for explanation afterward [2]. We discuss this in further sections.

- The Wireless Application Protocol Wireless Transport Layer Security Specification [216] recommends 8 curves, 3 of which are over prime fields and are not copied from previous standards. Their method of generation is not described.
- Apart from verifiably pseudorandom curves, the Standards for Efficient Cryptography Group (SECG) [49] recommends so-called Koblitz curves[5], which possess an efficiently computable endomorphism. However, the standard only vaguely states *"The recommended parameters associated with a Koblitz curve were chosen by repeatedly selecting parameters admitting an efficiently computable endomorphism until a prime order curve was found."*

Verifiably pseudorandom curves. Aiming to re-establish the trust in ECC, several standards have picked the coefficients a, b in the Weierstrass form $y^2 = x^3 + ax + b$ in a so-called *verifiably pseudorandom* way, as Figure 3.1 demonstrates.

The details differ and we study them more closely in Section 3.3.2. The common idea is that the seed is made public, which limits the curve designer's freedom to manipulate the curve. However, this might not be sufficient, as Bernstein et al. [26] show that one could still iterate over many seeds (and potentially also over other "natural" choices) until they find a suitable curve. If a large enough proportion of curves (say, one in a million) is vulnerable to an attack known to the designer, but not publicly, this offers an opportunity to insert a backdoor. In particular, we should be suspicious of curves whose seed's origin is unknown.

[5] Analyzing these curves is of great importance due to the usage of `secp256k1` in Bitcoin [142].

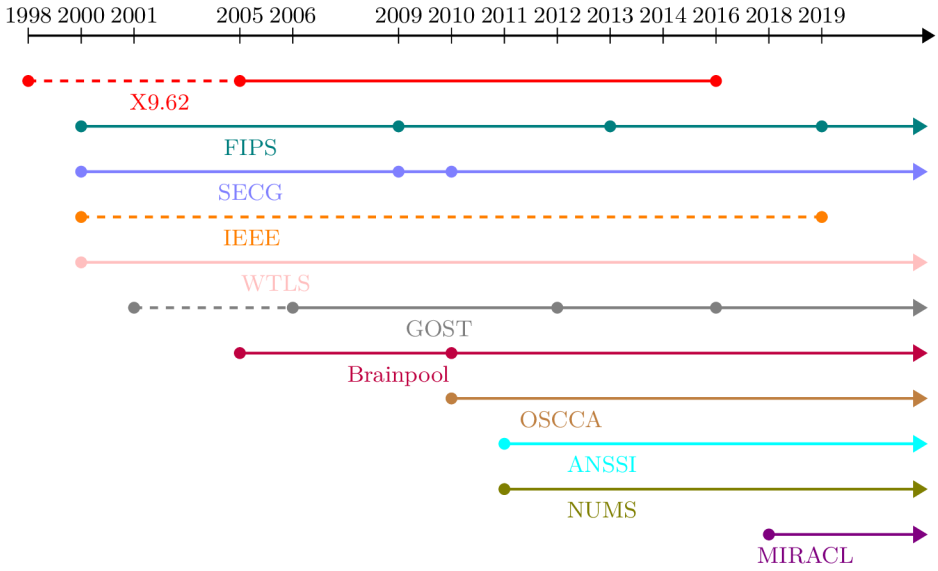


Figure 3.2: Timeline of standardization of publicly available prime-field curves. Dashed line indicates that the source does not specify curve parameters. Individual curves (e.g., Curve25519) are omitted.

In our analysis, we will focus on two major standards containing verifiably pseudorandom generation procedures:

- The Brainpool curves, proposed in 2005 by Manfred Lochter and Johannes Merkle under the title "ECC Brainpool Standard Curves and Curve Generation" [70] to focus on issues that have not been previously addressed such as verifiable choice of seed or usage of prime of nonspecial form.
- NIST in collaboration with NSA presented the first standardization of curves in FIPS 186-2 in 2000 [144]. However, the used generation method already appeared in ANSI X9.62 in 1998 [4]. Other standards recommend these curves (e.g., SECG [49] or WTLS [216]).

Another notable paper considering the verifiably pseudorandom approach is "The Million dollar curve" which proposed a new source of public entropy for the seeds, combining lotteries from several countries [13].

Other rigid methods. As a reaction to the NIST standard curves, an interest in faster curves generated using rigid and verifiable methods has emerged:

- Three Weierstrass curves and three twisted Edwards curves were proposed by Black et al. [33] as an Internet draft called "Elliptic Curve Cryptography (ECC) Nothing Up My Sleeve (NUMS) Curves and Curve Generation" [33].
- MIRACL library [134] combines approaches of NUMS and Brainpool by extracting seeds from well-known constants and iteratively incrementing one of the parameters of a curve in the Weierstrass form.
- Bernstein's Curve25519 and its sibling Ed25519 were created in 2006 [24] and since then have been widely accepted by the cryptography community. The curve is in the Montgomery form and allows extremely fast x-coordinate point operations while meeting the SafeCurves security requirements. In 2013, Bernstein et al. [22] proposed the curve Curve1174 with an encoding for points as strings indistinguishable from uniform random strings (the Elligator map).
- To address higher security levels and maintain good performance, several authors developed curves of sizes in the 400-521 bit range such as the Edwards curve Curve41417 by Bernstein et al. [27], Ed448-Goldilocks by Hamburg [93], E-3363 by Scott [180] or M, E curves by Aranha et al. [8].

3.3 Methodology

Our ultimate goal is to enable assessment of security for standard curves. But since it is not clear how to look for unknown curve vulnerabilities, we aim to pinpoint possible problems via identifying standard curves that deviate from other curves in specific aspects. To find deviations we employ one of the following strategies:

1. **Compare curves from a standard to corresponding simulated ones.** If the standard curves were generated using the defined processes without any hidden conditions, it would be unexpected to statistically distinguish them from our simulated ones, yet we try to do exactly that. We are looking for a standard curve achieving a value that is highly improbable when compared to the simulated curves.
2. **Compare curves from a standard to Random curves.** If the generation algorithm introduces a systematic bias, the first strategy will not allow us to find it — the problem might occur in both the standard

and simulated curves. Thus this strategy tries to detect any sort of unexpected behaviour by comparing curves from a given standard to *Random*[6] curves.

To target specific curve properties, we describe and implement traits, which are functions that take a curve as an input (sometimes with additional parameters) and output numerical results. We run these traits on standard curves as well as simulated ones whenever it is computationally feasible and store the results in our database.

3.3.1 Standard curve database

DiSSECT is, as far as we know, the first public database of all standard elliptic curves, divided into 18 curve categories by their source. The database includes:

1. verifiably pseudorandom curves (X9.62, NIST, SEC, Brainpool);
2. pairing-friendly curves: Barreto-Lynn-Scott curves [15], Barreto-Naehrig curves [161], Miyaji-Nakabayashi-Takano curves [135]);
3. amicable curves: Tweedledee/Tweedledum [38], Pallas/Vesta [97];
4. NUMS curves and curves from the MIRACL library [134];
5. Bernstein's high performance curves [24] and M, E curves [8];
6. curves from the standards ANSSI [108], OSCCA [191], GOST [164, 196], OAKLEY [158], WTLS [216], ISO/IEC [100] and others.

Although our analysis focuses mainly on prime-field curves, the database contains 31 curves over binary fields and one over an extension field. Currently, there are 188 standard curves in total. Note that we also include curves that were but are no longer supported by the standards, and curves that are not recommended for public use, but have been included in the documents for various reasons (e.g., curves for implementation checks). The database provides filtering by bit-length, field type, cofactor size, and curve form.

Additionally, our database contains five categories of simulated curves: $\text{X9.62}_{\text{sim}}$, $\text{Brainpool}_{\text{sim}}$, NUMS_{sim}, $\text{Curve25519}_{\text{sim}}$ and Random.

[6] In the remainder of this work, the word Random with capital R refers to the curves we generated by our own method to be as generic as possible.

Source: number of curves							
X9.62	40	BARP	6	OSCCA	1	$\text{X9.62}_{\text{sim}}$	120k
Brainpool	14	BLS	6	BN	16	NUMS_{sim}	1.2k
NUMS	24	GOST	9	AMIC	4	$\text{Curve25519}_{\text{sim}}$	784
SECG	33	ISO	4	DJB	10	$\text{Brainpool}_{\text{sim}}$	12k
NIST	15	MNT	10	ANSSI	1	Random	250k
MIRACL	8	OAKLEY	2	WTLS	8	other	11

Table 3.1: Numbers of elliptic curves in our database grouped by their source.

For each curve, we also precomputed often occurring properties such as the CM discriminant, the j-invariant, the trace of Frobenius t, and the embedding degree. This precomputation significantly speeds up computation of some traits.

3.3.2 Simulations

We have picked four major standards X9.62 [4], Brainpool [70], NUMS [33] and Curve25519 [24] for simulations, since their generation method was explained in enough detail and can be easily extended for large scale generation. At a few points, the standards were a little ambiguous, so we filled the gaps to reflect the choices made for the actual standard curves whenever possible. We have simulated over 120 000 X9.62 curves, 12 000 Brainpool curves, 1 200 NUMS curves and 750 Curve25519 curves[7].

The aim of this part is not to undertake a thorough analysis of the published algorithms, rather explain our approach to the large-scale generation using the given methods. For the details of the original algorithms, see the individual standards or our website[8]. Although NUMS, Brainpool and Curve25519 provide a method for generating group generators, we are currently not focusing on their analysis.

X9.62 – the standard. We focus on the generation method of the 1998 version [4]. Its input is a 160-bit seed and a large prime p and the output is an elliptic curve in short Weierstrass form over the field $\mathbb{F}_p$, satisfying the following security conditions:

[7] This took up to a week per standard on 40-core cluster of Intel Xeon Gold 5218.
[8] https://dissect.crocs.fi.muni.cz/standards

- "Near-primality": The curve order shall have a prime factor n of size at least $\min\{2^{160}, 4\sqrt{p}\}$. Furthermore, the cofactor shall be s-smooth, where s is a small integer (the standard proposes $s = 255$ as a guide).
- The embedding degree of the curve shall be greater than 20. The standard also specifies that to check this condition, we may simply verify that $p^e \neq 1 \pmod{l}$ for all $e \leq 20$.
- The trace t shall not be equal to 1.

Given a seed and a prime p, the standard computes a $\log(p)$-bit integer r using[9] the function (3.1). The next step is choosing $a, b \in \mathbb{F}_p$ such that $b^2 r = a^3$. This process is repeated until the curve satisfies the security conditions mentioned above.

$$H(\text{seed}) = \underbrace{\texttt{SHA-1}(\text{seed})}_{\text{discard}} \underbrace{||\texttt{SHA-1}(\text{seed}+1)|| \ldots ||\texttt{SHA-1}(\text{seed}+i)}_{\log(p)-\text{bit integer as output}}. \quad (3.1)$$

X9.62 – our approach. For each of the five bit-lengths 128, 160, 192, 224 and 256 bits we have fixed the same prime as the standard (hence all curves of the same bit-length are defined over the same field). Since there is no guidance in the standard how to pick the seed for each iteration, we have taken the published seed for each bit-length and iteratively incremented its value by 1. To pick a and b, we have fixed $a = -3$ (as was done for most X9.62 curves for performance reasons [54]) and computed b accordingly, discarding the curve if $b^2 = -27/r$ does not have a solution for $b \in \mathbb{F}_p$. We also restricted the accepted cofactors to 1, 2, or 4 as this significantly accelerated the point counting. This choice also agrees with the fact that the standard curves all have cofactor 1 and the SECG standard (which overlaps with the X9.62 specifications) recommends the cofactor to be bounded by 4. The point counting — the main bottleneck of the computations — was done by an early-abort version of the SEA algorithm [178]. For each of the five bit-lengths we have tried 5 million seeds, resulting in over 120 000 elliptic curves. Figure 3.3 captures a simplified overview of the algorithm.

Brainpool – the standard. The Brainpool standard proposes algorithms for generating both the prime p and the curve over $\mathbb{F}_p$. Since in our

[9] More precisely, H also changes the most significant bit of the output to 0.

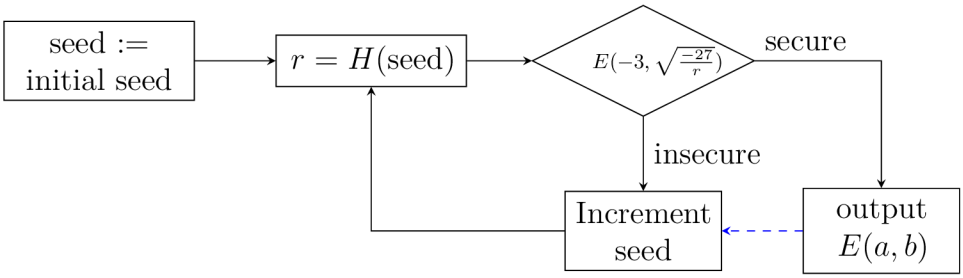


Figure 3.3: X9.62 algorithm adjusted (indicated by the dashed line) for large-scale generation.

simulations we have used the same finite fields as are in the recommended curve parameters we will skip the algorithm for prime generation.

Given a seed and a prime, the curve generation process outputs an elliptic curve in Weierstrass form that satisfies the following security conditions:

- The cofactor shall be 1, i.e., the group order shall be prime.
- The embedding degree shall be greater than $(n-1)/100$.
- The trace t shall not be equal to 1. Technical requirements then state that $t > 1$.
- The class number of the endomorphism algebra of the curve should be larger than 10^7.

The algorithm itself follows similar idea as X9.62, but in more convoluted way, as can be seen in Figure 3.4. This time, the H function (3.1) is used to compute both a and b in pseudorandom way. Roughly speaking, a given seed is repeatedly incremented by 1 and mapped by H until an appropriate a is found and the resulting seed is used for finding b in a similar way. If the curve does not satisfy the condition, the seed is incremented and used again as the initial seed.

Brainpool – our approach. We have used the same approach as in X9.62 and used the published seed as initial seed for the whole generation, incrementing seed after each curve was found. Since generation of both a and b can in theory take an arbitrary number of attempts, the seed is incremented after each failed attempt. We have again used 5 million seeds for each of the four bit-lengths (160, 192, 224, 256) Brainpool recommends. The number of generated curves in total is over 12 000. The

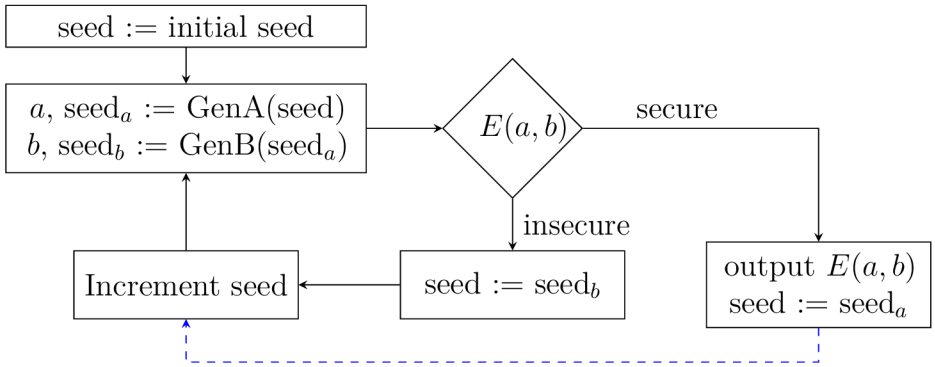


Figure 3.4: Brainpool algorithm adjusted (indicated by the dashed line) for large-scale generation.

drop in the proportion of generated curves compared to X9.62 standard is caused by stricter conditions.

There is currently no known efficient method that computes the class number; instead we checked that a related quantity – the CM discriminant – is greater than 2^{100}, following the SafeCurves recommendations [29].

NUMS – the standard. The NUMS generation method, in accordance with its name, does not fall into the verifiably pseudorandom category. As Brainpool, NUMS proposes algorithms for generating both the prime field and the curve. The lowest recommended bit-length for prime fields by NUMS is 256. The method of prime generation works by starting with $c = 1$ and incrementing this value by 4 until $p = 2^s - c$ is a prime congruent to 3 mod 4.

Although NUMS proposes algorithms for generating both Weierstrass and Edwards curves, we have focused only on the Weierstrass curves. The curve is found by searching for $E(-3, b)$ satisfying the following security conditions by incrementing b, starting from $b = 1$:

- The curve order as well as the order of its twist shall be primes.
- The trace t shall not equal 0 or 1. This condition is further extended by requiring $t > 1$, supposedly for practical reasons.
- The embedding degree shall be greater than $(n - 1)/100$ following the Brainpool standard.
- The CM discriminant shall be greater than 2^{100}, following the SafeCurves recommendations.

NUMS – our approach. We have used the same process, but this time iterating over 10 million values for b, for each of the four bit-lengths 160, 192, 224 and 256. Though the lowest recommended bit-length for prime fields by NUMS is 256, we have generated 160, 192 and 224-bit primes using the proposed method to study curves of lower bit-lengths. This process produced over 1 200 curves, implying that the twist condition is strongly restrictive.

Curve25519 – the standard. Curve25519 was created in a similar rigid manner as NUMS curves by minimizing one coefficient of the curve equation. The prime $2^{255} - 19$ for the prime field was chosen to be close to a power of 2 and with bit-length slightly below multiple of 32 for efficiency of the field arithmetic. Curve25519 is a Montgomery curve of the form $y^2 = x^3 + Ax^2 + x$ with the smallest $A > 2$ such that $A - 2$ is divisible by 4 and:

- The cofactor of the curve and its twist shall be 8 and 4 respectively.
- The trace t shall not equal 0 or 1.
- The embedding degree shall be greater than $(n - 1)/100$.
- The CM discriminant shall be greater than 2^{100}.

Curve25519 – our approach. Similarly to the NUMS approach we have iterated over A for bit-lengths 159, 191 and 255 where we have picked primes $2^{159} - 91$ and $2^{191} - 19$ for lower bitlengths. Iterating over 10 million values for A resulted in roughly 804 curves in total.

Random – our approach. Since all of the simulated standards contain certain unique properties (small b for NUMS, small A for Curve25519, bit overlaps in Brainpool, $rb^2 = -27$ for X9.62) and curves of the same bit-length share a base field, we have also generated curves for bit-lengths 128, 160, 192, 224, 256 using the method depicted in Figure 3.5 that aims to avoid such biases. We call such curves Random, as this term clearly describes our intention. However, for practical reasons, we made the process deterministic using a hash function. For each curve, we generated a prime for the base field by hashing (SHA-512) a seed and taking the smallest prime bigger than this hash when interpreted as an integer. Curve coefficients were chosen using the hash function as well, see Figure 3.5. We kept such a curve if it was not anomalous, had cofactor 1, 2, 4 or 8 and satisfied the SafeCurves criteria on embedding degree and CM discriminant. We tried 5 million seeds, resulting in over 250 000 Random curves.

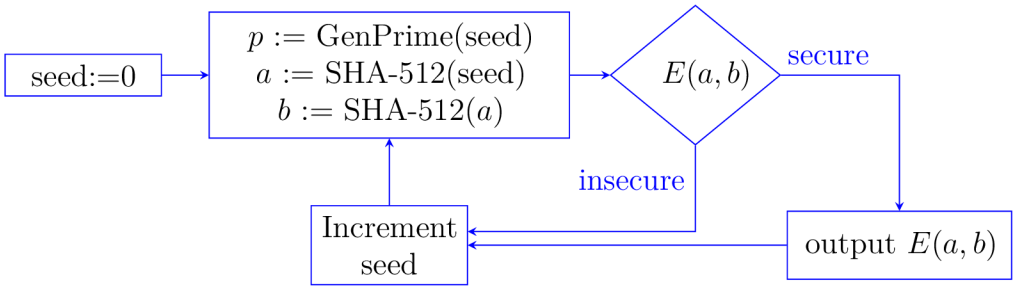


Figure 3.5: Random simulation.

3.3.3 Outlier detection

Our framework offers options for graphical comparisons for described distinguishing strategies. However, manual inspection does not scale well, so we utilize automated approaches to identify suspicious curves. Since we do not have (and cannot have) a labeling that could be used for the typical supervised approaches, we had to resort to unsupervised methods, namely outlier detection.

We built several datasets of simulated curves according to the selected distinguishing strategies. Each dataset of simulated curves consists only of curves of the same bit-length that were generated by the same method. Table 3.2 shows numbers of curves contained within each dataset. The rows correspond to the used generation method and the columns to the bit-length of the selected curves.

	256 bits	224 bits	192 bits	160 bits	128 bits
$\text{X9.62}_{\text{sim}}$	18 502	22 211	18 836	27 780	36 126
$\text{Brainpool}_{\text{sim}}$	1 677	2 361	2 640	3 184	0
NUMS_{sim}	83	109	191	325	0
$\text{Curve25519}_{\text{sim}}$	140	0	366	278	0
Random	18 636	21 226	24 805	29 639	37 311

Table 3.2: Numbers of curves in our datasets of simulated curves.

Additionally, in cases where trait results are (mostly) independent of the curve bit-length, we analyzed curves of all bit-lengths together. This approach allowed us to study even curves that did not match any of the generated bit-lengths. We augmented each dataset with standard curves according to distinguishing strategies, i.e., Random curves with each category of standard curves, and the other three simulated categories

only with standard curves of the same type. From these augmented datasets, we derived feature vectors consisting of all computed trait results, as well as some of their subsets. The features were scaled using a min-max scaler to fit within the $[0; 1]$ range. In case some results were not computed, they were replaced with -1, signifying that the given value took too long to compute (e.g., the factorization of large numbers). Finally, we ran the local outlier factor algorithm [40] to identify outliers within each augmented dataset. If the approach reported a standard curve as an outlier, we inspected such results closer.

3.4 Traits

DiSSECT currently contains 22 trait functions[10] — traits for brevity — designed to test a wide range of elliptic curve properties, including mathematical characteristics of curves, all SafeCurves ECDLP requirements, and properties connected to curve standards or implementations. In particular, the traits are not limited to properties connected to known attacks or vulnerabilities, and aim to find new interesting deviations.

We have divided the traits into five categories:

Potential attacks. These traits test the classical properties of elliptic curves relevant to known attacks (the order and cofactor of the curve; the quadratic twist; the embedding degree). However, we also cover lesser-known and threatening attacks, for example, the factorization of values of the form $kn \pm 1$ as a generalization of [52]. To test scalar multiplication, essential to all ECC protocols, we inspect multiplicative orders of small values modulo the group order n.

Complex multiplication. While at this moment there is no serious attack utilizing any knowledge about the CM discriminant or class number, these values are the defining features of an elliptic curve. They determine the structure of the endomorphism ring, the torsion points, isogeny classes, etc. Multiple traits deal with the factorization of t, the size and the factorization of the discriminant $D = t^2 - 4p$, as well as how D changes as we move the base field to its extensions. We also compute bounds on the class number using the Dirichlet class number formula [66].

Torsion. We have designed several traits to directly analyze the torsion points of a given elliptic curve. One of the traits computes the degree

[10] See https://dissect.crocs.fi.muni.cz/ or Appendix B.1 for a full list.

of the extension $\mathbb{F}_{p^k}/\mathbb{F}_p$ over which the torsion subgroup is (partially) defined. Lower degrees might lead to computable pairings. Another trait approaches torsion from the direction of division polynomials and computes their factorization. We also consider the lift $E(\mathbb{F}_p) \to E(\mathbb{Q})$ and computes the size of the torsion subgroup over $\mathbb{Q}$ which might be relevant for lifting of ECDLP.

Isogenies. Both torsion and complex multiplication can be described using isogenies [194]. Kernel polynomial of every isogeny is a factor of the division polynomial, and the special cases of isogenies, endomorphisms, form an order in an imaginary quadratic field of $\mathbb{Q}(\sqrt{D})$. One of isogeny traits computes the degree of the extension where some/all isogenies of a given degree are defined. Isogeny traits also analyze isogenies of ordinary curves and the corresponding isogeny volcanoes — their depth and the shape of so-called crater, and also the number of neighbors for a given vertex in the isogeny graph.

Standards and implementation. During our analysis of standard methods for generating curves, we have noticed unusual steps in the algorithms, so we designed traits to capture the resulting properties. The nature of the Brainpool standard (see Figure 3.4) causes overlaps in the binary representations of the curve coefficients in roughly half of the generated curves. The NUMS standard, as well as the Koblitz curves, use small curve coefficients by design. The X9.62 standard uses the relation $b^2r = a^3$ to determine the curve coefficients from r. As the last example, the SECG standard recommends the curves `secp224k1` and `secp256k1` with group generators. The x-coordinate of half of both of these generators is the same small number.

Most attacks on the ECDLP are aiming at specific implementation vulnerabilities. To address this, we propose traits that target possible irregularities in practical implementations. One trait follows the idea of [214], where the authors analyze the side-channel leakage caused by improper representation of large integers in memory. Based on [14], we designed a trait that analyzes the number of points with a low Hamming weight on a given curve.

3.4.1 Notable findings

GOST curves. The trait that analyzes the size of the coefficients in the Weierstrass form was motivated by the NUMS standard. However, our

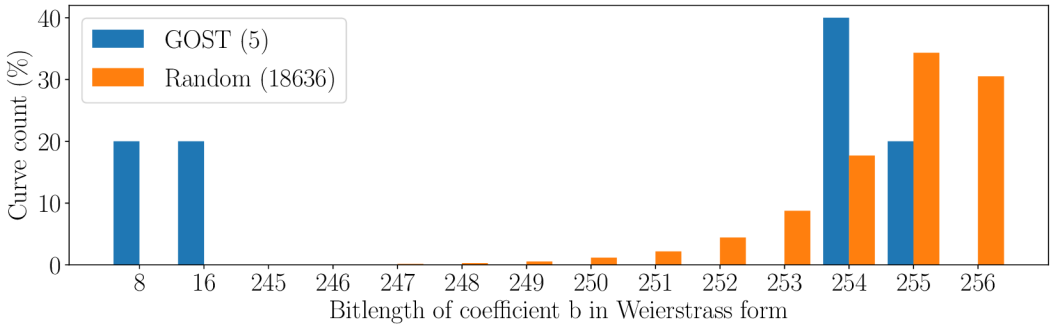


Figure 3.6: Two GOST curves (`CryptoPro-A-ParamSet`, `CryptoPro-C-ParamSet`) exhibit particularly low bit-length of b parameter, even though Alekseev et al. [2] claim that they were generated pseudorandomly.

outlier detection also recognized two 256-bit GOST curves (`CryptoPro-A-ParamSet`, `CryptoPro-C -ParamSet`) in this trait, and a closer inspection in Figure 3.6 revealed that both curves have small b coefficients (166 and 32858). This contradicts Alekseev et al. [2], who claim that all of the seven standardized GOST R curves were generated in the following way[11]:

1. Select p that allows fast arithmetic.
2. Compute r by hashing a random seed with the Streebog hash function.
3. For the generation of twisted Edwards curve $eu^2+v^2 = 1+du^2v^2$, put $e = 1, d = r$. For the generation of Weierstrass curve $y^2 = x^3+ax+b$, put $a = -3$ and b equal to any value such that $rb^2 = a^3$.
4. Check the following security conditions:
   - $n \in (2^{254}, 2^{256}) \cup (2^{508}, 2^{512})$.
   - The embedding degree is at least 32 (resp. 132) if $n \in (2^{254}, 2^{256})$ (resp. if $n \in (2^{508}, 2^{512})$).
   - The curve is not anomalous.
   - The j-invariant is not 0 or 1728.

Thus the small size of b (which should be pseudorandom if r is) implies that it is very unlikely that they were generated with this claimed method. More precisely, since $b^2 = -27/r$ in $\mathbb{F}_p$ and p has 256 bits, then the probability for $b = 166$, resp. $b = 32858$ is 2^{-248}, resp. 2^{-240}, considering

[11] The authors support their claims by providing seeds for two of the seven curves. Note that the seeds were not previously made public.

the bitlengths of the parameters. We hypothesise that the `CryptoPro-A-ParamSet` curve was generated by incrementing b from 1 until the GOST security conditions were satisfied. We have verified that $b = 166$ is the smallest such value with the added condition that the cofactor is 1 (otherwise, the smallest value is $b = 36$). The `CryptoPro-C-ParamSet` does not have this property, as its b coefficient 32858 is only the 80th smallest such value. However, the generator point $(0, \sqrt{32858})$ is clearly of special form as well, with $x = 0$. If we also consider the existence of such point as another condition imposed on the curves, then 32858 is the 46th smallest value. We employed DiSSECT to distinguish this particular curve for from the rest of the 45 curves using the implemented traits in order to explain their choice, but without results.

Furthermore, the trait inspecting CM discriminant revealed that the third curve `CryptoPro-B-ParamSet` from [164] has a CM discriminant of -619. Such a small value is extremely improbable, unless the curve was generated by the CM method [41]. (The CM discriminant -915 of `gost256` is small as well, but this curve was used just as an example and there are no claims about its generation.)

The BLS12-381 curve. The trait that measures $\phi(n)$ divided by the multiplicative order of 2 modulo n (low multiplicative orders translate to high values and vice versa) identified the BLS12-381 curve [37] as an outlier in the set of Random curves. While some traits were expected to show statistical differences between pairing-friendly curves and Random curves, the cause was not clear for this particular trait. To further investigate this, we have adopted the generation method of the BLS12-381 curve to DiSSECT. Figure 3.7 shows the results of the trait on Random curves, the BLS12-381 curve and the BLS simulated curves.

Figure 3.7 shows that the unexpected detection of BLS12-381 as an outlier was not a coincidence. Rather it was caused by the generation method, and consequently the properties, of the BLS12-381 curve. To summarize, the generation method works by finding an integer x satisfying:

- x has low hamming weight,
- x is divisible by 2^{16},
- $n = x^4 - x^2 + 1$ is a prime,
- $p = \frac{1}{3}n(x-1)^2 + x$ is a prime.

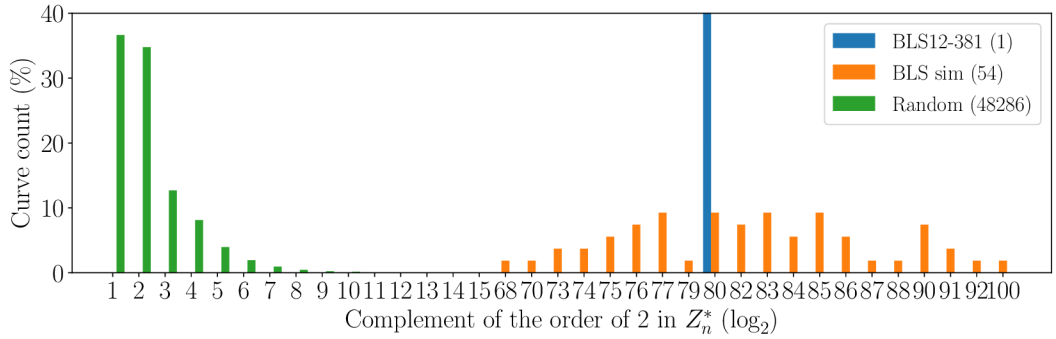


Figure 3.7: BLS12-381 and BLS simulated curves exhibit unexpectedly large complement of the order of 2 in $\mathbb{Z}_n^*$ when compared to Random curves.

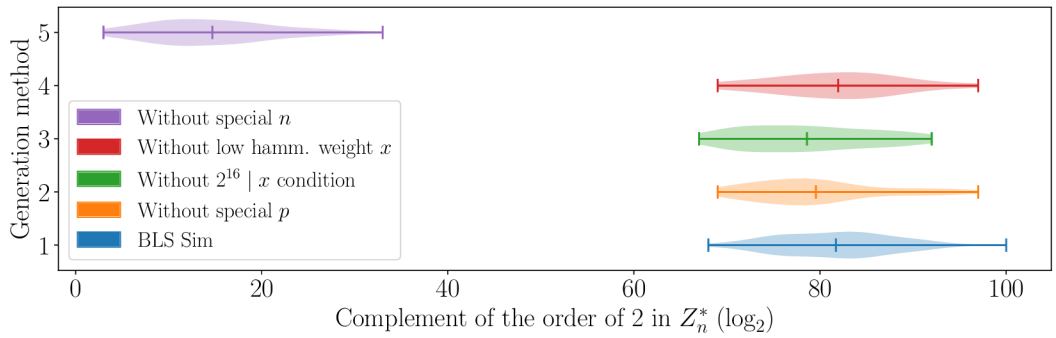


Figure 3.8: Simulation of BLS curves while ignoring different properties have shown that the special construction of n caused the unexpected behavior.

When such x is found, the CM method is used to construct a BLS curve[12] over $\mathbb{F}_p$ with cardinality $\frac{1}{3}n(x-1)^2$. We have identified exactly what conditions in the BLS generation method were behind the outlier detection of BLS12-381 curve by removing each condition and computing the trait for these modified curves. On one hand, Figure 3.8 shows that the conditions on x and special form of p have little or no impact on the results of the trait. On the other hand, we can see that the conditions that were the main cause of the unexpected results of BLS12-381 is the special form of the order $n = x^4 - x^2 + 1$. Indeed, in this case $\phi = x^2(x-1)(x+1)$ which means that small multiplicative orders are more likely than for general n. Nevertheless, this does not seem to cause any vulnerabilities in the BLS12-381 curve.

[12]Iterating through 100 million values of x we have generated 54 BLS curves.

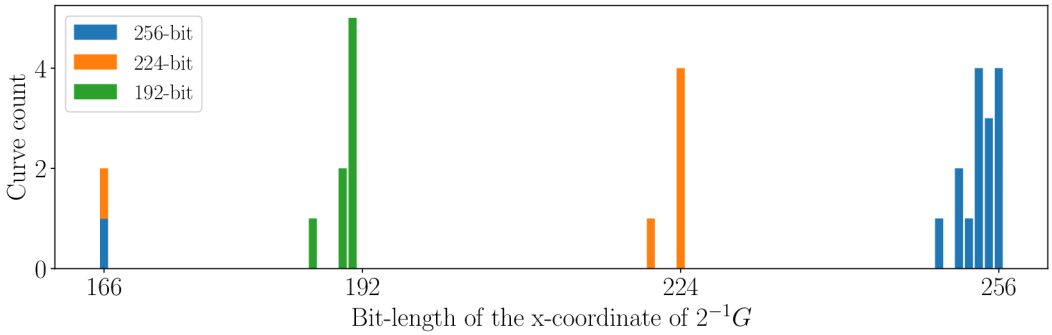


Figure 3.9: Curves secp256k1 and secp224k1 exhibit significantly shorter x-coordinate of the point $2^{-1}G$ than expected. The other standard curves of given bit-lengths are also plotted.

The Bitcoin curve. A trait that inspects the x-coordinate of inverted generator scalar multiples, i.e., x-coordinates of points $k^{-1}G$, where $k \in \{1, \ldots, 8\}$, was inspired by Brengel et al. [39], who reported an unexpectedly low value for $k = 2$ on the Bitcoin curve secp256k1. Furthermore, Maxwell [127] pointed out that secp224k1 yields exactly the same result. Pornin [165] guesses that this was caused by reusing the code for Koblitz curves with the same seeds, together with poor documentation.

We analyzed the results of this trait in our visualization framework (Figure 3.9) and discovered that secp256k1 and secp224k1 are the only standard curves for which the x-coordinate of $k^{-1}G$ is significantly shorter than the full bit-length.

Brainpool overlaps. The trait designed to detect bit-overlaps in the Weierstrass coefficients, revealed a structure in the Brainpool curves, already observed by Bernstein et al. [26].

More precisely the trait compares coefficients a and b when stripped of 160 bits (SHA-1 output size) from the left and right respectively. Inspecting `brainpoolP256r1`, we can see the identical segments in a, b:

$a =$ `0x7d5a0975fc2c3057eef67530417affe7fb8055c126dc5c6ce94a4b44f330b5d9`
$b =$ `0x26dc5c6ce94a4b44f330b5d9bbd77cbf958416295cf7e1ce6bccdc18ff8c07b6`

Such overlaps occur during roughly half of the time during generation; e.g., it happened for `brainpoolP{192,256,384}` but not for `brainpoolP{160,224,320}`. Figure 3.10 illustrates this effect for the Brainpool standard and simulated curves.

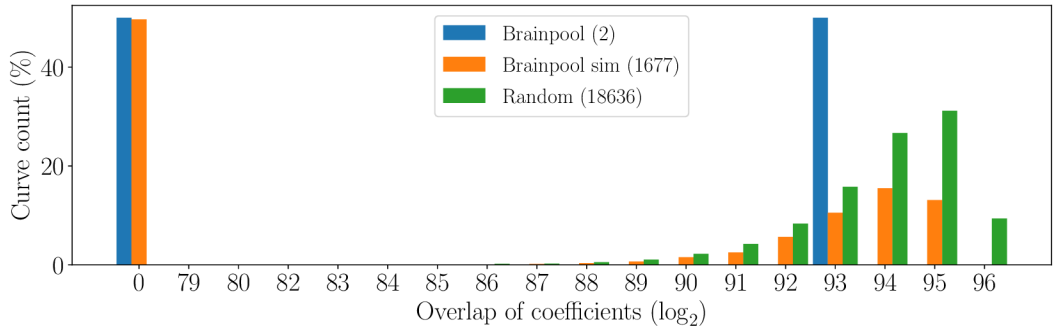


Figure 3.10: Bit-overlap of 256-bit standard Brainpool, simulated Brainpool, and Random curves.

3.5 DiSSECT

DiSSECT is an open-source tool for generating elliptic curves according to our standard simulation methods, computing traits on elliptic curves, and analyzing data using automated approaches and an easy-to-use visualization environment. Although DiSSECT is not intended to perform rigorous statistical tests — largely due to the limited sample size provided by standard curves — it serves as a practical and flexible platform for investigating structural patterns and anomalies. Its code contains implementations of all 22 traits described in Appendix B.1, and it can be easily extended. Adding a new trait requires only a short description of its function and writing a few lines of Sage code that computes the new trait result for a given curve.

To support systematic exploration, the tool incorporates an automated outlier-detection approach that helps highlight unusual behaviors or deviations in the computed datasets. A Jupyter notebook environment can be used to perform a more detailed analysis of computed trait results. Both of these approaches allow configuring the subset of analyzed curves, traits, and their parameters, and can access locally stored data as well as our publicly accessible database.

Our website contains detailed information about curves in our database and corresponding trait results, and trait descriptions with statistics. For a more complex data inspection, the analysis environment configured with access to our database can be launched directly from the website.

3.6 Conclusions

Our framework DiSSECT aspires to survey all standard elliptic curves and to assist in identification of potential problems by comparing them to simulated ones and visualizing the results. We built it as a foundation of elliptic curve cryptanalysis for the cryptographic community and invite more cryptographers and mathematicians to join the project.

Our tool revealed two surprising types of deviations. We realized that the generation process of three GOST curves described by Alekseev et al. [2] is inconsistent with the sizes of the b coefficient in two cases and with the size of the CM discriminant in the third one. Properly documenting such properties is crucial for re-establishing trust in the standard curves and the whole ECC ecosystem. We cannot expect its users, developers, or policy-makers to notice even fairly obvious deviations (e.g., small Weierstrass coefficients), as they often do not access the parameters directly.

Follow up

A subsequent discussion with the authors of [2] revealed that the perceived contradiction was caused by a different understanding of the word "standard". The authors do not consider the RFC document [164] (which specifies the CryptoPro-X-ParamSet curves) as a standard and therefore did not mean to claim anything about the generating method of these curves.

We found an interesting, previously undescribed property of the BLS12-381 curve (related to smoothness) that is caused by its generation. Recent attacks [112] on special properties of pairing-friendly curves have proven that awareness of all properties caused by the pairing-friendly generation methods is crucial. In particular, our approach of isolation of individual properties in BLS generation has shown to have a lot of potential for future research of security of pairing-friendly curves. One further interesting improvement of DiSSECT would be an implementation of clustering algorithms that would help uncover biases systematically introduced by these properties. Another approach might be combining DiSSECT with statistical tools such as [153].

Selected parts of DiSSECT could also be used to quickly assess new individual curves. This might be useful for implementations following the idea of Miele et al. [132], trading standard curves for ephemeral on-the-fly generated ones. Besides cryptographic applications, DiSSECT might also

be useful to number theorists by providing them with empiric distributions of various traits. It is unrealistic to go through the whole space of trait results for different parameter choices and curve sets manually. Thus we employed an automated outlier detection method, which found all discrepancies we discovered manually. Still, there may be other outliers, and we believe it is an interesting open problem to statistically evaluate the results in a way that takes into account the inner structure of the data for a given trait.

4 Side-channel attacks on endomorphisms

This chapter is based on the following publication:

[VS4] V. Suchanek, V. Sedlacek, and M. Sys. "Decompose and Conquer: ZVP Attacks on GLV Curves". In: *Applied Cryptography and Network Security: 23rd International Conference, ACNS 2025, Munich, Germany, June 23–26, 2025, Proceedings, Part II*. Springer-Verlag, Munich, Germany, 2025, pp. 49–73. ISBN: 978-3-031-95763-5. DOI: 10.1007/978-3-031-95764-2_3

4.1 Introduction

ECC provides several efficient schemes suitable for memory-constrained devices such as smartcards, but it can fall prey to side-channel attacks (SCAs) [117]. These exploit extra information leaked from the device during the execution of the protocol — e.g., timing, power consumption, or electromagnetic radiation. While most basic SCAs against ECC can be prevented by coordinate randomization, this is not the case for RPA, EPA or the most general one — the ZVP attack [1]. The ZVP attack repeatedly forces a zero intermediate value during the computations of an addition formula in the ECDH protocol to recover the upper bits of a static private key. For each bit of the private key, the attacker needs to solve an instance of the DCP [184]. The difficulty of the DCP depends on the implemented addition formulas, but generally it remains an open problem to find an efficient algorithm for the DCP. If the implemented addition formula contains only hard DCPs, only a few bits of the private key can be recovered. Hence, existing progress on the ZVP attacks focuses on addition formulas containing easy instances of DCP [1, 140, 184].

Meanwhile, Gallant et al. [82] introduced GLV curves equipped with efficiently computable endomorphisms, which allow us to replace single-scalar multiplication with a faster multiscalar multiplication using two

scalars. Extending this idea often led to record-breaking ECC implementations [80, 122, 92, 36]. A popular example of a GLV curve is secp256k1 [182], which is used in the Bitcoin Core, or several of the popular pairing-friendly curves, such as BLS12-381, used in Zcash [37]. The ZVP attack was designed for single-scalar multiplication, and it is unclear whether multiscalar multiplication algorithms are more or less vulnerable to this attack. Additionally, while it is believed that endomorphisms do not significantly help to solve the discrete logarithm problem [29], their effect on DCP has not been studied.

Contributions:

- In Section 4.2, we classify all the addition formulas from the Explicit-Formulas Database [28] with respect to the difficulty of the underlying DCP problems. We show that the EPA attack cannot be used on the secp256k1 curve, and the ZVP attack on all but one of the addition formulas in EFD supporting secp256k1 must solve hard instances of DCP. We provide a semi-automatic tool[1] for the classification of formulas with respect to other curves as well.
- In Section 4.3, we develop new ZVP attacks on the classical multiscalar multiplication algorithms (interleaving and the Straus-Shamir trick), focusing on hard DCP problems. Our attacks use a fast endomorphism to recover twice as many bits when compared to the classical ZVP attack. Additionally, for some implementations of the interleaving algorithm, our ZVP attack is able to recover 63% of the private key on average.

4.2 Classification of formulas

During the ZVP attack, the attacker forces a zero intermediate value (IV) in an addition formula by solving the corresponding DCP. Every formula contains an IV for every finite field operation, allowing the attacker to choose the one with the easiest DCP. In this section, we semi-automatically analyze DCP problems of all addition formulas from the EFD together with almost 200 popular curves. We show that secp256k1 is resistant to EPA attacks, and any ZVP attack on secp256k1 must solve hard DCP problems with the exception of one add formula.

[1] Available at https://github.com/crocs-muni/dcp-glv.

4.2.1 Methodology

Our source of formulas was the updated export[2] of the EFD from [184]. For curves, we have used the DiSSECT database of standard prime field curves [186], including 144 Weierstrass curves, 46 (Twisted) Edwards curves, and 5 Montgomery curves.

Example 2. The following is part of the three-operand code for the projective add formula by Cohen et al. [60] (Equation (2.1)):

```
t0 = Y1*Z2, t1 = Y2*Z1, u = t1-t0, t2 = X1*Z2, t3 = X2*Z1,
v = t3-t2, t4 = u^2, t5 = v^3, t6 = v^2,..., A = t12-t9,
X3 = v*A, Y3 = t20-t19, Z3 = t21*t22
```

Given an elliptic curve, every IV can be expressed as a polynomial function in the coordinates of the input points of the formula:

$$IV = f(X_1, Y_1, Z_1, X_2, Y_2, Z_2), \quad f \in \mathbb{Z}[X_1, Y_1, Z_1, X_2, Y_2, Z_2]. \tag{4.1}$$

For instance, for the third IV in Example 2, we have `u = (Y2*Z1-Y1*Z2)`. For any intermediate polynomial $IV = f$ and a factorization $f = f_1 \cdots f_r$ we define the set of factors $\mathcal{F}_{IV} = \{f_1, ..., f_r\}$. For each formula F and a curve E, we then consider the set of polynomials $\mathcal{I}_{F,E}$ containing the factors of the polynomials of every IV in F, i.e., $\mathcal{I}_{F,E} = \bigcup_{IV} \mathcal{F}_{IV}$, where the union goes through all the intermediate polynomials of the formula F. Any ZVP attack on the formula must then use one of the polynomials in $\mathcal{I}_{F,E}$ to force a zero IV.

We label any polynomial $f \in \mathcal{I}_{F,E}$ as *easy* if it depends on the coordinates of only one point: $f \in \mathbb{Z}[X_1, Y_1, Z_1]$ or $f \in \mathbb{Z}[X_2, Y_2, Z_2]$. We have seen in Section 2.5.1 that the DCP is easily solvable in such cases. Any other polynomial in $\mathcal{I}_{F,E}$ is labeled *hard*. Moreover, we exclude the following polynomials f from $\mathcal{I}_{F,E}$:

- easy polynomials f that do not have a solution corresponding to a point on the curve E, including constant polynomials that are trivially without roots.
- the few f for which $f = 0$ depends on the choice of the projective representation of the input point(s). This choice is based on the assumption that a projective randomization of points is in place. It also allows us to transform all polynomials in $\mathcal{I}_{F,E}$ to an affine form by putting $Z_i = 1$.

[2] Available at https://github.com/crocs-muni/efd.

- f s.t. $f = 0$ implies that one of the coordinates of the input or output points (e.g., `X3` in Example 2) is 0. Recall that forcing zero coordinates of points is the domain of RPA and EPA attacks (Section 2.5). We exclude RPA attacks in our analysis of formulas since they do not depend on the choice of formula. We also exclude the general EPA attacks (with the exception of EPA on secp256k1 in Section 4.2.3) as their analysis has been done in [184].
- f s.t. $f = 0$ implies that the input or output points are of small order or one is a small multiple of the other. The assumption is that such points do not appear together in the formulas during the scalar multiplication (except maybe at the beginning of the scalar multiplication) as their relative discrete log is a part of some random private scalar. This assumption depends on the overall context and so we also recomputed the analysis without it. All of these polynomials are documented in our repository. As an example:
  - $f = X_1 - X_2$. Any points $(X_1, Y_1), (X_2, Y_2)$ that are solutions to $f = 0$ are the same or differ by a sign, i.e., $(X_1, Y_1) = \pm(X_2, Y_2)$.
  - $f = 2X_1^3 - 3X_1^2X_2 + X_2^3 - Y_1^2 + 2Y_1Y_2 - Y_2^2 = 0$. Any points that are solutions to f satisfy $(X_2, Y_2) = 2 \cdot (X_1, Y_1)$ or $X_1 - X_2 = 0$.

All of the above exceptions can be checked automatically, with the exception of the last one, which we did manually. We then classify each formula for each curve as either *resistant*, *semi-resistant*, or *vulnerable*, depending on the properties of $\mathcal{I}_{F,E}$. A formula is resistant if $\mathcal{I}_{F,E}$ is an empty set. Vulnerable formulas have at least one easy polynomial $\mathcal{I}_{F,E}$. Semi-resistant formulas are the remaining ones, i.e., $\mathcal{I}_{F,E}$ contains at least one hard polynomial and no easy polynomials. Semi-resistant formulas are vulnerable to ZVP attacks as well, but the underlying DCPs are difficult to solve. Formulas for dbl cannot be semi-resistant as they depend on one input point by definition, and so all of the polynomials in $\mathcal{I}_{F,E}$ are easy if there are any.

4.2.2 Results

We have analyzed over 200 formulas with almost 5000 IV polynomials in combination with 195 curves. We present here the results for Weierstrass add and dbl formulas with four popular curves, including the secp256k1 curve. The rest of the results can be found in our repository.

	#IV	P-256	secp256k1	P-384	P-521
resistant	0	jac(12) mod(3),xyz(6) proj(9)	jac(12) mod(3), xyz(2) proj(5), xz(5)	jac(2) proj(2) xyz(3)	jac(2) proj(2) xyz(3)
vulnerable	1	proj xz(5)		jac(10) mod(3), xyz(3) proj(7), xz(3)	jac(4), mod(3) proj(3) xyz(1), xz(3)
	2	proj(2)		proj(2)	jac(6), proj(4) xyz(2), xz(2)
	3			proj, xz(2)	
	5				proj
	6				proj(2)

Table 4.1: Distribution of IV polynomials for dbl formulas and four popular curves. Formulas are grouped by their coordinate system with their amount in brackets. Each formula is either resistant or vulnerable with the number of easy IV polynomials in rows.

Table 4.1 shows the results for dbl formulas. For clarity, formulas are grouped by their coordinate systems with the number of formulas in brackets. The coordinate systems are abbreviated: jac for jacobian, proj for projective, mod for modified jacobian, xz for xz, xyz for xyzz. The formulas are divided into either resistant or vulnerable, as per our definition above. The rows further divide the formulas per the number of easy IV polynomials (e.g., resistant formulas have none). For example, for P-256, there are two vulnerable projective formulas with two easy IV polynomials and twelve resistant jacobian formulas. Clearly, the resistance of formulas depends on the choice of the curve. While most formulas are vulnerable for P-521, the opposite is true for P-256. Surprisingly, none of them are vulnerable for secp256k1. This means that the ZVP attack cannot target any dbl formula from EFD if the secp256k1 curve is used.

Similarly, Table 4.2 shows the results for add formulas. It turned out that there are no resistant Weierstrass add formulas, and so formulas are only divided into semi-resistant or vulnerable. This means that each formula either contains an easy IV polynomial (vulnerable formula) or no easy ones and at least one hard IV polynomial (semi-resistant formula). The number of these polynomials is again in the rows. Since the semi-

	#IV	P-256	secp256k1
semi-resistant	1		jac(22), mod(3), proj(4), xyz(3)
	2	jac(22), mod(3), proj(8), xyz(6)	proj(2)
	3		mod(1)
	4	proj(4), mod(1)	
	5		proj(1)
vulner.	2		proj(2)
	4	proj(4)	

Table 4.2: Distribution of IV polynomials for add formulas and for P-256 and secp256k1. Formulas are grouped by their coordinate system with their amount in brackets. Each formula is either semi-resistant or vulnerable with the number of hard (for resistant) or easy (for vulnerable) IV polynomials in rows. There are no resistant Short Weierstrass add formulas in EFD.

resistance of formulas did not change significantly between curves, we only show the results for P-256 and secp256k1. Focusing on secp256k1, we found the following eleven hard IV polynomials shared between the formulas, which we will use for our attacks (Section 4.3).

$$
\begin{aligned}
f_1 &= X_1 + X_2, \quad f_3 = X_1^4 + X_2^4 + (X_1 + X_2)^4 - 3(X_1 + X_2)(Y_1 + Y_2)^2, \\
f_2 &= 2X_1(X_1 - X_2)^2 - (Y_1 - Y_2)^2, \quad f_4 = (X_1 - X_2)^3 + (Y_1 - Y_2)^2, \\
f_5 &= X_1(X_1 - X_2)^2 - (Y_1 - Y_2)^2, \quad f_6 = X_1 - X_2 - 1, \\
f_7 &= X_1 - X_2 - 2, \quad f_8 = Y_1Y_2 - 3b, \quad f_9 = Y_1Y_2 + 3b, \\
f_{10} &= X_2Y_1 + X_1Y_2, \quad f_{11} = X_1X_2 + Y_1Y_2
\end{aligned}
$$

Out of the 36 semi-resistant add formulas: 32 formulas contained one IV polynomial (f_1, f_2, f_4 or f_5), two formulas contained two polynomials (both f_1, f_3), one formula contained f_4, f_6, f_7 and one formula contained $f_1, f_8, f_9, f_{10}, f_{11}$. The conclusion for secp256k1 is that the ZVP attack on any classical scalar multiplication algorithm using dbl and add must focus on the add with all but two formulas being semi-resistant with hard DCPs (the exceptions are two variants of the formula by Renes et al. [170]). Furthermore, the amount of possible IVs is limited (only one for most

formulas). We have to emphasize that we do not make any claims for formulas that are not in the EFD database.

Curve25519. To illustrate our results on other operations and curve models, we will summarize our findings for the popular Curve25519 in the Montgomery form. All four formulas for the differential addition are resistant (contain no IV polynomials usable for ZVP). Similar holds for ladder and doubling formulas with the exceptions of `xz:ladd-1987-m` and `xz:dbl-1987-m`. Both of these formulas contain an easy IV polynomial $X_1 - a$, where a is the Montgomery model parameter. Surprisingly, all of the considered Montgomery curves have a solution point for this polynomial (M-221, M-511, Curve383187 or M-383) except for Curve25519. This means that all of the formulas supporting Curve25519 are resistant against ZVP with this curve.

Remark 3. We have treated the detection of zero values as a black-box oracle, assuming the ability to detect operations involving zeroes using side channels. In practice, it might be crucial to distinguish the operations as the gate counts might differ for (modular) addition/subtraction and multiplication. The attacker is expected to require more effort to detect a zero addition than a multiplication by zero [1]. If we consider only multiplication to be detectable, the presented results do not change for Weierstrass formulas. Our repository contains these results as well.

4.2.3 Resistance of secp256k1 to EPA and RPA

We excluded the EPA attack (a special case of ZPV) in our analysis of formulas as it was already done by Sedlacek et al. [184]. The EPA targets a polynomial f, for which $f = 0$ causes an error during the scalar multiplication. As shown in [184], all known add formulas in the EFD database for short Weierstrass curves offer only one f that could be zero: $y_1 + y_2$. The authors focused on the general case and did not consider how the situation changes for particular curves. We will now show that for the secp256k1 (and the general GLV curves) that $\mathrm{DCP}_{y_1+y_2}(k, g)$ has a solution only for three values of $\frac{k}{g}$. Since k corresponds to a prefix of a random secret scalar (target of the EPA attack), the probability of encountering such k and g is negligible, making the secp256k1 curve resistant to EPA.

The GLV curves of the form $y^2 = x^3 + b$ admit an endomorphism $\lambda(x, y) = (\beta x, y)$, where $\beta \in \mathbb{F}_p$ is a (nontrivial) third root of 1. If P is

a solution to $\mathrm{DCP}_{y_1+y_2}(k, g)$, then $Q = kP$, $G = gP$ and $y_Q + y_G = 0$. Squaring the equation and replacing y_Q^2 and y_G^2 using the curve equation, we get $x_Q^3 = x_G^3$, which means that either $x_Q = x_G$, $x_Q = \beta x_G$ or $x_Q = \beta^2 x_G$. These three cases imply, respectively, that either $Q = -G$, $Q = -\lambda G$, or $Q = -\lambda^2 G$. Consequently, $\frac{k}{g} \in \{-1, -\lambda, -\lambda^2\}$ as claimed.

Similarly for ladd formulas, the only potential zero polynomial is $f = x_2$ [184]. This means that as long as the curve does not contain points with zero x-coordinate, the curve is resistant to EPA attack. This is the case for secp256k1 as the curve equation is $y^2 = x^3 + b$ with nonsquare b. For the same reason, secp256k1 is resistant to RPA attack, for which a zero coordinate point is necessary. Thus, if the target is secp256k1, only the general ZVP attack is relevant and is also the focus of the next section.

4.3 ZVP-GLV attack

During the classical ZVP attack as described in Section 2.5, the attacker needs to find a solution to a $\mathrm{DCP}_f(k, g)$ where k is a small prefix of the private key d and g is the guess on the next digit. The situation changes when the implementation uses the GLV decomposition $d = d_0 + d_1\lambda \pmod n$. The multiscalar multiplication algorithms gradually go through the prefixes of both the scalars d_0 and d_1 at the same time. The attacker would now need to solve $\mathrm{DCP}_f(k_0 + k_1\lambda, g)$, where both k_0, k_1 are small but $k_0 + k_1\lambda$ is not. Since the state-of-the-art approach for solving DCP works only for small k, it seems that the ZVP attack cannot be used for implementations using the GLV decomposition. This section shows that the opposite is true:

- We show that $\mathrm{DCP}_f(k_0 + k_1\lambda, g)$ can be solved efficiently for small k_0, k_1, g even though $k_0 + k_1\lambda$ is a large scalar. Moreover, we can solve such DCPs for t-bit k_0 and t-bit k_1 in a comparable time as $\mathrm{DCP}_f(k, g)$ for a single t-bit k.
- We design a ZVP-GLV attack that can recover upper digits of both scalars d_0, d_1 at the same time. As a result, the ZVP-GLV attack can recover twice as many digits of the secret scalar d when compared to an implementation not using the GLV decomposition.

Firstly, Section 4.3.1 describes a method for solving $\mathrm{DCP}_f(k_0 + k_1\lambda, g)$. In Section 4.3.2, 4.3.3 and 4.3.4, we use the method to attack the Straus-Shamir trick (Algorithm 3) and the interleaving algorithm (Algorithm 2).

We also show that one variant of the interleaving algorithm is especially vulnerable to ZVP attack, allowing the attacker to recover the majority of the digits of d (Section 4.3.5). All of the sections contain experimental results on the secp256k1 curve as well.

Attacker model. The target is a device that performs MSM $P \mapsto dP$ on a GLV curve $E/\mathbb{F}_p : y^2 = x^3 + b$ using a fixed secret scalar $d = d_0 + d_1\lambda \pmod{n}$. Here, n is the order of $E(\mathbb{F}_p)$, λ is the action of an endomorphism ϕ on $E(\mathbb{F}_p)$, where $\lambda^2+\lambda+1 = 0 \pmod{n}$. The MSM is done using either Algorithm 3 or Algorithm 2. The scalars d_0 and d_1 have l digits and are encoded in GLV-SAC for Algorithm 3 and in w-NAF for Algorithm 2). The attacker controls the input point P and can detect (using a suitable side channel) whether a zero appears during the computation of add, i.e., they have an oracle $\mathcal{O}$ described in Section 2.5 (we extend this model in Section 4.3.5). The attacker's goal is to recover as many digits of d as possible using $\mathcal{O}$. Our experiments are simulated, i.e., we **do not** perform any side-channel measurements and treat $\mathcal{O}$ as a black-box oracle.

4.3.1 DCP on GLV curves

The state-of-the-art approach to solving $\mathrm{DCP}_f(k, g)$, as described in Section 2.5.1, is to find a root of a univariate polynomial $\overline{f}$ of degree $\approx 2\max(k, g)^2 \deg f$. As we show in Lemma 1, we can extend the approach to $\mathrm{DCP}_f(k_0+k_1\lambda, g)$ where both k_0, k_1 are small. We construct a polynomial F of degree $\approx 16\max(k_0, k_1, g)^2 \deg f$ and then look for roots of F. The idea behind the construction of F is that the multiplication-by-k map can be replaced by two multiplication-by-k_i maps while the degree of the map for λ is one, bringing no additional cost.

Lemma 1. *Let $E/\mathbb{F}_p$ be a GLV curve with an endomorphism ϕ and an action λ. For $k_0, k_1, g \in \mathbb{Z}$ and $f \in \mathbb{F}_p[x_1, x_2]$, consider the $DCP_f(k_0+k_1\lambda, g)$ and assume that not every point in $E(\mathbb{F}_p)$ is a solution to this DCP. There is a nonzero polynomial $h \in \mathbb{F}_p[x]$ of degree $< \delta = 16\max(k_0, k_1, g)^2 \deg f$, such that if $P \in E$ is a solution to the DCP then x_P is a root of h. The polynomial h can be found with the time and space complexity $O(M(\delta^2)\log\delta\log\delta^2 p)$.*

Proof. Denote $F_x = \mathbb{F}_p(x)$ and let $u_0, u_1, u_g \in F_x$ be the rational functions for the x-coordinates of multiplication maps by k_0, $k_1\phi$ and g, respectively. Let $S_3 \in \mathbb{F}_p[X_1, X_2, X_3]$ be the *Semaev* polynomial S_3 which satisfies

$S_3(x_{P_0}, x_{P_1}, x_{P_2}) = 0$ for any points $P_0 \pm P_1 = P_2$ [190]. Assume that there is a solution point P to the DCP and let $Q = k_0P + k_1\lambda P$, $\overline{Q} = k_0P - k_1\lambda P$. The Semaev polynomial gives us $S_3(x_Q, u_0(x_P), u_1(x_P)) = 0$ and $S_3(x_{\overline{Q}}, u_0(x_P), u_1(x_P)) = 0$. Since P is a solution to the DCP, we also have $f(u_g(x_P), x_Q) = 0$. We will construct the desired polynomial h using a resultant. Consider the rational function $S_3(x_2, u_0(x), u_1(x))$ as a quadratic polynomial $S_3'(x_2)$ in x_2 with coefficients in F_x, i.e., $S_3' \in F_x[x_2]$. Similarly, let $f' = f(u_g(x), x_2) \in F_x[x_2]$. Denote $H \in F_x$ the resultant of the polynomials S_3' and f'.

If $H = 0$, then f' and S_3' share a common factor $c \in F_x[x_2]$. Since c divides S_3' then c vanishes at $x = x_T, x_2 = x_R$ for every pair of points $T, R = (k_0 + k_1\lambda)T$ or for every pair of points $T, R = (k_0 - k_1\lambda)T$. In the first case, every T is a solution to DCP. In the latter case, we replace f', S_3' with $f'/c, S_3'/c$ and define H as their resultant, which is now nonzero.

Assuming $H \neq 0$, the desired polynomial h is the numerator of H. It remains to compute the degree of h. The resultant H is the determinant of the Sylvester matrix of dimension $\deg S_3' + \deg f'$, where each entry is an element of F_x. The matrix is composed of two parts corresponding to the polynomials S_3' and f':

- The first $\deg S_3' = 2$ columns contain coefficients (rational functions from F_x) of f' of degree $\leq \deg u_g \cdot \deg_{x_1} f = g^2 \deg_{x_1} f$. Here, $\deg_{x_1} f$ is the degree of f with respect to the first variable x_1.
- The remaining $\deg f'$ columns contain coefficients of S_3' of degree $\leq 2(\deg u_0 + \deg u_1) = 2(k_0^2 + k_1^2)$, since S_3 is quadratic in each variable.

The determinant H is, by definition, a sum of products of the elements of the matrix. From the above, each summand is a rational function of degree smaller than $2 \cdot g^2 \deg_{x_1} f + \deg f' \cdot 2(k_0^2 + k_1^2)$. Additionally, each denominator of each summand is a product of the denominators of u_0, u_1, u_g, and so bringing to the common denominator increases the degree by $g^2 \deg_{x_1} f + 2(k_0^2 + k_1^2)$. The degree of H, and consequently of h, is then $3g^2 \deg_{x_1} f + 2(\deg f' + 1)(k_0^2 + k_1^2)$. Replacing k_0, k_1, g with $m = \max(k_0, k_1, g)$ and $\deg_{x_1}, \deg f'$ with the total degree $\deg f$, we get a conservative upper bound $\deg h < 16m^2 \deg f$. The complexity of constructing h is dominated by the construction of the maps u_0, u_1, u_g, and their substitution into S_3 with the complexity discussed in Section 2.5.1. □

Poly.	2+2b	3+3b	4+4b	5+5b
f_2	0.3s	2.5s	13.7s	70.1s
f_5	0.3s	2.3s	11.9s	61.4s
f_6	0.1s	0.4s	2.3s	10.8s
f_9	0.2s	1.5s	7.5s	38.9s
f_{10}	0.1s	1.0s	4.8s	23.6s
f_{11}	0.3s	1.7s	8.4s	42.6s

(a) $\mathrm{DCP}_f(k_0 + k_1\lambda)$

Poly.	4b	5b	6b	7b	8b
f_2	2.1s	10.4s	43.2s	252.0s	19.2m
f_5	1.8s	10.5s	57.3s	176.3s	22.4m
f_6	0.1s	1.3s	7.0s	47.2s	225.3s
f_9	1.5s	6.7s	37.1s	159.5s	15.6m
f_{10}	1.1s	4.0s	20.9s	95.4s	8.8m
f_{11}	1.4s	6.4s	36.5s	180.1s	14.9m

(b) $\mathrm{DCP}_f(k)$

Table 4.3: Timings for solving $\mathrm{DCP}_f(k_0+k_1\lambda)$ and $\mathrm{DCP}_f(k)$ for different polynomials f_i and scalar bit-lengths on the secp256k1 curve.

Note that Lemma 1 easily extends to $\mathrm{DCP}_f(k_0 + k_1\lambda, \lambda g)$ since the rational map for λg is βu_g, whereu u_g is the rational map for g and $\beta \in \mathbb{F}_p$ is the action of λ. In principle, our approach works for $\mathrm{DCP}_f(k_0 + k_1\lambda, g_0 + \lambda g_1)$ as well (it requires a higher degree Semeav polynomial S_4), but we will not need it.

Table 4.3a shows the timings[3] of solving $\mathrm{DCP}_f(k_0 + k_1\lambda)$ for scalars k_0, k_1 of small bit-lengths (2–5 bits) and a selection of IV polynomials that we found in our analysis of formulas for secp256k1 in Section 4.2. For comparison, Table 4.3b shows the timings of solving $\mathrm{DCP}_f(k)$ classically for small k. While solving $\mathrm{DCP}_f(k)$ for 8-bit k can take more than 18 minutes (for f_2), solving $\mathrm{DCP}_f(k_0+k_1\lambda)$ for two 4-bit scalars k_0, k_1 takes less than 14 seconds. In Section 4.3.2, we will show that this makes the decomposed private key $d = d_0 + d_1\lambda$ more vulnerable.

Using Lemma 1, we can efficiently solve $\mathrm{DCP}_f(k_0+k_1\lambda)$ for small k_0, k_1 *assuming* that there is at least one solution. To empirically estimate the probability that a DCP has a solution, we have solved $\mathrm{DCP}_f(k_0+k_1\lambda)$ for all positive $2-5$ bit scalars k_0, k_1 and all the IV polynomials $\{f_1, \ldots, f_{11}\}$. For each such DCP instance, we have recorded whether it has at least one solution. Additionally, as we saw in Section 4.2, some formulas have up to five IV polynomials. Hence, we have combined the results for individual IV polynomials to see how the success rate of finding a solution changes if it is enough to have at least one solution for at least one polynomial from a given set of IV polynomials. Results are in Figure 4.1 with the number of IV polynomials on the y-axis and the success rate (in percentages) on

[3]All our experiments were computed on a 2.3 GHz Intel Xeon processor. Our implementation uses C/C++ for the computationally demanding parts and Python with Sagemath for the overall interface.

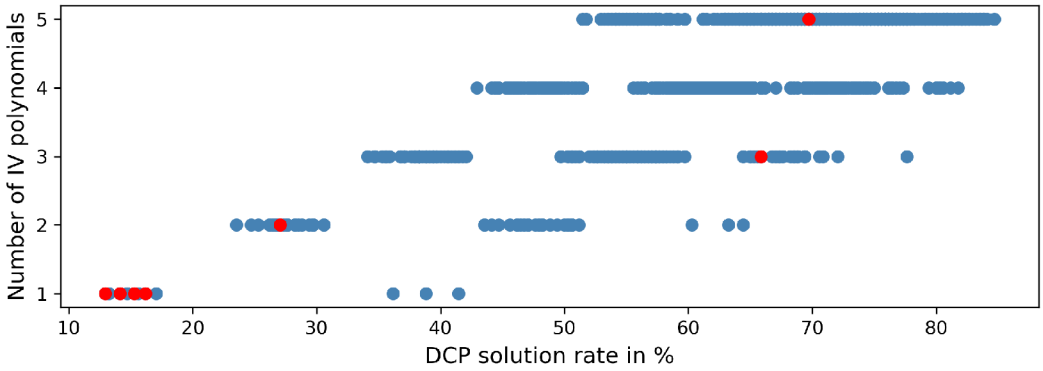


Figure 4.1: Each dot represents a subset of IV polynomials from $\{f_1, \ldots, f_{11}\}$ with their amount on y-axis and the rate of at least one solvable $\mathrm{DCP}_{f_i}(k_0 + k_1\lambda)$ problem on the x-axis. The red dots represent EFD `add` formulas.

the x-axis. Each dot represents a set of IV polynomials of size $1, 2, 3, 4$ or 5. Surprisingly, the DCP success rate for single polynomials is very low (below 25% for all except for f_6, f_7, f_{11}), which limits the ZVP attack on formulas with single IVs. With at least three IV polynomials, the average rate is over 50%, and with five IV polynomials, the rate reaches 87%.

The red dots represent the IV polynomials corresponding to `add` formulas from EFD. The majority of EFD formulas have a $13 - 16\%$ success rate. The exceptions are `proj:add-2002-bj` with 27% (2 polynomials), `mod:mmadd-2009-bl` with 66% (3 polynomials) and `proj:madd-2015-rcb` with 70% (5 polynomials). While the success rate is low for almost all formulas, some formulas (and all the polynomial combinations in blue dots) show that the rate can vary over a wide range. This is especially relevant for the development of new formulas.

4.3.2 Description of the ZVP-GLV attack

This section describes a new ZVP attack on implementations using GLV decompositions (ZVP-GLV attack). We target both the Straus-Shamir trick (Algorithm 3) and the interleaving algorithm (Algorithm 2). We will go through the general idea of the attack and then discuss details specific to the individual algorithms.

The general idea of the ZVP-GLV attack follows the idea of the classical ZVP attack: we iteratively recover the upper digits of d_0, d_1, where $d = d_0 + d_1\lambda$ is the decomposition of the private key. In each iteration,

we take a guess of the digits $d_0^{(i)}$, $d_1^{(i)}$. To verify our guess, we attempt to invoke a zero during the computation of add with a special input point found as a solution to DCP using Lemma 1. If our guess is correct, then we detect a zero. It might happen that the DCP does not have a solution. As we saw in Figure 4.1, the probability of that event is relatively high (as high as 88%) unless multiple intermediate polynomials are used. This means that the individual guesses for $d_0^{(i)}, d_1^{(i)}$ might not be verified and we must keep track of all possibilities for the values of $d_0^{(i)}, d_1^{(i)}$.

We denote K_i the set of candidates for $(d_0^{\geq i}, d_1^{\geq i})$. We start with K_{l-1} containing all possible pairs of the most significant digits of d_0, d_1 and iteratively transform K_i to K_{i-1} until we reach the limit given by the complexity of DCP.

$$K_{l-1} \to K_{l-2} \to \ldots K_i \to K_{i-1} \to \ldots K_{l-t} \mid \text{limit given by the complexity of DCP.}$$

The sizes of the sets K_i might fluctuate, but each K_i contains the correct pair of prefixes $(d_0^{\geq i}, d_1^{\geq i})$. The following steps transform K_i to K_{i-1}:

- For each candidate pair $(\kappa_0, \kappa_1) \in K_i$, we make a guess g_0, g_1 for the following digits $d_0^{(i-1)}, d_1^{(i-1)}$. The set of options for g_0, g_1 depends on the representation (GLV-SAC for Algorithm 3 or regular w-NAF for Algorithm 2).
- For each candidate pair κ_0, κ_1 and a guess g_0, g_1 we create an instance of the DCP and attempt to find a solution point P using Lemma 1. If we have multiple intermediate polynomials available, we create and solve a DCP for each one of them until a solution is found or we have no more polynomials. Note that some candidate pairs and guesses might have a common solution point (see Remark 1).
- Using all found points P, we query the zero detection oracle $\mathcal{O}(P)$. We reject all candidate pairs and guesses for which a zero was not detected, as they have to be incorrect.
- Any of the candidate pairs κ_0, κ_1 and guesses g_0, g_1, for which a zero was detected on the corresponding point P, might be correct as some of them might share the point P. We put $(\kappa_0|g_0, \kappa_1|g_1) \in K_{i-1}$ for all of these (where $|$ stands for concatenation).
- If no zeros are detected (and so far $K_{i-1} = \{\}$) for any point P, then the correct candidate pair κ_0, κ_1 and guess g_0, g_1 must be one of the ones for which the DCP had no solution point, and so we put all

of these in K_{i-1}. In principle, we should keep the guesses without solutions even if a zero was detected, as we might lose the correct guess g_0, g_1 due to accidental zeros. We never found this to be the case in our experiments and so we use this heuristic for more efficiency.

At the end, we are left with K_{l-t}, a set of candidate pairs for $d_0^{\geq l-t}, d_1^{\geq l-t}$. While the attack targets $2t$ digits ($t+t$ for each d_i), it recovers them with the ambiguity given by the size of K_{l-t}. For comparison between different representations, we will express the amount of information gained in bits. All pairs of prefixes with t digits give us $2tw$ bits (where w is the window size for interleaving and $w = 2$ for GLV-SAC) with $\log K_{l-t}$ bits uncertainty. Hence, by the number of recovered bits, we will mean the value $2wt - \log K_{l-t}$.

The following subsections describe the details specific to Algorithm 3 and Algorithm 2 and the experimental results. The pseudocodes for the attacks can be found in Appendix B.2.

4.3.3 ZVP-GLV attack on Straus-Shamir trick

The ZVP-GLV attack on the Straus-Shamir trick (Algorithm 3) iteratively uncovers the GLV-SAC representation of the upper bits. This means that $(g_0, g_1) \in \{(1,0), (-1,0), (1,1), (-1,-1)\}$, using the general notation from above. We will now explain how to construct the DCP for GLV-SAC. The rest of the attack follows the general description above.

A guess (g_0, g_1) for $d_0^{(i)}, d_1^{(i)}$ corresponds to a guess of the input point $(g_0 + g_1\lambda)P \in \{P, -P, P + \lambda P, -P - \lambda P\}$. Given a prefix candidate (κ_0, κ_1), we verify the guess by solving $\mathrm{DCP}_f(\kappa_0 + \kappa_1\lambda, g_0 + g_1\lambda)$, i.e., find P such that $f(x_G, x_Q) = 0$, where $G = (g_0 + g_1\lambda)P$ and $Q = (2\kappa_0 + 2\kappa_1\lambda)P$ using Lemma 1. It might seem like a complication that Lemma 1 can only solve $\mathrm{DCP}_f(\kappa_0 + \kappa_1\lambda, g)$ for small g, while $g_0 + g_1\lambda$ is not small. However, we can easily reduce this problem to a DCP with $g = 1$. Denote $\gamma = 2g_0\kappa_0 + 2g_1\kappa_1 - 2g_1\kappa_0 + \lambda(2g_0\kappa_1 - 2g_1\kappa_0)$. Then $Q = \gamma G$, as $2\kappa_0 + 2\kappa_1\lambda = \gamma(g_0 + g_1\lambda)$ (using the facts that $g_0^2 = 1, g_0g_1 = g_1^2$, and $\lambda^2 + \lambda + 1 = 0$). This means that G is a solution to $\mathrm{DCP}_f(\gamma)$. To find P, we just compute $P = (g_0 + g_1\lambda)^{-1}G$.

The experimental results for this attack are in Table 4.4. For comparison, we also simulated the classical ZVP attack on the regular 2-NAF double-and-add. We ran the experiments for four addition EFD formulas with a different number of IV polynomials. The target bits were 4, 6,

Formula	2+2b		3+3b		4+4b		5+5b	
`proj:madd-2015-rcb(5)`	14.4s	3.7b	23.3s	5.8b	47.6s	7.2b	102.4s	9.0b
`mod:mmadd-2009-bl(3)`	4.1s	1.4b	33.6s	2.9b	166.5s	5.3b	11.8m	7.8b
`proj:add-2002-bj(2)`	2.4s	1.2b	34.4s	2.1b	241.3s	5.6b	15.2m	8.0b
`jac:add-1998-cmo(1)`	0.5s	1.2b	10.4s	1.4b	188.1s	2.2b	43.2m	3.2b

(a) ZVP-GLV

Formula	4b		5b		6b		7b	
`proj:madd-2015-rcb(5)`	11.9s	4.0b	43.5s	4.8b	384.1s	5.5b	28.3m	6.3b
`mod:mmadd-2009-bl(3)`	6.9s	3.0b	40.3s	3.0b	386.4s	4.1b	32.7m	5.4b
`proj:add-2002-bj(2)`	11.7s	2.3b	88.3s	2.6b	370.2s	4.0b	64.2m	4.9b
`jac:add-1998-cmo(1)`	4.8s	1.0b	46.4s	1.0b	420.3s	1.6b	64.4m	2.4b

(b) ZVP

Table 4.4: Experimental results for the ZVP-GLV attack on the Straus-Shamir trick and the classical ZVP attack on the double-and-add (using `secp256k1`). Each cell contains the average timing and the number of recovered bits for four formulas (number of IV polynomials in brackets) and four target bits.

8 and 10 for ZVP-GLV and only 4,5,6,7 for the classical ZVP, as more than 7 bits was too time-consuming. Note that these formulas represent the four red clusters in Figure 4.1 with their DCP solution rate 70%, 66%, 27%, and 16%, respectively. The duration and the recovery rate of both the classical ZVP and our ZVP-GLV attack improve with more IV polynomials. A lower number of IV polynomials causes a lower DCP solution rate, which increases the set of possible candidates K_i and the number of DCPs in each iteration.

Our ZVP-GLV attack was able to recover significantly more bits in less time than the classical ZVP attack. For instance, while the classical ZVP attack took 28 minutes to recover 6.3 bits (with `proj:madd-2015-rcb`), our ZVP-GLV attack recovered 7.2 bits in 48 seconds. For `proj:add-2002-bj`, the ZVP attack took 64 minutes to recover 4.9 bits, and the ZVP-GLV attack managed to recover 5.6 bits in 241 seconds. We also targeted higher bit-lengths for `proj:madd-2015-rcb` to see the advantage of ZVP-GLV. We were able to recover 16 bits with the ZVP-GLV attack in 230 minutes — the same time it took the classical ZVP attack to recover 7 bits.

Formula	3+3b ($w = 3$)		4+4b ($w = 4$)		5+5b ($w = 5$)	
`proj:madd-2015-rcb(5)`	1.1s	2.2b	5.2s	4.1b	22.6s	5.5b
`mod:mmadd-2009-bl(3)`	0.5s	3.0b	2.7s	4.5b	12.4s	5.7b
`proj:add-2002-bj(2)`	0.6s	1.3b	3.1s	1.7b	14.4s	1.8b
`jac:add-1998-cmo(1)`	0.3s	0.6b	1.9s	1.3b	8.7s	1.5b

Table 4.5: Experimental results for the ZVP-GLV attack on the interleaving algorithm. Each cell contains the average timing and the average number of recovered bits for four formulas and three target bits.

4.3.4 ZVP-GLV attack on window interleaving

The interleaving algorithm (Algorithm 2) performs two add operations in each iteration, one for each scalar d_0, d_1. Hence, our ZVP-GLV attack targets both additions separately in a similar spirit. We first make a guess g_0 for $d_0^{(i)}$, verify it and then repeat this for g_1 and $d_1^{(i)}$. To verify a prefix candidate $(\kappa_0, \kappa_1) \in K_i$ and a guess g_0 for $d_0^{(i)}$, we need to find point P such that $f(x_G, x_Q) = 0$ where $G = g_0 P$ and $Q = (2\kappa_0 + 2\kappa_1\lambda)P$. This means solving $\mathrm{DCP}_f(\kappa_0 + \kappa_1\lambda, g_0)$. For the guess g_1, the DCP instance is $\mathrm{DCP}_f(\kappa_0 + \kappa_1\lambda, g_1\lambda)$. The ZVP-GLV attack, in this case, uncovers the regular w-NAF representation, which means that it can only target multiples of the window size. Table 4.5 shows the experimental results of the attack on the first window for $w = 3, 4$ and 5. The recovery rate is lower than for the Straus-Shamir trick. Still, the attack recovered 5.7 bits (for `mod:mmadd-2009-bl`) in 12 seconds, while it took the ZVP attack 32 minutes to recover 5.4 bits.

4.3.5 ZVP-GLV attack on window interleaving (alternative)

This section describes a ZVP-GLV attack on the alternative version of the interleaving Algorithm 2 in which $d_0^{(i)}P + d_1^{(i)}\lambda P$ is computed in each iteration. For any IV polynomial f, the DCP instance corresponding to the addition $d_0^{(i)}P + d_1^{(i)}\lambda P$ is $\mathrm{DCP}_f(d_0^{(i)}, d_1^{(i)}\lambda)$. Both $d_0^{(i)}, d_1^{(i)}$ are small integers from $\{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$. On the one hand, this means that no large scalars are involved, and we have an easy instance of DCP. On the other hand, the DCP no longer depends on the accumulated scalar multiple Q, and so if a zero is detected using the simple binary oracle, we have no way of knowing in which iteration. Additionally, the zero might appear in multiple iterations. We will extend our side-

channel model and assume that the position of the zero is visible in the traces. This is a reasonable assumption since the number of dbl and add operations is constant. More formally, we will consider a side-channel oracle $\mathcal{O}^*(P) \in \{0,1\}^l$, which outputs a vector of 1s and 0s for each iteration, indicating whether a zero appeared in the iteration or not.

At the beginning of the attack, we try to solve $\mathrm{DCP}_f(g_0, g_1\lambda)$ for every possible combination $g_0, g_1 \in \{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$. These DCPs contain small scalars and can be solved in trivial time and since they do not depend on the private key, they can also be precomputed. This gives us a set $\mathcal{P}$ of solutions to the DCPs (some of them might not have a solution). The idea is to query the oracle with all the points from $\mathcal{P}$ and record in which iteration a zero was detected and in which not. However, this will not necessarily tell us which guess g_0, g_1 corresponds to which $d_0^{(i)}, d_1^{(i)}$ as some points might invoke a zero for multiple possible values of $d_0^{(i)}, d_1^{(i)}$.

For each $P \in \mathcal{P}$ we collect a list $\mathcal{T}_P$ of guesses (g_0, g_1) for which P invokes a zero. For every solution point $P \in \mathcal{P}$, we query the oracle $\mathcal{O}^*(P)$. Since the set $\mathcal{T}_P$ contains all the possible pairs for which a zero might occur, a zero is detected in iteration i if and only if $(d_0^{(i)}, d_1^{(i)}) \in \mathcal{T}_P$. Repeating this for every $P \in \mathcal{P}$, we get that $(d_0^{(i)}, d_1^{(i)}) \in \bigcap_{P\in\mathcal{P}} \mathcal{T}_P$.

As a result, the attack cannot recover the full key on its own and only reduces the space of possible values for the key. The full key can then be recovered using a baby-step giant-step algorithm (see the discussion regarding the discrete logarithm problem below). The extent of the reduction of the space depends on the used IV polynomials and the size w of the windows used in the interleaving.

Figure 4.2 shows the results of the attack on the secp256k1 curve with window size $w = 4$ (results for $w = 3, 5$ are in Appendix B.2). We ran the attack with 100 scalars for four selected addition formulas (the number of IV polynomials is in the parentheses) and one fictitious formula containing IV polynomials with a high DCP success rate. The resulting figure is a histogram of the size of the space (in bits) given by the non-recovered $d_0^{(i)}, d_1^{(i)}$. The average bit-sizes are 98, 94, 162, 210 and 51 for `madd-2015-rcb`, `mmadd-2009-bl`, `add-2002-bj`, `add-1998-cmo`, $\{f_1, f_{11}, f_6, f_7, f_9\}$, respectively. In particular, for `mmadd-2009-bl` with the 94-bit space, this means a reduction of the original 256-bit space by 63% with 47-bit complexity for the baby-step giant-step to find the remaining bits $d_0^{(i)}, d_1^{(i)}$. For 3% of the private keys, the bit-size of the

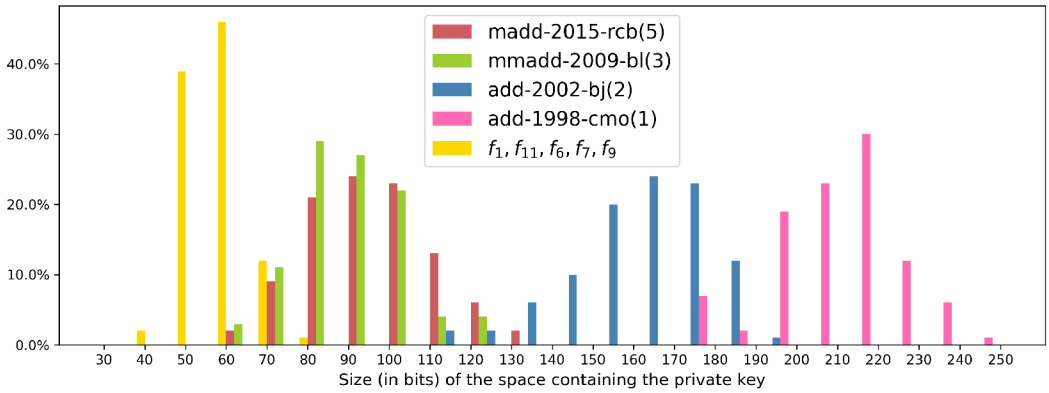


Figure 4.2: Histogram of the number of unknown bits after the ZVP-GLV attack on the regular 4-NAF interleaving algorithm. The attack was performed on 100 scalars for four different addition formulas and one fictitious formula containing IV polynomials with a high success rate.

space was below 70 bits, which might be relevant for attacks on multiple private keys. The space for the fictitious formula has 51 bits. This shows that for any formula with IV polynomials with a high DCP success rate, the private key can be fully recovered. While we have not found any such formula in EFD supporting secp256k1, it is an important takeaway for standards and developers proposing new formulas to avoid complex formulas with a large number of IV polynomials.

Finding the private key. We will now sketch how to find the private key d in the reduced space. After the ZVP-GLV attack on the interleaving algorithm, we know certain digits of the decomposed parts d_0, d_1. We can assume that we know P, Q such that $dP = d_0P + d_1\lambda P = Q$. We can now cut d_0, d_1 in halves into upper and lower digits, i.e., $d_0 = 2^{e_0}u_0 + l_0$, $d_1 = 2^{e_1}u_1 + l_1$. We iterate through the unknown parts of u_0, u_1 and store the possible values for $2^{e_0}u_0P + 2^{e_1}u_1\lambda P$. Then, we iterate through the unknown parts of l_0, l_1, computing $Q - l_0P - l_1\lambda P$ and comparing it with the stored candidates. While this baby-step giant-step approach gives us 50-bit time complexity to recover the unknown 100 bits in Figure 4.2, it requires to store 2^{50} points. Solving this problem in constant space is an open problem, e.g., using a variant of Pollard's rho algorithm [131].

4.4 Conclusions

Elliptic curves with the GLV decomposition offer a speedup of scalar multiplication. Our new ZVP-GLV attacks show that this, in the end, comes at the price of a less protected scalar in the context of side-channel attacks. In particular, this holds for the curve secp256k1 used by Bitcoin. We have designed and implemented the attacks for the Straus-Shamir trick and the interleaving algorithm — multiscalar algorithms for GLV curves used in popular libraries. As our experimental results show, the attacks can recover twice as many bits as the classical ZVP attack. Moreover, for certain implementations of the interleaving, we were able to recover more than 60% of the private key, which would be unfeasible using the classical ZVP attack with hard DCPs.

Our ZVP classification of formula-curve combinations (Section 4.2) gave us several important takeaways for standards and developers proposing new formulas for SCA protected implementations. Less complicated formulas with fewer IV polynomials are less vulnerable to ZVP attacks. Our ZVP-GLV attacks then demonstrate that the choice of formula can mean the difference between a full key recovery and a complete ZVP resistance.

The choice of the curve can significantly affect the resistance against ZVP as well. For example, for P-256, the majority of dbl formulas are resistant, and the opposite is true for P-521. The secp256k1 curve is resistant to EPA and RPA attacks, and the Curve25519 is resistant to all ZVP attacks with all formulas from EFD. The tools and complete results of our analysis and attacks are available at https://github.com/crocs-muni/dcp-glv.

5 Faster signature verification with endomorphism

This chapter is based on the following manuscript accepted for publication:

[VS5] V. Suchanek, M. Sys, and Ł. Chmielewski. *Faster signature verification with 3-dimensional decomposition.* Accepted to: International Conference on Applied Cryptography and Network Security – ACNS 2026.

5.1 Introduction

Classical elliptic curve signatures, including ECDSA or Schnorr signature, are the key components of modern cryptographic systems. Their importance grew even further with their adoption into the Bitcoin cryptocurrency [142], where each transaction contains a signature that is later verified as proof of ownership. In systems like Bitcoin, where nodes frequently process and validate large volumes of transactions, the efficiency of signature verification becomes a critical performance factor. The need for efficiency was one of the key motivations behind the introduction of Schnorr signatures to Bitcoin, as they offer improved verification speed compared to traditional ECDSA signatures, along with other benefits such as linearity and smaller signature sizes.

To verify a Schnorr signature[1], one needs to perform a (multi-)scalar multiplication on an elliptic curve to verify an equality $uG + vQ = R$, where Q is the public key point, G is a generator on the curve and the scalars u, v with the point R are derived from the signature. An equivalent equation is checked in the ECDSA verification, where only the x-coordinates of both sides are checked.

[1] More variants of the verification exist. We use the one usually considered in recent works, including the Bitcoin protocol [218].

The literature contains numerous techniques to speed up scalar multiplication to compute vQ or $uG + vQ$, in particular [95, 59]. Generally, the cost of these techniques is proportional to the sizes of the scalars u and v, which motivated several methods to lower their size [82, 80]. One of them is the GLV method [82] that decomposes vQ into $v_0Q + v_1\varphi(Q)$ with smaller scalars v_0, v_1 using an efficiently computable endomorphism φ. Brumley [44] reports 50% acceleration using the GLV curves, making them the fastest elliptic curves in the OpenSSL library. More recently, GLV has been used for the acceleration of scalar multiplication on pairing-friendly curves [126, 37] such as BLS12-381 used in Zcash or Ethereum.

In principle, decomposing the scalar v into more than two smaller scalars $(v_0, v_1, v_2, \dots)$ can lead to greater speed-ups [80]. However, existing methods that support such multi-dimensional decompositions rely on elliptic curves defined over extension fields. As a result, prime field curves have, so far, been limited to 2-dimensional decompositions. This chapter introduces new techniques to enable higher-dimensional decompositions over prime field curves, as detailed in the following contributions.

Contributions:

1. We present a new technique, called GLV-3, for the verification $uG + vQ = R$ using a fast endomorphism that applies to all Schnorr-like signatures, including modified ECDSA signatures (Section 5.3). The technique brings a 20% speed-up for a single signature verification, 13% speed-up for batch verification, and a 27% speed-up for the Blind Diffie-Hellman protocol [211].

2. The results are experimentally verified on a 64-bit AMD x86 processor and on a 32-bit ARM Cortex-M4 STM32F407 microcontroller over 384-bit and 521-bit curves[2]. This demonstrates that the approach applies not only to modern CPUs, which are commonly used in server applications, but also to embedded architectures found in secure elements, such as smartcards.

3. We experimentally compare Semaev algorithm [189] with the naive Lagrange 3-dimensional algorithm for scalar decomposition. Despite Semaev's better asymptotic complexity (quadratic vs. cubic), Lagrange algorithm is up to 45% faster on both the x86 processor and the microcontroller (Section 5.3.1).

[2]The source code is available at https://github.com/vojtechsu/glv3.

Our method reduces the size of the used scalars. The application of this reduction goes beyond the acceleration of signature verification as we show in our final contribution to the area of side-channel attacks. The zero-value point attack and the exceptional procedure attack are powerful side-channel attacks that can recover a static ECDH key [1, 184]. However, the amount of information these attacks recover is limited by the underlying DCPs. As an additional fourth contribution, we prove that the 4-dimensional variant of our technique (GLV-4) reduces the complexity of DCP from $O(n)$ to $O(\sqrt{n}\log^3 n)$ (Section 5.4). This is the first general and non-trivial reduction of DCP, previously solvable only via brute force.

Table 5.1: The speed-up of our 3-dimensional GLV technique when compared to the state-of-the-art approach for three different verification algorithms.

	bits	AMD x86	ARM Cortex-M4
Single Schnorr	384	18%	18%
	521	19%	20%
Batch Schnorr	384	11%	11%
	521	13%	13%
BDHKE	384	23%	25%
	521	24%	27%

5.2 Background

This chapter summarizes the state-of-the-art methods for the verification of signatures. We mainly focus on Schnorr signatures as presented in Section 2.3 but the ideas can be applied to modified ECDSA signatures and the BDHKE protocol that we recall below.

Modified ECDSA. The classical ECDSA computes a point R using the nonce value and outputs the x-coordinate $r = x_R$ as part of the signature. Antipa et al. [6] realized that if the y-coordinate y_R was not forgotten, it could be used to speed-up the verification. They considered two versions of ECDSA (denoted ECDSA*) which include y_R in the signature or append enough information for its reconstruction. For prime order curves, including the popular NIST curves or secp256k1, one bit of information is enough.

BDHKE. The Blind Diffie-Hellman Key Exchange (BDHKE) protocol [211] is not a classical signature algorithm, but uses the idea of the BLS signature (Section 2.3) and serves as another target of our work. The goal of this protocol is to make the signer sign a blinded value for a second party (user) so that when the unblinded value is revealed, the signer can verify it but cannot trace it back to the user. The exact steps of the protocol between a user and a signer are the following:

- User computes the point $h(m)$ as in the BLS signature and blinds it by adding a factor rG for a random r: $C = H_E(m) + rG$. The user sends C to the signer.
- The signer computes $T' = dC$ and sends it back to the user.
- The user unblinds T' using the signer's public key: $T = T' - rP$.
- Later, when the signer is provided (m, T), they verify $dH_E(m) = T$.

The protocol has been used for Cashu — an e-cash system backed by Bitcoin [155]. One of the key properties of this protocol is that the verification is done only by the owner of the private key. This means that the costly pairing-based verification can be replaced by a simple signature reconstruction.

Remark 4. The verification equation in the popular EdDSA signature scheme [21] is the same as in the Schnorr signature. Yet, we exclude it from this work for the following reasons. EdDSA is designed for Edwards curves with complete addition formulas. More precisely, EdDSA uses a curve of the form $-x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{F}_p$, where $d \in \mathbb{F}_p$ is a non-square and $p = 1 \pmod 4$ (implying that -1 is a square in $\mathbb{F}_p$). Our work uses GLV curves that have an efficient endomorphism of degree 1 that are of the form $y^2 = x^3 + b$ or $y^2 = x^3 + ax$; these curves are precisely the ones with endomorphism ring with nontrivial units that correspond to endomorphisms with their rational maps of degree 1. Such GLV curves cannot be transformed to the Edwards curve. The reason follows from the birational equivalence between Weierstrass curves $y^2 = x^3 + ax + b$ and twisted Edwards curves [25]. It can be shown that the Edwards d coefficient must be equal to $3\alpha^2 + a$ for the equivalence to exist, where α is a root of $x^3 + ax + b$. For both considered forms of GLV curves, simple expression manipulation shows that d is always a square, breaking the EdDSA assumption. Nevertheless, curves with less efficient endomorphisms (of higher degree) could be considered, but we leave this as a future work.

5.2.1 Verification of signatures

The focus of this work is the verification of Schnorr signatures (VSS), ECDSA* signatures, and the verification of scalar multiplication (VSM) in BDHKE. These can be generalized by the following two problems:

(VSM) Given two affine points P, R and a scalar $k \in \mathbb{Z}$, decide:

$$kP \stackrel{?}{=} R.$$

(VSS) Let G be a fixed generator. Given two affine points Q, R and two scalars $u, v \in \mathbb{Z}$, decide:

$$uG + vQ \stackrel{?}{=} R.$$

The first and obvious approach to both problems is to recompute kP or $uG + vQ$ and compare with the other side of the equation — the point R. The ECC literature contains a plethora of techniques for (multi-)scalar multiplication [95, 59]. The overall principle is to iterate over the digits or windows of the scalars in some representation. Depending on the context, the techniques might assume that one of the points is fixed and leverage some precomputation. Additionally, the threat of side-channel attacks is also often considered in the literature.

In our context, we assume that only the point G is fixed in VSS. Furthermore, we do not consider side-channel attacks, as only public data are verified. Our work follows the observation that the cost of (multi-)scalar multiplication is proportional to the bit-lengths of the involved scalars. Two notable techniques that are based on this idea can be summarized in the following way.

First, the GLV method (Section 5.2.2) decomposes the scalar k (or v for VSS) into $k = k_0 + k_1\lambda_\varphi \pmod n$ using a scalar λ_φ and then applies Straus-Shamir trick to check whether $k_0P + k_1\lambda_\varphi P = R$. It works with the assumption that $\lambda_\varphi P$ can be efficiently computed as $\varphi(P)$.

$$kP \stackrel{?}{=} R \quad \rightarrow \quad k = k_0 + k_1\lambda_\varphi \quad \rightarrow \quad k_0P + k_1\varphi(P) \stackrel{?}{=} R \qquad \text{(GLV)}$$

The second method by Antipa et al. [6] (Section 5.2.3) similarly splits the scalar k by decomposing 0 into $0 = k_0 - k_1k \pmod n$ using the scalar k. Then it applies the Straus-Shamir trick to check $k_0P - k_1kP = \infty$. The

assumption here is that the equation to be verified $kP = R$ holds and kP can be efficiently 'computed' by simply taking $kP = R$.

$$kP \stackrel{?}{=} R \quad \rightarrow \quad 0 = k_0 - k_1 k \quad \rightarrow \quad k_0 P - k_1 R \stackrel{?}{=} \infty \qquad ([6])$$

We will explain the full details of the above ideas in Section 5.2.2 and Section 5.2.3. To express the cost of the described algorithms, we use the following notation: A and D denote the cost of point addition and doubling, respectively. Also, recall that we work with points of order n, where n has $\ell(n)$ bits.

5.2.2 Verification with GLV

The fastest approach to implement GLV (as described in Section 2.6) is the Straus-Shamir trick with the JSF form. We will now compute the cost for VSM in terms of the number of point additions A and doublings D, as well as the number of field multiplications M, i.e., the cost in the form $c_1 A + c_2 D + c_3 M$. We will also use the standard Jacobian formulas [28] with $A = 11M$, $D = 7M$, to express the cost purely in the number of multiplications, i.e., cM. We have selected these formulas as they are generally the fastest for Weierstrass curves with $a = 0$ (in terms of the number of field multiplications), and they are also implemented in the Mbed TLS library that we used for experiments. Furthermore, we assume that the curve admits an endomorphism φ with the cost of its application equal to one finite field multiplication.

The Straus-Shamir trick is described by Algorithm 3, where $d_0^{(i)}, d_1^{(i)} \in \{0, 1, -1\}$ for the JSF form. For the precomputation (Step 1 of the algorithm), it is helpful to use the fact that opposite points differ only by a sign in the y-coordinate and so we can compute and store only half the points. This amounts to the input point P and three points $\varphi(P)$, $P + \varphi(P) = -\varphi(\varphi(P))$, $P - \varphi(P)$ that can be computed with the cost M, M, A, respectively. The evaluation part uses $\frac{\ell(n)}{2}$ doublings. The number of additions is given by the joint Hamming weight of the two $\frac{\ell(n)}{2}$bit scalars in JSF, which is $\frac{\ell(n)}{4}$ on average. The total cost is then

$$\left(\frac{\ell(n)}{4} + 1\right) A + \frac{\ell(n)}{2} D + 2M = (6.25\ell(n) + 13)M.$$

The idea can be adapted to the VSS problem (as in the libsecp256k1 library [217]) by decomposing both the scalars u, v and computing $u_0G + u_1\varphi(G) + v_0Q + v_1\varphi(Q)$ using two simultaneous Straus-Shamir tricks. We precompute the points for $u_0G + u_1\varphi(G)$ and $v_0 + v_1\varphi(Q)$ separately and then run the evaluation at the same time. The combined Hamming weight of the two pairs of scalars in JSF is $2\frac{\ell(n)}{4} = \frac{\ell(n)}{2}$. The overall cost is then

$$\left(\frac{\ell(n)}{2} + 2\right)A + \frac{\ell(n)}{2}D + 4M = (9\ell(n) + 26)M.$$

5.2.3 Verification by Antipa et al.

Antipa et al. [6] do not apply the simple recomputation of the (multi-)scalar multiplication to the VSS and VSM problems. Instead of using the point R just for the final comparison, they use it for the computation of the scalar multiplication. The idea is to express k as a fraction $k = \frac{k_0}{k_1}$ of two smaller scalars of half the bit-length of k. Or from another point of view, we decompose 0 as $0 = k_0 - k_1k$ using k. The pair (k_0, k_1) is a short vector in the lattice

$$\langle (z_0, z_1) \mid z_0 - z_1k = 0 \pmod{n} \rangle$$

and can be found using EEA as described in Section 2.6.1. To solve VSM, we then use the Straus-Shamir trick to compute $k_0P - k_1R$ and check if the result is ∞.

Applying this method to VSS is a bit more tricky and needs to use the fact that G is fixed: We can decompose $0 = v_0 - v_1v \pmod{n}$, $v_0, v_1 \approx n^{1/2}$, which gets us $uv_1G + v_0Q - v_1R = \infty$. The only scalar bigger than $n^{1/2}$ is uv_1, but we can use the fact that G is a fixed point and the multiple $G' = 2^{\ell(n)/2}G$ can be precomputed once and stored. Using G', we can split uv_1 into the upper and lower binary halves u_U, v_L which gets us the following verification equation:

$$u_LG + u_UG' + v_0Q - v_1R = \infty.$$

The left side is then computed as in Section 5.2.2 with a similar overall cost (slightly worse since the endomorphism cannot be used in the precomputation). Antipa et al. [6] report a speed-up of 36% on the ARM7TDMI platform when compared to the simple Straus-Shamir trick on $uG + vQ$.

Compared to the GLV approach (Section 5.2.2), this method has the advantage that no endomorphisms are necessary. One could get the idea of using both methods to reduce the size of the scalars even further. Unfortunately, they are not compatible in any trivial way. For instance, we could try to decompose $v = v_0 + v_1\lambda_\varphi$ using λ_φ and then try to express v_0, v_1 as fractions of smaller scalars. The problem is that since already $v_i \approx n^{1/2}$, they are not likely to be fractions of much smaller scalars. The EEA would most likely give us trivial equality $v_0 = \frac{v_0}{1}$. Trying the opposite way (decomposition and then fractions) suffers from the same problem. The decompositions of v_0, v_1 using λ_φ for $v = \frac{v_0}{v_1}$ are most likely to be trivial as well. Hence, a simple application of one method after the other does not work.

Antipa et al. [6] extended their idea of 4 half-size scalar multiplications to 6 third-size scalar multiplications, but assume that a multiple Q' must be precomputed and available for verification. One option is to append it to the signature, and another is to include it in the public key certificate as it depends on the public key only. Either way, this cannot be used for the current signature schemes, and so we will not discuss this further. In the following section, we present a new method with 6 third-size scalar multiplications without this assumption.

5.3 Verification using 3-dimensional GLV

This section presents the contribution of our work: a 3-dimensional GLV method that accelerates the verification of signatures (VSS) and scalar multiplication (VSM) using a fast endomorphism. The section contains theoretical analysis and results of experiments.

The two presented approaches (Section 5.2.2 and Section 5.2.3) to VSM and VSS reduced the size of the involved scalars using an appropriate decomposition. The GLV method decomposes $k = k_0 + k_1\lambda_\varphi \pmod n$ using λ_φ, with the assumption that we have an efficient endomorphism such that $\varphi(P) = \lambda_\varphi P$. The second method by Antipa et al. [6] decomposes $0 = k_0 - k_1 k \pmod n$ using k, assuming the equality $kP = R$ is true.

Our method (that we call *3-dimensional GLV* or GLV-3 for short) combines both of the methods described above and considers the decomposition of 0 by k and λ_φ at the same time: $0 = l_0 + l_1\lambda_\varphi + l_2 k \pmod n$. It essentially imitates the GLS method that uses two endomorphisms for

a decomposition by taking λ_φ and k as the endomorphisms. For VSM, we verify the following equation:

$$(l_0 + l_1\lambda_\varphi + l_2 k)P = l_0 P + l_1\varphi(P) + l_2 R = \infty,$$

assuming the equality $kP = R$ is true. To find an appropriate triple (l_0, l_1, l_2), we follow the GLS method and consider the lattice generated by the vectors $(k, 0, -1), (\lambda_\varphi, -1, 0)$, and $(n, 0, 0)$. Assuming k is a large random scalar, which is the case for signature schemes specified above, we should expect a short vector (l_0, l_1, l_2) in the lattice of norm $\approx n^{1/3}$.

In Section 5.3.1, we select an appropriate lattice reduction technique for finding the short vector (l_0, l_1, l_2). Section 5.3.2 and Section 5.3.3 apply GLV-3 to VSM and VSS in full detail and present experimental results. Section 5.3.4 presents how the technique can be easily modified for batch signatures.

Remark 5. All of our algorithms were implemented in the Mbed TLS library [77] and tested on a 64-bit AMD x86 processor and on a 32-bit ARM Cortex-M4 STM32F407 microcontroller (MC). We ran each experiment 1000 times on the AMD processor and 10 times on the MC [3]. We used elliptic curves over three different bit-lengths: 256, 384, and 521. For the 256-bit case, we used the popular secp256k1 curve $y = x^3 + 7$ over $\mathbb{F}_p$ with $p = 2^{256} - 2^{32} - 977$. For the remaining two bit-lengths, we generated two curves of the same form $y^2 = x^3 + 7$ as secp256k1 for easier implementation. The base field $\mathbb{F}_p$ is of the size $p = 2^{384} - 170937$ and $p = 2^{521} - 503629$, respectively (the bit-length 521 was chosen based on the NIST P-521 curve). Finally, the experiments for batch verification were only simulated in Python, and the cost was approximated based on the previous experiments on both platforms. We have to emphasize that the goal of the experiments is not to break the records for signature verification on the platforms. Rather, our goal is a fair comparison between individual algorithms and so we are interested in the relative performance.

[3] This is a reasonable number since, for the MC, we measure execution time in clock cycles. We expect to obtain always the same number since our algorithms are not randomized.

5.3.1 Reducing 3-dimensional lattices

This part describes our choice of a lattice reduction algorithm for the lattice given by the vectors

$$(k, 0, -1), (\lambda_\varphi, -1, 0), (n, 0, 0).$$

Recall that the classical 2-dimensional GLV (or 4-dimensional GLS) decomposition precomputes the reduced basis (e.g., using the LLL algorithm) once and then uses it to decompose each scalar efficiently. The situation with the method by Antipa et al. [6] is slightly more complicated since the lattice is defined by the given scalar, and the reduction cannot be precomputed. However, the dimension is only 2, and the efficient EEA can be used, which makes the cost of the reduction negligible when compared to the scalar multiplication. Our GLV-3 method inherits the worst of both worlds as we have a 3-dimensional lattice defined per each scalar. Our goal is to accelerate signature verification, and so the choice of the reduction algorithm is crucial.

Our strategy is to follow the work by Semaev [189] on the reduction of 3-dimensional lattices. However, rather than selecting the lattice reduction algorithm with the best asymptotic complexity that always finds a good basis, we take the experimental approach. We select the algorithm that experimentally worked best for the typical sizes of input values in our context.

Algorithm 4: Lagrange lattice reduction.

Input: Basis b_1, b_2 of a lattice $\mathcal{L} \subset \mathbb{R}^2$
Output: Basis b_1, b_2 of $\mathcal{L}$ satisfying $|b_1| \leq |b_2|$, $2|\langle b_1, b_2 \rangle| < |b_1|^2$

do
  if $|b_1| > |b_2|$ **then** swap b_1, b_2
  $m \leftarrow \lfloor \langle b_1, b_2 \rangle / |b_1|^2 \rceil$
  $b_2 \leftarrow b_2 - m b_1$
while $m \neq 0$
return b_1, b_2

Semaev [189] builds up on the basic reduction algorithm for a general 2-dimensional lattice — the Lagrange algorithm (Algorithm 4). The idea of Lagrange algorithm for given vectors b_1, b_2 is to iteratively reduce the bigger vector, say b_2, by subtracting a multiple of the other vector, i.e.,

replace b_2 with $b_2 - mb_1$. The optimal choice for m is to approximate the orthogonal projection of b_2 to the orthogonal complement of b_1: $m = \lfloor \langle b_1, b_2 \rangle / |b_1|^2 \rceil$. Here, $|\cdot|$ is the Euclidean norm and $\langle \cdot, \cdot \rangle$ is the scalar product. The output of the algorithm is a *reduced pair* b_1', b_2' satisfying $|b_1'| \leq |b_2'|$, $2|\langle b_1', b_2' \rangle| < |b_1'|^2$, which we denote $(b_1', b_2') = G(b_1, b_2)$ as in [189]. The reduced pair forms the so-called Minkowski reduced basis and always contains the shortest vector in the lattice. Note that the costly division in Step 3 in Algorithm 4 can be computed with low precision since the result is then rounded to the closest integer. As a further optimization, the scalar products $\langle b_i, b_j \rangle$ are computed once at the beginning and stored in the Gram matrix. With each replacement in Step 4, we only update the Gram matrix, store m, and reconstruct the final vectors at the end.

Algorithm 5: Lagrange 3-dimensional lattice reduction.

Input: Basis b_1, b_2, b_3 of a lattice $\mathcal{L} \subset \mathbb{R}^3$
Output: A 2-reduced basis b_1, b_2, b_3 of $\mathcal{L}$

```
1 while True do
2     2-reduced ← True
3     foreach i, j ∈ {1, 2, 3} do
4         if b_i, b_j not reduced then
5             b_i, b_j ← G(b_i, b_j)                    // Using Algorithm 4
6             2-reduced ← False
7     if 2-reduced then return b_1, b_2
```

As Semaev [189] further shows, Algorithm 4 can be naively extended to 3 dimensions by iteratively applying Algorithm 4 to every pair until we get a basis in which each pair is reduced (Algorithm 5). We call such basis *2-reduced*. As Semaev explains, one can easily construct basis vectors that are 2-reduced but do not contain the shortest vector, and thus the algorithm is not guaranteed to find it. Luckily, this can be solved by appending one quick computation at the end of the algorithm (see Algorithm 2 in [189]). The algorithm then outputs the Minkowski reduced basis containing the shortest vector. Regarding asymptotic complexity, this can take as many $O(\log^3 M)$ operations to find the 2-reduced basis, where $|b_1|, |b_2|, |b_3| \leq M$. As an example supporting this lower bound on the complexity, Semaev presents a specially constructed

lattice with vectors $(1,1,0),(M,0,0),(0,M^2,1)$ with significantly different sizes. However, as we will see, this is not the typical case.

Finally, Semaev presents an algorithm that finds the Minkowski reduced basis with quadratic asymptotic complexity (Algorithm 6). The idea is to minimize the quadratic form $|b_3 - x_1 b_1 - x_2 b_2|$ in x_1, x_2, where $|b_1| \leq |b_2| \leq |b_3|$ and b_1, b_2 is a reduced pair. The optimal values for x_1, x_2 are found in real numbers, and the four possible combinations of the nearest integers are then checked. This is done repeatedly until the biggest vector cannot be reduced.

Algorithm 6: Semaev lattice reduction.

Input: Basis b_1, b_2, b_3 of a lattice $\mathcal{L} \subset \mathbb{R}^3$, s.t. $|b_1| \leq |b_2| \leq |b_3|$
Output: A reduced basis b_1, b_2, b_3 of $\mathcal{L}$

while True **do**
  $b_1, b_2 \leftarrow G(b_1, b_2)$
  $D \leftarrow 1 - \frac{\langle b_1, b_2 \rangle}{|b_1|^2} \cdot \frac{\langle b_1, b_2 \rangle}{|b_2|^2}$
  $y_1, y_2 \leftarrow -\frac{1}{D}\left(\frac{\langle b_1, b_3 \rangle}{|b_1|^2} - \frac{\langle b_1, b_2 \rangle}{|b_1|^2} \cdot \frac{\langle b_2, b_3 \rangle}{|b_2|^2}\right), -\frac{1}{D}\left(\frac{\langle b_2, b_3 \rangle}{|b_2|^2} - \frac{\langle b_1, b_2 \rangle}{|b_2|^2} \cdot \frac{\langle b_1, b_3 \rangle}{|b_1|^2}\right)$
  Select integers $x_1 \in [y_1 - 1, y_1 + 1], x_2 \in [y_2 - 1, y_2 + 1]$ such that $|b_3 + x_2 b_2 + x_1 b_1|$ is minimal
  $b_3' \leftarrow b_3 + x_2 b_2 + x_1 b_1$
  if $|b_3'| \geq |b_3|$ **then return** b_1, b_2, b_3
  Replace b_3 by b_3' and order b_1, b_2, b_3 by their norms.

Table 5.2 presents our experimental results of Lagrange 3-dimensional algorithm and Semaev algorithm (Algorithm 6). The column 'Bits' shows the average bit-length of the shortest found vector norm. While Lagrange algorithm is theoretically not guaranteed to find the shortest vector, it found it every time except for 4 out of 3000 cases (across the three bit-lengths). In these 4 cases, the sizes of the norms of the shortest found vectors for Semaev and Lagrange algorithms differed by at most one bit. Surprisingly, Lagrange algorithm was 45% faster than Semaev on the AMD. It was also much faster on the microcontroller, though it depended on the bit-length: $35\%, 45\%, 21\%$ for the bit-lengths $256, 384, 521$, respectively. This is in contrast with the theoretical comparison of the asymptotical complexities. As a result, we select Lagrange algorithm for lattice reductions in the following sections.

Table 5.2: Comparison of implementations of Lagrange 3-dimensional Algorithm and Semaev Algorithm on a 64-bit AMD x86 processor and on a 32-bit ARM Cortex-M4 STM32F407 microcontroller. The columns show the asymptotic complexity, bit-lengths of inputs k and n, average bit-length of the shortest vector norm, the runtime in milliseconds (AMD), and cycle count on the MC.

	Complexity	$\ell(n)$	Avg. bits	AMD (ms)	MC (cycles)
Lagrange	$\Omega(\log^3 n)$	256	84.813	0.062	937211
		384	127.593	0.097	1738715
		521	173.215	0.139	2782500
Semaev	$O(\log^2 n)$	256	84.811	0.110	1449521
		384	127.593	0.176	3077910
		521	173.214	0.257	3538717

Remark 6. Experimentally, we have found that in Algorithm 4, it is enough to perform the division on only the most significant byte of the numerator $\langle b_1, b_2\rangle$ and denominator $|b_1|^2$. Similarly, in Algorithm 6, we have found that y_1, y_2 can be efficiently computed as

$$y_1 = -\frac{\langle b_1, b_3\rangle - \langle b_1, b_2\rangle \cdot \langle b_2, b_3\rangle}{|b_1||b_2|^2 - \langle b_1, b_2\rangle^2}, \quad y_2 = -\frac{\langle b_2, b_3\rangle - \langle b_1, b_2\rangle \cdot \langle b_1, b_3\rangle}{|b_1||b_2|^2 - \langle b_1, b_2\rangle^2},$$

where only the two most significant bytes of each term $\langle b_i, b_j\rangle$, or $|b_i|^2$ are considered. This differs from the original approach in [190], which we found to be slightly slower for the selected bit-lengths. Other approximations could be used as in [166] but this might depend on the platform. Furthermore, since the second and third vector of the initial basis $(k, 0, -1), (\lambda_\varphi, -1, 0), (n, 0, 0)$ are fixed for the given curve, we compute the reduction $b_1, b_2 \leftarrow G((\lambda_\varphi, -1, 0), (n, 0, 0))$ using Algorithm 4 once and store u_1, u_2. Finally, at the beginning we also decompose k into k_0, k_1 (Section 2.6.1) and replace $(k, 0, -1)$ with $u_3 = (k_0, k_1, -1)$. The input vectors to both algorithms are the three vectors u_1, u_2, and u_3.

5.3.2 Scalar multiplication verification with GLV-3

We assume that the curve admits an endomorphism φ with the cost of its application equal to one finite field multiplication. This section explains how our GLV-3 can be used to solve VSM faster than the previous methods (Section 5.2.2 and Section 5.2.3). As explained at the

beginning of this section, we can transform the equation $kP = R$ (VSM) to $l_0P + l_1\varphi(P) + l_2R = \infty$, where $l_i \approx n^{1/3}$ are found using the Lagrange 3-dimensional algorithm.

To verify $l_0P + l_1\varphi(P) + l_2R = \infty$, we compute the left-hand side using the classical Straus-Shamir trick with JSF for three scalars. We iterate through the signed bits of l_0, l_1, l_2 and add one of the 26 points $j_0P + j_1\varphi(P) + j_2R$, $j_i \in \{\pm 1, 0\}$, which are precomputed at the beginning. If we leverage the fast sign switch of points, it is enough to precompute 13 points (including the input points) with the cost of $9A + 2M$. The cost of the Straus-Shamir trick is then given by the number of additions and doublings. The number of doublings is equal to the maximum of the bit-lengths of l_0, l_1, l_2, which is $\approx \frac{\ell(n)}{3}$. The number of additions is given by the compound Hamming weight of l_0, l_1, l_2 which is $\approx 0.59 \cdot \frac{\ell(n)}{3} \approx \frac{\ell(n)}{5}$ in the JSF form. The overall cost is

$$\left(9 + \frac{\ell(n)}{5}\right)A + \frac{\ell(n)}{3}D + 2M \approx (4.53\ell(n) + 101)M.$$

Compared to the GLV method with the cost $(6.25\ell(n) + 13)M$, this is a $\approx$ 22–25% speed-up for the typical values of $\ell(n)$ (between 256 and 521).

We have to emphasize that this 25% speed-up only considers the complexity of the Straus-Shamir trick in terms of additions and doublings, and the actual speed-up will be lower. The computation usually also involves, for example, a transformation of the precomputed points into an affine form. More importantly, we did not include the cost of Lagrange algorithm to find l_0, l_1, l_2, which is a significant burden that the GLV does not have. Rather than theoretically assessing the complexities of the GLV and our GLV-3, we experimentally compare them. Table 5.3 shows the comparison for the common curve bit-lengths and the timing on our two platforms. For 384 and 521-bit curves, we measured the speed-up to be 23–24% on the AMD and 25–26% on the MC. For the 256-bit curves, it is only 5% on the AMD and 13% on the MC. This is mainly because of Lagrange algorithm, whose cost grows with the bit-length (see Table 5.2), but not as rapidly as with the whole scalar multiplication.

Remark 7. Note that a similar idea for verifying scalar multiplication was recently introduced in the context of the SNARK protocol [69]. While in this work the scalar decomposition is part of the running time, the verification in the SNARK circuit works with already precomputed

Table 5.3: Comparison of the GLV method and our GLV-3 for solving VSM. The first column shows the theoretical cost in the number of point additions, doublings, and field multiplications. The third column contains equivalent cost purely in the number of field multiplications for common bit-lengths. The last two columns show the speed in milliseconds (on a 64-bit AMD x86) and the cycle count (ARM Cortex-M4 MC).

	Cost	t	Mult. (M)	AMD (ms)	MC (cycles)
GLV	$\left(\frac{\ell(n)}{4}+1\right)A+\frac{t}{2}D+2M$	256	1613	0.50	12 668 727
		384	2413	4.6	226 492 679
		521	3269	10.67	539 123 262
GLV-3	$\left(9+\frac{\ell(n)}{5}\right)A+\frac{t}{3}D+2M$	256	1261	0.48	11 048 969
		384	1841	3.56	170 030 994
		521	2461	8.13	400 341 296

decomposition. Moreover, the authors use the half extended Euclidean algorithm for the scalar decomposition.

5.3.3 Signature verification with GLV-3

This section applies GLV-3 to the verification of signatures (VSS). Let $uG+vQ=R$ be the verification equation. Using Lagrange algorithm, we can find scalars $l_0, l_1, l_2 \approx n^{1/3}$ such that $l_0+l_1\lambda_\varphi+l_2v=0 \pmod n$. We multiply the verification equation by l_2, and after the substitution, we get $l_2uG-l_0Q-l_1\varphi(Q)-l_2R=\infty$. Now we split the scalar $l_2u \pmod n$ into three scalars of a third of the bit-length, i.e., $l_2u=u_0+2^{\ell(n)/3}u_1+2^{2\ell(n)/3}u_2 \pmod n$. Denote $G_1=2^{\ell(n)/3}G$, $G_2=2^{2\ell(n)/3}G$. Putting this all together, we get:

$$u_0G_0+u_1G_1+u_2G_2-l_0Q-l_1\varphi(Q)-l_2R=\infty.$$

All of the scalars have $\approx \ell(n)/3$ bits. To evaluate the left-hand side, we run two simultaneous Straus-Shamir tricks for u_0, u_1, u_2 and for l_0, l_1, l_2, both with JSF (also described by Algorithm 7). More precisely, at the beginning, we precompute two sets of points: $h_0G_0+h_1G_1+h_2G_2$ and $j_0Q+j_1\varphi(Q)+j_2R$, $h_i, j_i \in \{\pm1, 0\}$, where we exclude points with opposite signs. This amounts to 26 points computed with the cost $19A+2M$. During the Straus-Shamir tricks, we iterate through the $\ell(n)/3$ bits of the scalars u_i, l_i and make two additions, one for each

Table 5.4: Comparison of the GLV method and our GLV-3 for solving VSS. The first column shows the theoretical cost in the number of point additions, doublings, and field multiplications. The third column contains equivalent cost purely in the number of field multiplications for common bit-lengths. The last two columns show the speed in milliseconds (on a 64-bit AMD x86) and the cycle count (ARM Cortex-M4 MC).

	Cost	t	Mult. (M)	AMD (ms)	MC (cycles)
GLV	$\left(\frac{\ell(n)}{2}+2\right)A+\frac{\ell(n)}{2}D+4M$	256	2330	0.81	19 341 897
		384	3482	7.13	328 380 849
		521	4715	15.72	772 028 416
GLV-3	$\frac{\ell(n)}{3}D+\left(19+\frac{2\ell(n)}{5}\right)A+2M$	256	1934	0.82	18 902 318
		384	2795	5.79	268 629 338
		521	3717	12.80	615 912 490

precomputed set. Since the compound Hamming weight of all the scalars is $2 \cdot 0.59 \cdot \frac{\ell(n)}{3} \approx \frac{2\ell(n)}{5}$, the total cost is

$$\left(19+\frac{2\ell(n)}{5}\right)A+\frac{\ell(n)}{3}D+2M \approx (6.73\ell(n)+211)M.$$

Compared to the GLV method with the cost $(9\ell(n)+26)M$, this is a 17–21% speed-up for t between 256 and 521 bits. Table 5.4 presents the experiment results. For 256 bits, the overhead of Lagrange algorithm and the precomputation table is too large, and GLV-3 is only 2% faster than GLV on the MC and even 2% worse on the AMD. However, for 384 and 521 bits, the speed-up is 18–20% on both platforms.

5.3.4 Batch verification

Previous sections discussed accelerations of the verification of a single signature. Systems often need to verify a *batch* of signatures, so we will now modify the GLV-3 method for batch verification. The signatures in a batch might all correspond to one public key Q or, more generally, to several Q_i. We will first assume that the public key is fixed and focus on the general case afterward. The usual approach for batch signatures, proposed in [141] and since then adopted for EdDSA [21] and Schnorr signatures [218], is to verify a random linear combination of the verification

Algorithm 7: Signature verification using GLV-3.

Input: $u, v \in \mathbb{Z}$, points G, Q, R, $G_1 = 2^{\ell(n)/3}G$, $G_2 = 2^{2\ell(n)/3}G$ on a curve with order n and endomorphism φ

Output: $uG + vQ \stackrel{?}{=} R$

$l_0, l_1, l_2 \leftarrow$ Algorithm 5 on $(v, 0, -1), (\lambda_\varphi, -1, 0), (n, 0, 0)$

Find u_0, u_1, u_2, s.t. $ul_2 \pmod n = u_2|u_1|u_0$, where $|$ is concatenation of bits.

Compute $h_0G + h_1G_1 + h_2G_2$ and $j_0Q + j_1\varphi(Q) + j_2R$ for $j_i, h_i \in \{\pm 1, 0\}$

$T \leftarrow \infty$

for $(h_0, h_1, h_2), (j_0, j_1, j_2) \in \mathrm{JSF}(u_0, u_1, u_2), \mathrm{JSF}(l_0, l_1, l_2)$ **do**

  $T \leftarrow 2T$

  $T \leftarrow T + h_0G + h_1G_1 + h_2G_2$

  $T \leftarrow T - j_0Q - j_1\varphi(Q) - j_2R$

return $T \stackrel{?}{=} \infty$

equations. Let $u_iG + v_iQ = R_i$ be the set of B equations. The idea is to multiply each equation by a random integer z_i and verify their sum:

$$\left(\sum_{i=1}^{B} z_i u_i\right)G + \left(\sum_{i=1}^{B} z_i v_i\right)Q - \sum_{i=1}^{B} z_i R_i = \infty.$$

The left side, containing $B + 2$ scalars and points, is then computed and checked with the neutral point on the right side. One of the state-of-the-art methods for multi-scalar multiplication, also used for batch signatures in [21], is the Bos-Coster method [67]. To compute the general expression $k_1P_1 + k_2P_2 + \cdots + k_sP_s$, where $k_1 \geq k_2 \geq \cdots \geq k_s$, the Bos-Coster method recursively computes $(k_1 - k_2)P_1 + k_2(P_1 + P_2) + k_3P_3 \cdots + k_sP_s$.

The complexity of the Bost-Coster method is determined by the number of iterations, which equals the number of point additions. In general, it is difficult to find a closed formula for the number of iterations. Hence, we simulated the execution for the usually considered batch of size 64 signatures. The average number of iterations turned out to be 3352, 5012, and 6788 for scalars of bit-length 256, 384, and 521, respectively.

The GLV method can be used to decompose each of the $B + 2$ scalars, resulting in twice the amount $(2B + 4)$ of scalars, but half the size. For a batch of size 64, the average number of iterations is then reduced to 2958,

4400, and 5925 for the individual bit-lengths. This results in roughly 12% speed-up when compared to simple Bos-Coster for all three bit-lengths.

Our GLV-3 method can be adapted to batch signatures as well. For each equation $z_i u_i G + z_i v_i Q = z_i R_i$, we find $l_{i,0}, l_{i,1}, l_{i,2}$ such that $l_{i,0} + l_{i,1}\lambda_\varphi + l_{i,2} z_i = 0 \pmod{n}$ using Lagrange algorithm. Substituting into the verification equation, we get

$$l_{i,2} z_i u_i G + l_{i,2} z_i v_i Q + l_{i,0} R_i + l_{i,1} \varphi(R_i) = \infty,$$

with the following sum to be verified:

$$\left(\sum_{i=1}^{B} l_{i,2} z_i u_i\right) G + \left(\sum_{i=1}^{B} l_{i,2} z_i v_i\right) Q + \sum_{i=1}^{B} l_{i,0} R_i + \sum_{i=1}^{B} l_{i,1} \varphi(R_i) = 0.$$

The scalar corresponding to G can be further split into three parts as in the previous sections assuming that we have two precomputed multiples of G. We can do the same trick for the point Q and the corresponding scalar, but here we have to compute the multiples of Q for each batch corresponding to Q, and so the cost of this must be included as well. The left side contains $2B + 6$ scalars of size similar to $n^{1/3}$.

For a batch of size 64, we found that the average number of additions (including the precomputation for Q) is then reduced to 2192, 3260, and 4376 for the individual bit-lengths. We can use the measurements on both platforms to compare these batch versions of GLV-3 and GLV. If we multiply the number of iterations with the average measured cost of point addition, we get the approximate cost of GLV in Table 5.5 ('Fixed key' columns). The cost of GLV-3 can be approximated in the same way with the addition of the cost of the lattice reduction measured in Table 5.2. The speed-up of GLV-3 for batch signatures is similar to the single-signature case. For 256 bits, the speed-up is negligible on the MC (2%) and even negative on AMD, but for 384 and 521 bits, it reaches 20–23% on both platforms.

Now, we will remove the assumption of a fixed public key Q. Consider a public key Q_i for every verification equation. The linear combination of the equations is then

$$\left(\sum_{i=1}^{B} z_i u_i\right) G + \sum_{i=1}^{B} z_i v_i Q_i - \sum_{i=1}^{B} z_i R_i = \infty.$$

Table 5.5: The cost of the GLV method and our 3-dimensional GLV for batch verification on two platforms (64-bit AMD x86 and 32-bit ARM Cortex-M4 MC) for three bit-lengths. The time and the clock cycles are approximated based on the number of iterations and the measured cost of individual operations. Two cases are considered: all signatures in the batch share the same public key and the general case of multiple keys.

		Time on AMD (ms)		Cycles on MC	
	$\ell(n)$	Fixed key	Multiple keys	Fixed key	Multiple keys
GLV	256	0.22	0.38	3 919 026	6 666 850
	384	2.03	3.49	39 809 138	67 946 960
	521	4.71	8.10	92 734 489	159 409 414
GLV-3	256	0.23	0.39	3 841 371	6 697 835
	384	1.61	3.10	31 233 667	60 285 290
	521	3.62	7.08	71 272 985	139 403 489

Experiments showed that Bos-Coster takes approximately 5705, 8532, and 11556 iterations to compute the left side with $2B + 1$ scalars for individual bit-lengths. The GLV method doubles the number of scalars (with half the size), resulting in 5108, 7588, and 10229 iterations, i.e., 10–11% speed-up.

Applying the GLV-3 method is not as straightforward as with the fixed public key. Using the same substitution with $l_{i,0} + l_{i,1}\lambda_\varphi + l_{i,2}z_i = 0 \pmod{n}$, we get

$$\left(\sum_{i=1}^{B} l_{i,2}z_iu_i\right)G + \sum_{i=1}^{B} l_{i,2}z_iv_iQ_i + \sum_{i=1}^{B} l_{i,0}R_i + \sum_{i=1}^{B} l_{i,1}\varphi(R_i) = \infty,$$

where $l_{i,0}$ and $l_{i,1}$ are small scalars ($\approx n^{1/3}$), but $l_{i,2}z_iv_i \approx n$. We can, however, decompose each $l_{i,2}z_iv_i$ using λ_φ to two scalars of half the size ($\approx n^{1/2}$). As a result, we have $2B + 3$ scalars of similar size as $n^{1/3}$ (corresponding to $R_i, \varphi(R_i)$ and G) and $2B$ scalars of similar size as $n^{1/2}$ (corresponding to Q_i). Because of this gap between the two sets of the scalars, in one iteration of the Bos-Coster, the largest scalar $k_1 \approx n^{1/2}$ can be much bigger than the second largest $k_2 \approx n^{1/3}$. Hence, a simple scalar multiplication by k_2 might be needed to reduce k_1. Otherwise, the technique works the same as with scalars of the same size.

Table 5.5 (columns 'Multiple keys') summarizes the cost approximated in the same way as for the case with fixed Q. On both platforms, there is no speed-up for 256 bits and 11–13% speed-up for the higher bit-lengths.

5.3.5 4-dimensional GLV

The GLS method described in Section 2.6.1 uses two endomorphisms for the 4-dimensional decomposition. Our GLV-3 method also uses two endomorphisms. This section describes why we did not focus on the 4-dimensional alternative.

For the VSM equality $kQ = R$ on a curve with endomorphism φ, we have considered k as an endomorphism for which kQ has been computed for us. Similarly, we can consider $k\varphi(Q) = \varphi(R)$ and find $l_0, l_1, l_2, l_3 \approx n^{1/4}$ satisfying:

$$\begin{aligned} l_0 + l_l\lambda_\varphi + l_2 k + l_3 k\lambda\varphi = 0 \pmod n \\ l_0 Q + l_1\varphi(Q) + l_2 R + l_3\varphi(R) = \infty \end{aligned}$$

with the assumption that $kQ = R$. In principle, this gives us a verification equation[4] with smaller scalars than GLV-3. Using two Straus-Shamir tricks with two pairs of scalars l_0, l_1 and l_2, l_3 in the JSF form, we can compute the left side with the cost $(\frac{\ell(n)}{4} + 2)A + \frac{\ell(n)}{4}D + 4M = (4.5\ell(n) + 26)M$. Compared to the GLV-3 method, this brings only 4–7% speed-up. We can also precompute all possible combinations $j_0 Q + j_1\varphi(Q) + j_2 R + j_3\varphi(R)$, $j_i \in \{0, \pm 1\}$ and consider all four scalars in the JSF form with the join Hamming weight $0.64 \cdot \frac{\ell(n)}{4} = \frac{4\ell(n)}{25}$. The precomputation then costs $34A + 4M$ and the overall cost is $\approx (3.51\ell(n) + 378)M$, which is actually worse than GLV-3 for $\ell(n) = 256$ and 10% better for $\ell(n) = 521$.

The main issue of this 4-dimensional approach is the search for the short vector (l_0, l_1, l_2, l_3) in a 4-dimensional lattice. Our experiments showed that Lagrange algorithm adapted to 4 dimensions very often does not find a short vector of norm $\approx n^{1/4}$. Other algorithms could be used, e.g., [148], but we leave this as a future work. We could also apply this idea to VSS, resulting in eight scalars. Based on the observations with VSM, we decided that the overhead of the precomputation and the lattice reduction is too troublesome and most likely not worth the speed-up.

[4] It should be noted that if the second equation holds, then $kQ = R$ is true (and gets correctly verified) as long as $l_2 + l_3\lambda_\varphi \neq 0 \pmod n$, but this is expected.

5.4 The dependent coordinates problem

The GLV-3 method uses a fast endomorphism and a precomputed equality $kP = R$ to decompose the scalar k into three smaller scalars. Smaller scalars imply faster scalar multiplication, which increases the presented speed. Beyond fast scalar multiplication, this decomposition also finds another application. In this section we return back to the field of side-channel attacks from Chapter 4. We show that the GLV-3 technique can be used to significantly reduce the complexity of the ZVP attack.

Recall that during the ZVP attack (Section 2.5), the attacker needs to solve several instances of the DCP. To solve $\mathrm{DCP}_f(k)$ means to find two points P, Q such that $Q = kP$ and $f(x_P, x_Q) = 0$. Finding such a solution efficiently is an problem and the best approach is to just guess the point P with the complexity $O(n)$.

In Chapter 4, we realized that an implementation that uses the GLV method for scalar multiplication exposes more bits of the private. The DCP is, in that case, defined for $k = k_0 + k_1\lambda_\varphi$, where k_0, k_1 are small for the first several iterations. This allows the attacker to target both k_0 and k_1 at the same time, recovering almost twice as many bits as the original ZVP. While this result increases the number of recovered bits, the asymptotic complexity of the general DCP remains to be $O(n)$. The method uses multiplication-by-k_i maps of degree k_i^2. Since for general k we should expect $k_i^2 \approx n$, the complexity is at best $O(n)$.

We will now show how the 4-dimensional GLV method described in Section 5.3.5 can be used to reduce the asymptotic complexity of DCP on a GLV curve to $O(\sqrt{n}\log^3 n)$. Let Q, P be the DCP solution points and $l_0, l_1, l_2, l_3 \approx n^{1/4}$ such that

$$l_0 + l_1\lambda_\varphi + l_2 k + l_3\lambda_\varphi k = 0 \pmod{n}.$$

These can be found using, e.g., the LLL algorithm. The idea is to follow the symbolical approach described in Section 2.5.1, but instead of using a multiplication-by-k map of degree n^2, we use the multiplication-by-l_i maps of degree $\sqrt{n}$. Specifically, as in Chapter 4, we use the Semaev polynomial [190] that satisfies $S_4(x_{P_1}, x_{P_2}, x_{P_3}, x_{P_4}) = 0$ for any $P_1 \pm P_2 \pm P_3 \pm P_4 = \infty$. In our case with $l_0P + l_1\varphi(P) + l_2Q + l_3\varphi(Q) = \infty$, we get:

$$S(x_P, x_Q) = S_4(r_0(x_P), r_1(x_{\varphi(P)}), r_2(x_Q), r_3(x_{\varphi(Q)})) = 0,$$

where r_0, r_1, r_2, r_3 are the x-coordinates of the multiplication maps for l_0, l_1, l_2, l_3, respectively. The degree of S is given by $\deg S_4 = 16$ and by the degrees of the r_i maps, which is $\approx \sqrt{n}$. Together with $f(x_P, x_Q) = 0$, we have two polynomial equations with two unknowns x_P, x_Q. It remains to compute the resultant, a univariate polynomial in x_P, and find its roots. Assuming fast integer arithmetic, this can be done in $O(\sqrt{n}\log^3 n)$ operations — a significant reduction from the original $O(n)$ complexity.

5.5 Conclusions and future work

Our work shows that endomorphisms that have been used so far for 2-dimensional decomposition can also be used for higher dimensions in the right context. This improves the verification of Schnorr-like signature schemes. Specifically, experiments demonstrate a $\approx 20\%$ speed-up for single signature verification and 13% speed-up for batch verification. The improvement is mainly relevant for curves with higher bit-lengths with endomorphisms, including the classical pairing-friendly curves, such as BLS12-381 [37]. In addition, the technique applies to any protocol involving the verification of scalar multiplication, such as the Blind Diffie-Hellman protocol, where we show 27% speed up.

The three dimensions in our GLV-3 technique came with the price of reducing 3-dimensional lattices. We analyzed 3-dimensional lattice reduction algorithms and concluded that the simple adaptation of Lagrange algorithm for 3 dimensions is efficient enough and even faster than Semaev algorithm with the state-of-the-art complexity.

Finally, to demonstrate that the idea of the GLV-3 extends beyond a signature verification, we lower the asymptotic complexity of the hard DCP that is hindering the so-called Zero-value point attack.

This work considered Schnorr signatures, modified ECDSA signatures, and the BDHKE protocol. Other schemes, not necessarily signatures, could also benefit from the GLV-3 method. There are only two requirements for the method: 1) the curve must admit an efficient endomorphism, and 2) scalar multiplication is verified in the scheme. As discussed in Remark 4, the EdDSA scheme over an Edwards curve with complete formulas could take advantage of the method as long as an appropriate curve with an efficient endomorphism would be found.

6 Reverse-engineering ECC using side-channel attacks

This chapter is based on the following publication:

[VS1] J. Jancar, V. Suchanek, P. Svenda, V. Sedlacek, and Ł. Chmielewski. "pyecsca: Reverse engineering black-box elliptic curve cryptography via side-channel analysis". *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024:4, 2024, pp. 355–381. DOI: 10.46586/TCHES.V2024.I4.355-381

The publication itself is the outcome of a long-term research effort that originally grew out of the Master's thesis of Jancar [104]. As my contribution did not extend to all parts of the publication, this chapter focuses exclusively on the components to which I contributed substantially, and omits material beyond the scope of my involvement and this thesis.

6.1 Introduction

While ECC [133, 114] 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 [136] with homogenous projective coordinates [194], some addition formulas and a simple double-and-add-always scalar multiplier, while another implementation might pick the Barrett reduction with Jacobian coordinates [53] and a different combination of addition formulas and scalar multiplier. These two implementations, while significantly 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 [72, 73, 64]. Most of these require an attacker to have detailed information about the target implementation. For example, [17] 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 [64] 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, or cryptocurrency wallets. The popular JavaCard platform [157] is a clear example of this — it offers rich cryptographic APIs, including ECC [202]. 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 [199], a widely recognized international standard for evaluating and certifying the security of IT products, uses the Joint Interpretation Library attack rating [183] that encourages keeping knowledge about the implementation as secretive as possible to increase attack time. Open-sourcing a certified implementation would result in potential attacks being considered easier to execute and 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 [111] and partly bases its security on obscurity, inviting catastrophic issues. When obscurity dissipates, for example, by an expert analysis or an information leak, security might be gone as well, as in the cases of the Infineon RSA library (ROCA [146]), Athena IDProtect smartcard (Minerva [105]) or NXP P5x secure elements (Side Journey to Titan [173]). This argument is especially clear in the case of the RSA library [146]: a flawed key generation algorithm allowed efficient

factorization of 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 in 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 high 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 6.2 we present related work on side-channel-based reverse-engineering. Variety of implementation choices in open-source ECC libraries is documented in Section 6.3. We present automated techniques for reverse-engineering of black-box ECC implementations in Section 6.4. Then we showcase the functionality of our toolkit in Section 6.5. In Section 6.6 we evaluate the reverse-engineering techniques, and finally conclude in Section 6.7.

[1]Pronounced [pɪɛtska]. Available at pyecsca.org.

6.2 Reverse-engineering

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

Side-channel-based disassembly

Quisquater et al. [168] 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. [209] 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 on Bayesian classifier targeting the PIC microcontroller [71], a disassembler based on k-Nearest Neighbor (kNN) classifier targeting the ATMega163 microcontroller [139], and a disassembler based on hierarchical classification setup and principal component analysis targeting ARM Cortex-M3.

A more complex architecture, system-on-chip based on ARM Cortex-A9, was targeted by Maillard et al. [125]. 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. [201] and Iyer et al. [101] 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, some other works concentrated on also recovering

operands: a disassembler that classifies the bit encoding of instructions of the PIC16F microcontroller [63] and a disassembler targeting ATMega328P via hierarchical classification [159].

Slightly different information was recovered by Gao et al. [83]: the presented disassembler 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 [143] and ATMega328P [9], 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 [85, 12].

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 [56], who recovered the substitution tables of the proprietary GSM A3/A8 ciphers, which were secret at the time. Daudigny et al. [65] 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 [169], LFSRs and generic non-linear functions [90], and S-boxes in typical structures such as Substitution-Permutation Network (SPN), Feistel, and eXtended Feistel [203]. While previous techniques recovered only some secret part of a symmetric cipher and assumed the rest as known, [171] only assumed that the SPN is used. They presented a technique for recovery of all parts of the design, also with a more realistic noisy leakage model.

Active side-channel attacks, such as fault injection, have also been used for RE. Pedro et al. [160] presented a technique called FIRE (Fault Injection for Reverse Engineering) targeting an unknown S-box in a known cipher. Fault injection, together with passive SCA, was used by Clavier et al. [57] to recover information about an AES-like cipher, even with some SCA countermeasures implemented and without knowledge of the used key.

Amiel et al. [3] 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. [173] performed SCA against a black-box smartcard ECDSA implementation. To do that, they first manually reverse-engineered the used scalar multiplier before developing an attack.

6.3 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 SymCrypt. Our analysis was restricted to code implementing ECDH, ECDSA on prime-field curves and X25519, Ed25519. Understanding the diversity observed in open-source libraries provides insights about 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.

6.3.1 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 multiplier (e.g., precomputed multiples of the

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

generator). As a result, it is not enough to only consider specific library, but also specific hardware platform and curve on which the library is executed on.

6.3.2 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 to birationally equivalent curves in a different curve model. This is supported by our analysis that shows 4 out of 13 implementations of X25519 using 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` [19], a tool for generating ECC implementations that picks a Twisted Edwards model whenever able, even for Short-Weierstrass curves.

6.3.3 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 [95, Alg. 3.44], fixed-window method [95, Alg. 3.41], simple ladder method [137] and double-and-add-always [61]. 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 [35, 123] 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 [82] and wNAF [95] methods.

In the Montgomery model, the fixed base multiplier was either the Montgomery ladder [137] 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 [162], comb methods, fixed-window methods and double-and-add-always.

6.3.4 Coordinate systems

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

The Montgomery model implementations all used xz coordinates, 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 use projective coordinates. For an overview of Ed25519 implementations, see [84]. For the rest of this work, we will focus on single coordinate systems, leaving aside these complicated implementations.

6.3.5 Formulas

We counted a total of 112 formula implementations in the analyzed libraries. Out of those, we could directly map 50 to formulas from the EFD, meaning that the implementation either pointed at the EFD entry or that manual analysis of the formula lead us to conclude that it is equal to an EFD formula. There were 21 distinct EFD formulas used. For 39 implementation formulas we were not able to perform this mapping, but have written them down in EFD format for further analysis. The remaining 23 formulas are out-of-scope of our analysis as they mix coordinate systems.

The unmatched 39 formulas showed similarities to EFD formulas to a certain extent. We measured this similarity in the following way. As in

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

Chapter 4, 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.

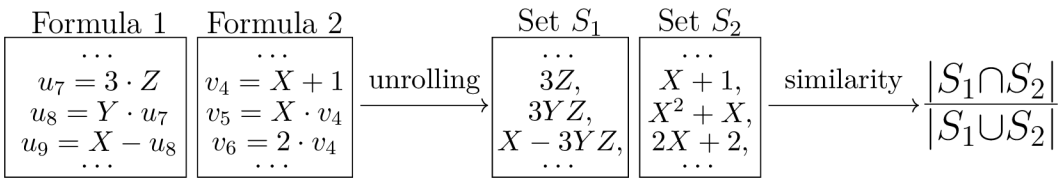


Figure 6.1: Measuring similarity of formulas by expressing intermediate values as polynomials and computing the ratio of the size of the intersection and the union.

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 6.1). For each of the 39 formulas, we found an EFD formula with the highest similarity. For 8 formulas an EFD match with similarity 1 was found and for the rest the largest similarity varied between 0.21 and 0.96.

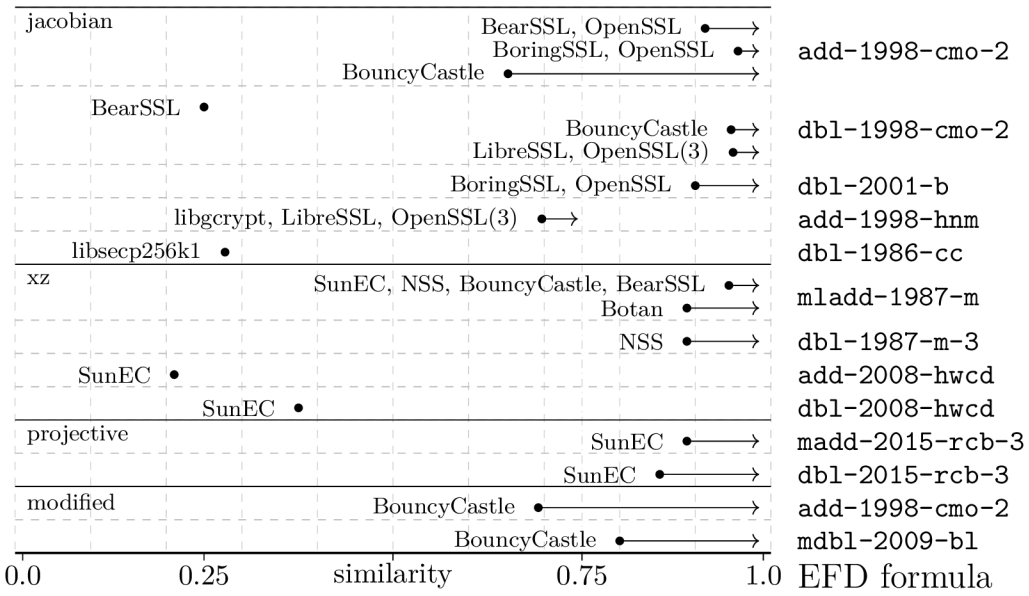


Figure 6.2: Formula similarity between non-EFD formulas found in libraries (dots) and their closest EFD formulas (y-axis). The arrows represent the increase of similarity after our expansion of the EFD.

To bridge the gap between the library implementations of formulas and the EFD 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.
- 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 power of 2 (and can be computed efficiently).

Applying these transformations, we were able to expand the EFD database to contain almost 20 000 formulas[5]. This significantly improved the similarity matching as shown by Figure 6.2 in which the dots represent the 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 89 library formulas, 80 were successfully matched to a transformed variant of an EFD formula with only 4 formulas having the closest match below 0.75.

While the EFD is quite extensive (nearly 200 formulas in the specified models), one should clearly not expect to find all real-world ECC implementations in it without modifications. Our extended EFD (almost 20 000 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 between them for more optimization. This shows the diversity and the vast amount of real-world ECC implementations.

[4] We named this after the climbing move from [215].
[5] This dataset is available at https://doi.org/10.5281/zenodo.10908698.

6.4 Reverse-engineering techniques

This section presents three reverse-engineering methods that utilize existing side-channels attacks — RPA, ZVP and EPA. We use these side-channel attacks to build oracles using which we ask questions about what is happening inside the 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 6.1 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.

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

Table 6.1: Overview of our methods with parts of configurations they target for reverse-engineering. Some parts need to be either chosen or known.

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 table as in Figure 6.3. The rows of the table correspond to configurations from $\mathcal{C}$ and columns correspond to the inputs from $\mathcal{I}$. Based on the table, we construct a search tree for each attack. This tree is then used for an efficient search through the space of configurations $\mathcal{C}$ to minimize the amount of oracle calls (and thus side-channel measurements).

The search 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

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

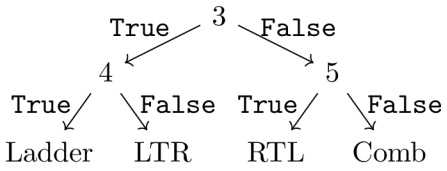


Figure 6.3: Example simulation of the RPA oracle on different multipliers (table rows) and input points (table columns), where $P_0 = (0, y)$. The search tree is constructed based on the table. The scalar is fixed to 213.

example in Figure 6.3 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. In sections 6.4.1, 6.4.2 and 6.4.3, we describe the details of the individual attack methods, including the construction of the set of inputs $\mathcal{I}$.

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 Figure 6.3 is the winning majority result of several RPA oracle calls.

Target implementation. The described methods require control over the input point and can thus target ECDH `Derive`, as well as ECDH `KeyGen`, ECDSA `Sign`, ECDSA `KeyGen`, 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 the 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 [185], computationally demanding checks (such as deterministic primality tests) are often ignored, so we exclude these for the EPA method.

Countermeasures. Our attack-based reverse-engineering methods inherit resistance against several countermeasures from the corresponding attacks (see Section 2.5), 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.

6.4.1 RPA

In RPA, the side-channel oracle is the detection of points with zero coordinates during the scalar multiplication kP, 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 kP. The set of inputs is then

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

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.

6.4.2 ZVP

In ZVP, the side-channel oracle is the detection of 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) and so the goal is to reverse-engineer the used formulas and coordinate system. The set of configurations $\mathcal{C}$ will be the combinations of formulas for all operations used in the multiplier (e.g., $(\texttt{add-2007-bl}, \texttt{dbl-2015-rcb}) \in \mathcal{C}$ for a multiplier using add and dbl).

The general idea of the construction of inputs $\mathcal{I}_{\text{ZVP}}$ is illustrated in Figure 6.4. 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 used 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 6.3.5. We solve the dependent coordinates problem

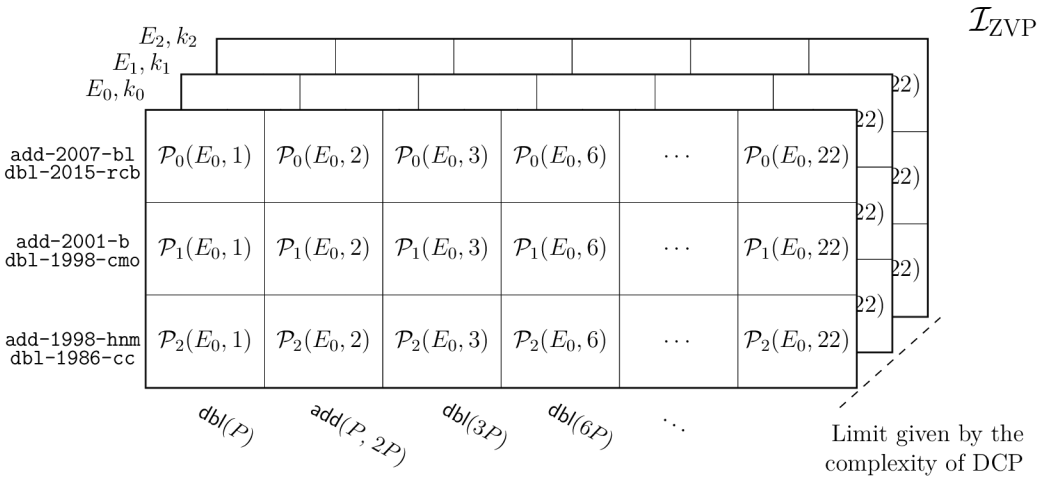


Figure 6.4: The space of possible ZVP points for combinations of formulas for add/dbl and for different curves. Each set $\mathcal{P}_{ij}$ of ZVP points is computed from the intermediate values of the given formula (row) for the corresponding operation (column).

for all the intermediate polynomials (see Section 2.5.1), 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 used scalar, 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., $\mathsf{add}(P, jP)$ or $\mathsf{dbl}(jP)$) as long as the i-th row is the correct choice of formulas. The set $\mathcal{I}_{\text{ZVP}}$ of inputs is then the union of all triples of points, curves and private scalars.

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.

6.4.3 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 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 [184]. 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. [185]. The authors 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.

6.5 Toolkit

In this section we present features of the pyecsca toolkit that are relevant for this chapter. 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. Our reverse-engineering methods are included in the toolkit, along with documentation and tutorials. For a full description of the toolkit, see the original publication [VS1].

6.5.1 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 with 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:

- **Curve model**. One of `Short-Weierstrass`, `Montgomery`, `Edwards`, `Twisted Edwards`. Note that some of these models have restrictions on classes of curves representable by them.
- **Coordinate system**. The implementation choice of coordinates on the curve model. Different 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 6.3. 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.

While our analysis of library formulas (Section 6.3.5) 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.

Curve	Coords	#	Total
$\mathcal{E}_{\mathrm{SW}}$	jacobian	68 180	113 502
	modified	2 856	
	projective	21 890	
	w12-0	476	
	xyzz	4 284	
	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	8 568	11 424
	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	466
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.

Table 6.2: Number of configurations by their components.

RQ2

As Table 6.2 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.

6.6 Results

We evaluate our attack-based RE 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 the execution trace. 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.

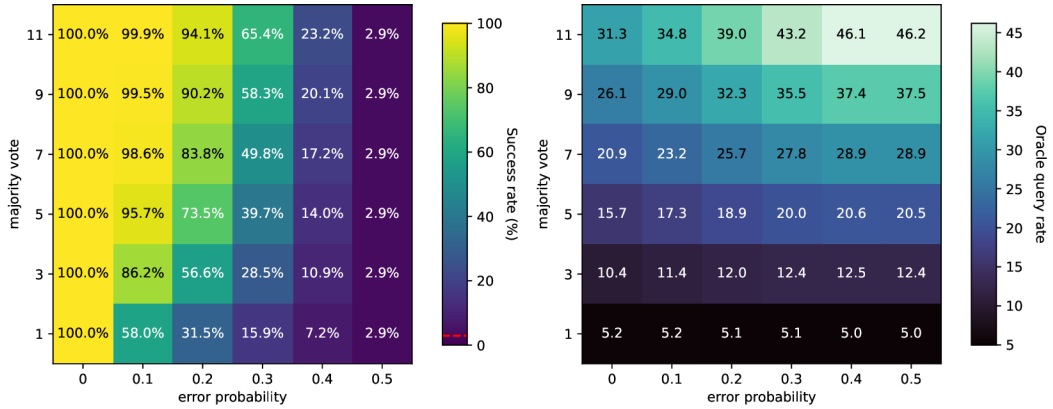
Noise. We model the noise in binary oracles in two ways, symmetric and asymmetric. In the symmetric case there is a single error probability e of a flip in the oracle's output. In the asymmetric case there are two error probabilities (e_0, e_1), corresponding to a flip in the oracle's output $0 \to 1$ 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 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 of the methods because each oracle query amounts to performing a side-channel attack on the target and thus increases attacker effort.

We also examine the search trees built by the methods as they offer insights into the behavior of the methods. Leaves represent indistinguishable sets of configurations. Assuming a uniform probability of each configuration, the expected leaf depth (🍃↕) corresponds to the average oracle query rate, and the expected leaf size (|🍃|) corresponds to the average result size.

Table 6.3: Search tree metrics of our reverse-engineering methods. **Avg.** considers a uniform probability of each leaf, **Exp.** considers a uniform probability of each configuration.

Method	Oracle	$\lvert\mathcal{C}\rvert$	#	Avg. $\lvert$	Avg.	Exp. $\lvert$	Exp.
RPA-RE	binary	34	34	1	5.1	1	5
ZVP-RE	binary	214	74	2.9	5.6	8.7	5.1
ZVP-RE	count	214	134	1.6	3.7	2.4	4
ZVP-RE	position	214	196	1.1	2.2	1.2	2.1



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

Figure 6.5: Results for the RPA-RE method with symmetric noise.

6.6.1 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 search tree that has single element leaves (result sets).

The success rate and oracle query rate for the oracle simulation level are visible in Figure 6.5. 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 metrics of the search tree are in Table 6.3. Our experiment with asymmetric error probabilities behaved symmetrically in all metrics and we thus omit its visualization.

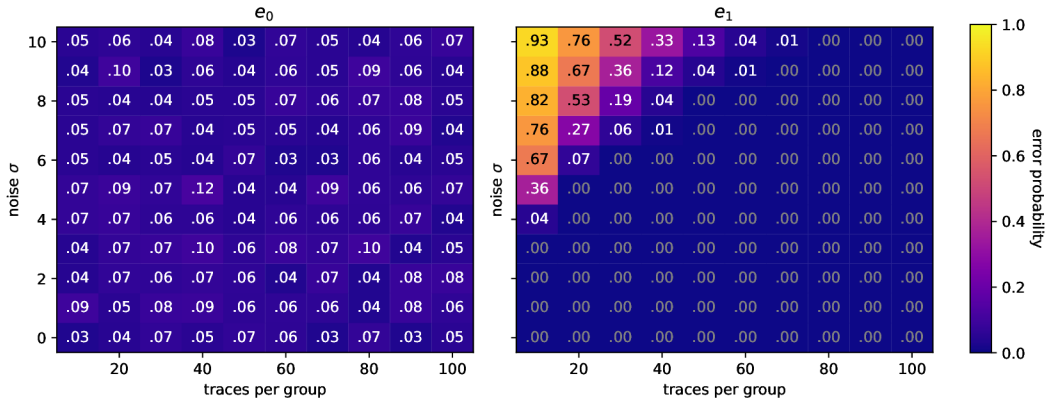


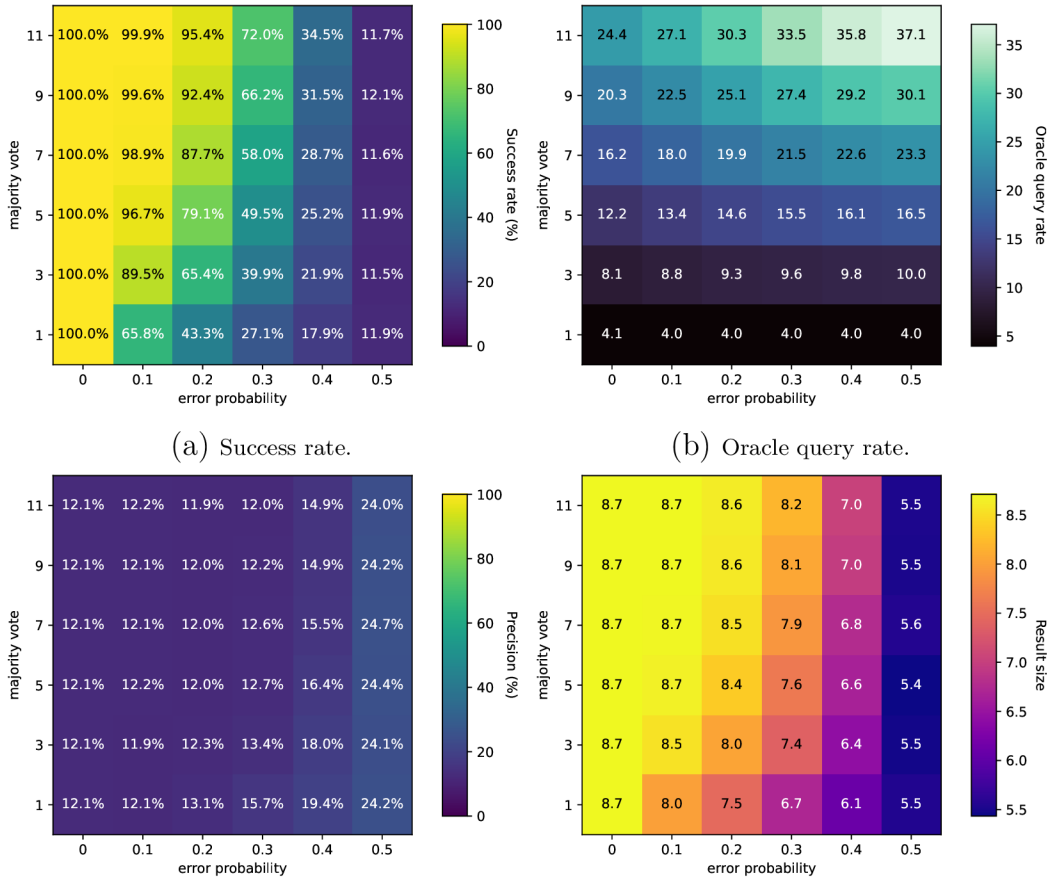
Figure 6.6: Error probabilities e_0 and e_1 in the RPA-RE method during method simulation.

Method simulation. We evaluated the oracle building step of the RPA-RE method while varying the number of simulated traces in $\{10, 20, 30, \ldots, 100\}$ and standard deviation of noise $\sigma \in \{0, 1, 2, \ldots, 10\}$. The trace simulation included the coordinate randomization countermeasure. We implemented the RPA oracle by running simple peak-finding on a difference-of-means trace. The resulting error probabilities e_0 and e_1 are visible in Figure 6.6. As these results show, increasing the number of traces allows us to build an oracle with minimal error probabilities even under considerable noise.

6.6.2 ZVP-RE

For evaluating the ZVP-RE method we chose a basic LTR multiplier on the Weierstrass curve model. Thus all (add, dbl) formula pairs from compatible coordinate systems were 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 ZVP points.

Binary oracle. In this setting the ZVP-RE method builds a tree with 74 leaves. Its metrics are visible in Table 6.3. It is no longer always precise like RPA-RE as Figure 6.7 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 6.3. Note that the leaves vary in size, with the largest leaf containing 30 configurations and 26 leaves containing



(a) Success rate.

(b) Oracle query rate.

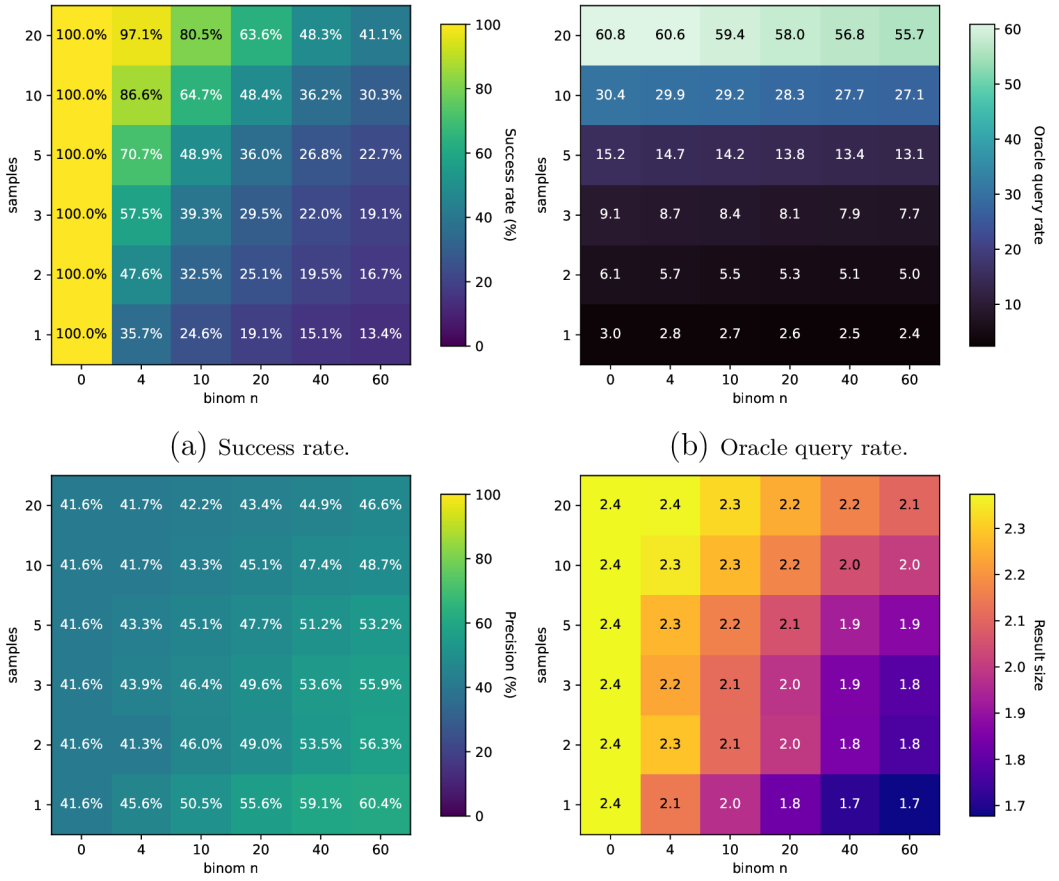
(c) Precision.

(d) Average result size.

Figure 6.7: Results for the ZVP-RE method with a binary oracle with symmetric noise.

one configuration each. This imbalance in the leaf sizes also explains why the high error oracle with low success rate has higher precision than the low error oracle (see Figure 6.7c). When considering the coordinate system only, the method achieves a precision of 94%. Our experiment with asymmetric error probabilities did not behave symmetrically, since search tree was unbalanced. Success rate was better in the prevalence of $0 \to 1$ errors over $1 \to 0$ errors.

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. Figure 6.8 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.



(a) Success rate.

(b) Oracle query rate.

(c) Precision.

(d) Average result size.

Figure 6.8: Results for the ZVP-RE method with a zero count oracle with binomial noise.

RQ3

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

6.7 Conclusions

Implementing elliptic curve cryptography is a non-trivial process, requiring a range of implementation choices — especially when considering performant, 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 analysing 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 a well documented and tested package, providing extensive functionality, including enumeration of millions of ECC implementations, their synthesis for embedded devices as well as acquisition, processing, and visualization of traces.

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 reduce the space of addition formulas considerably. Our reverse-engineering methods are resistant to coordinate and curve randomization. We leave the adaptation of the methods against more countermeasures as future work.

7 Lying to ECC implementations

This chapter is based on the following publication:

[VS3] V. Suchanek, J. Jancar, J. Kvapil, P. Svenda, and Ł. Chmielewski. "ECTester: Reverse-engineering side-channel countermeasures of ECC implementations". *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025:4, 2025, pp. 290–316

7.1 Introduction

ECC implementations have been hard hit by bugs and vulnerabilities, likely due to their complexity. Focusing just on input validation in ECDH alone, there has been the small-subgroup attack [120], twist curve attack [78], invalid curve attack [31, 7] and degenerate curve attack [147]. TLS servers and libraries in the wild were found vulnerable to the invalid curve attack in 2015 [103]. The same then happened to JSON Web Token implementations in 2017 [149, 174]. Staying with ECDH, there were several cases where an innocuous miscomputation bug lead to key-recovery, first in OpenSSL in 2012 [45], and then in the Go standard library in 2017 [208].

Expanding the lens to include ECDSA and similar signature schemes, we see more issues. Input validation issues lead to such curious cases, where the $(0, 0)$ signature was deemed valid for any message by any key. This issue was dubbed "psychic signatures", discovered in OpenJDK in 2022 and surfaced both before and after in cryptocurrency-related implementations [124, 145, 150]. Failure to properly validate certificate contents lead to the "CurveBall" vulnerability in Microsoft CryptoAPI, which allowed trivial certificate spoofing (`CVE-2020-0601`). Finally, we invite an interested reader to examine the list of bugs found by the CryptoFuzz project [138, 138].

This unending stream of issues and vulnerabilities raises the question: *Do current implementations of ECC properly defend against known vul-*

nerabilities? Bug reoccurrence, i.e., a regression, is quite common in software development, as is bug co-occurrence, as demonstrated by the several different implementations vulnerable to invalid curve attacks at different points in time. The efforts that uncovered the above issues were usually focused only on a single or a few implementations, with the exception of the CryptoFuzz project. However, all of them were focused on open-source software implementations. To answer the posed question, we systematically test open-source ECC libraries and smartcards against a plethora of tests constructed from known attacks in our tool ECTester.

In Chapter 6, we have shown that the space of possible ECC implementations is vast enough to warrant reverse engineering. All the combinations of different coordinate systems, formulas, multipliers, and their parameters yielded almost 140 thousand configurations. The analysis of open-source libraries proved that this is not true just in theory, but that developers leverage all of the options and even extend them by modifying existing techniques. While we proposed several methods for reverse-engineering using side-channel attacks, they were *not* applicable to implementations using scalar randomization.

This raises yet another question: *Can the behavior of implementations under tests be used for reverse-engineering?* We answer this in the positive, by designing several techniques for reverse engineering scalar randomization countermeasures that are able to distinguish between group scalar randomization, additive, multiplicative, or Euclidean splitting. The techniques do not require side-channel measurements and are immune to other side-channel attack countermeasures. Furthermore, the techniques can recover the random mask size, and with some probability the mask value for two of the countermeasures. Such mask value recovery on black-box targets allows for more side-channel analysis than was possible before, including mounting the learning phase of profiled attacks. Our techniques question the practice of rewarding information hiding and security by obscurity in security certifications such as Common Criteria [199, 183].

Contributions:

- The ECTester tool for testing of ECC libraries and JavaCards, supporting 12 popular libraries and any JavaCard, offering a unified API for testing. Released[1] under a permissive open-source license.

[1]Its repository is available here https://github.com/crocs-muni/ECTester.

- Techniques for reverse-engineering of scalar randomization countermeasures that only require control over the domain parameters, but require no side-channel measurements or faults, and can recover the mask size and value.
- Reverse-engineering of countermeasures on 13 cryptographic smartcards from major manufacturers, many of which are certified under Common Criteria or FIPS 140.

Outline. In Section 7.2 we present related work on testing cryptographic implementations and reverse-engineering. Section 7.3 presents our methodology for testing ECC implementations, while Section 7.4 contains a selection of our results. In Section 7.5 we present our techniques for reverse engineering scalar randomization countermeasures and results on smartcards from major manufacturers. We discuss the implications of our findings and mitigations in Section 7.6.

Responsible disclosure. We contacted three out of the five manufacturers of smartcards we were able to reverse-engineer and shared our findings with them. Despite trying, we were unable to establish security contacts with the remaining two manufacturers.

7.2 Related work

There is relatively little work in the area of testing of cryptographic implementations that is directly related to our approach.

- Firstly, one may get assurances of security through code review and analysis. However, this is only applicable in a white-box scenario with available source code, whereas our approach works also for black-box implementations.
- Secondly, one may perform implementation testing, for example using test-vectors such as the Wycheproof project [46] or by fuzzing such as the Cryptofuzz project [138, 138]. Our approach in ECTester fits this category. In contrast to Wycheproof, which recently moved to only focus on aggregating test-vector data, we support testing many native libraries directly as well as any compatible JavaCards. The future of the Cryptofuzz project is uncertain, as its original author took down the repository and only a fork under the Mozilla Security organization is available.

- Thirdly, one can test artifacts produced by the implementation (e.g., keys). This is the approach taken by Google's project Paranoid [86] and also in the *CurveSwap* work [207]. Our approach is able to exercise the target implementation and gain more information on it than possible by purely analyzing its keys.

In the area of reverse-engineering of public key cryptography implementations there have been some notable examples of both manual and automated reverse-engineering. Amiel et al. [3] presented a technique for reverse-engineering the word size and modular multiplication algorithm in RSA implementations. Remarkable examples of manual reverse-engineering can be found in the "Side Journey to Titan" [173] and "EUCLEAK" [172] works, which demonstrate complex attacks on ECC implementations in real-world hardware.

The pyecsca tool presented in Chapter 6 is able to automatically reverse-engineer ECC implementation details via side-channel attacks. While it targets the scalar multiplier and addition formulas used by the implementation, its reverse-engineering does not work in the presence of scalar randomization countermeasures. In contrast, the techniques presented in this chapter are able to reverse-engineer the scalar randomization countermeasures and do not require side-channel measurements. We utilize the pyecsca toolkit for simulations and demonstration.

7.3 Methodology

In this section we present our methodology for testing black-box elliptic curve cryptography implementations using our ECTester tool. We do not discuss the reverse-engineering of randomization countermeasures in this section and leave it for Section 7.5.

7.3.1 Overview

The ECTester tool comprises of several components, see Figure 7.1:

Standalone: The standalone component focuses on testing ECC libraries, both native and Java-based ones. It uses the Java Cryptography Architecture (JCA) providers of the Java-based ECC libraries to interact with them. However, the large variety of native libraries we target does not offer any common API we could use. Thus, for each

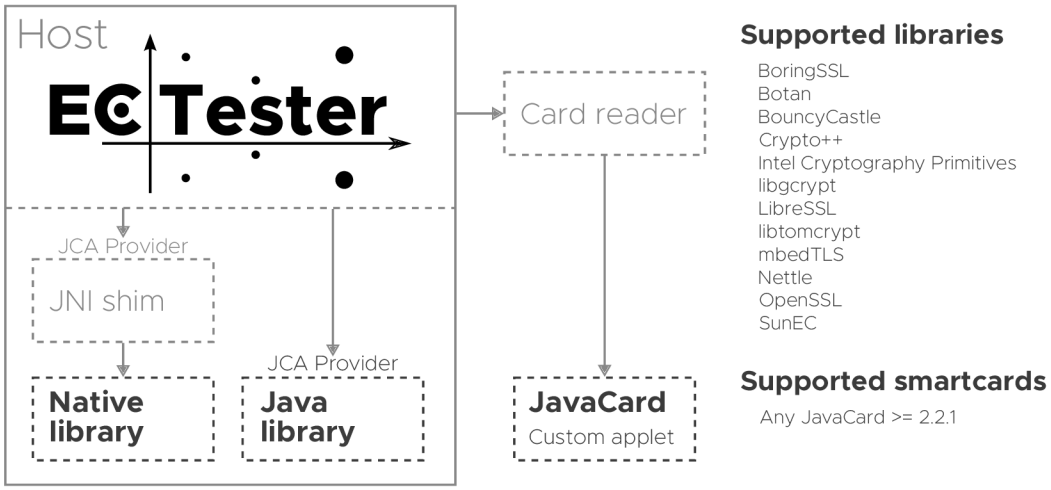


Figure 7.1: Architecture of the ECTester tool, which is able to interact with ECC implementations in Java libraries, native libraries through JNI shims, and JavaCards. Targets are displayed in bold.

native library we implement a *shim* using the Java Native Interface (JNI) that uses the library to offer a common JCA provider interface. The standalone component offers a unified command-line interface for working with, and testing, the libraries.

Reader: The reader component focuses on testing JavaCard-based smartcards. To do so, it requires that a custom applet (see Applet component bellow) is installed on the JavaCard. It offers a similar command-line interface as the standalone component.

Applet: The applet component contains the applet code that is loaded on tested JavaCards and provides a rich interface to their ECC implementations. It uses only standard JavaCard APIs and not the proprietary ones as those are often only available under a non-disclosure agreement with the card vendor. Since this component is compiled for JavaCards it needs to conform to the JavaCard subset of Java.

Common: The common component contains all of the test data: elliptic curves, points, private and public keys, signatures, and test results. It also contains common functionality shared between the reader and standalone components, such as I/O or test evaluation. It supports test result output in YAML/XML and rich text formats, allowing for post-processing and further analysis.

Finally, all of the components are complemented by a set of Jupyter notebooks for data and results analysis as well as visualization. Currently, the tool supports 10 native libraries and 2 Java ones, listed in Figure 7.1. Adding additional libraries involves writing a simple shim which implements a conversion layer between ECTester and the native library. However, as libraries evolve over time and their API changes, we found that we need to update these shims to new library versions quite often. Furthermore, the changes to the shim necessary to make it work with a new library version may not be backwards compatible with old library versions. We use the nix tool and nixpkgs repository to enable testing of historic and current library versions and simplify the library build system.

7.3.2 Test suites

ECTester contains a wide range of test-suites which we document here. A full listing of the tests is out-of-scope of this work, but can be found in our repository.

Default. Tests support for ECC and the presence of any default curves on the target. Given that we also target JavaCards, these might not be present or the target might not even support ECC. It also tests keypair allocation, generation, ECDH, and ECDSA. ECDH is first tested with two valid generated keypairs, then with a compressed public key to test support for valid compressed points.

Test-Vectors Tests ECDH using known test vectors provided by NIST [110], SECG [48], and Brainpool [96, 130]. This test-suite also performs a cross-validation test of ECDH and ECDSA, using the target and validating its output using a bundled version of the BouncyCastle library.

Compression. Tests support for the compression of public points in ECDH as specified in ANSI X9.62 including the compressed and hybrid form. Also tests target response to a hybrid point with a wrong y coordinate and to the point at infinity (as public key in ECDH). Tests ECDH with an invalid compressed point which does not lie on the curve.

Miscellaneous. Tests ECDH and ECDSA over weak curves such as supersingular curves, anomalous curves, or Barreto-Naehrig curves with small embedding degree and CM discriminant. Also tests ECDH over MNT curves, M curves, and Curve25519 transformed into Weierstrass form.

Signature. Tests ECDSA verification, with well-formed but invalid signatures and a wide range of malformed signatures. This includes $r, s \in \{0, 1, n\}$, taking r, s random, malformed ASN.1 format or a negated public key point. This test suite also tests signature malleability, i.e., whether a correct signature modified by adding the curve order to one of the values r, s still verifies.

Wrong. Tests behavior on ill-defined curves with parameters that do not follow the standard definition of a curve. The base-field is invalid, i.e., the number defining the prime field is not a prime or the polynomial defining the binary field is reducible. Tests also include the generator point not on the curve or not a valid affine point. Tests include a curve where the order or the cofactor is zero or one.

Composite. Tests using various composite order curves, including pseudoprime orders.

Invalid. Tests the invalid curve attack [31, 103] using known named curves from several categories (SECG, NIST, and Brainpool) against pre-generated invalid public keys. Similar tests are implemented for the `Twist` curve attack and for the `Degenerate` curve attack in respective test-suites.

Cofactor. Tests the small-subgroup attack, i.e., whether the target correctly rejects points that lie on the curve but not on the subgroup generated by the specified generator during ECDH. This is done with curves where the cofactor subgroup has small order, then with curves that have order equal to the product of two large primes. The test sets the generator with order of one prime and tries points on the subgroup of the other prime order.

Edge-Cases. Tests various inputs to ECDH which may cause an implementation to achieve a certain edge-case state during ECDH. Some of the data is from Paranoid Crypto [86] or Wycheproof [46]. Tests include `CVE-2017-10176` and `CVE-2017-8932` and an OpenSSL modular reduction bug presented in [45]. Various custom edge private key values are also tested. `CVE-2017-10176` was in implementation issue in the SunEC Java library and NSS (`CVE-2017-7781`), that caused the implementation to reach the point at infinity during ECDH computation [175]. `CVE-2017-8932` was an implementation issue in the Go standard library, in particular its scalar multiplication algorithm on the P-256 curve which leaked information about the private key [208].

7.4 Results

In this section we present results obtained from running ECTester test-suites on supported libraries and several JavaCards. We do not include results from every test-suite due to space and pick only "interesting" results from the point-of-view of an attacker. For an overview of the tested cards, see Table 7.1. For an overview of the tested library versions (and supported versions) see Table B.2.

	Manuf.	Model	Chip	CPLC
N1	NXP	JCOP J3A081*	P5	4790.5168.4791.0078
N2	NXP	JCOP J2D081*	P5	4790.5167.4791.0078
N3	NXP	JCOP21 J2E145G*	P5	4790.5167.4791.2348
N4	NXP	JCOP3 J3H145*	P60	4790.0503.8211.6351
N6	NXP	JCOP4 J3R180*	P71	4790.D321.4700.0000
N9	NXP	JCOP31 J3A081*	P5	4790.5040.4791.8102
I1	Infineon	SECORA IDS SLJ52GDT120CS*	-	4090.1912.4090.9078
I2	Infineon	CJTOP SLJ52GLA080AL*	-	4090.7165.544C.2151
A1	Athena	IDProtect†	AT90	4180.010B.8211.0352
G1	G&D	Smartcafe 7.0	-	0005.0056.D001.4212
G2	G&D	Smartcafe 6.0†	-	4790.5037.1671.1146
S2	TaiSYS	SIMoME†	-	FFFF.FFFF.FFFF.FFFF
F1	Feitian	JavaCOS A22 CR*	-	4090.7892.86AA.7068
F2	Feitian	JavaCOS JC30M48 CR	-	-

Table 7.1: Smartcards analyzed in this work. The CPLC column presents the `ICFabricator`, `ICType`, `OperatingSystemID`, and `OperatingSystemReleaseDate` values from the smartcard.

*/† Denotes Common Criteria or FIPS 140 certified smartcards, respectively.

The identification of the chip, or even exact smartcard model, or certification status in Table 7.1 presents our best-effort and might contain mis-identifications. Precisely identifying the exact smartcard model and certificate coverage without access to manufacturer documentation is challenging. We used the sec-certs project to speed-up our search of certification documents [106].

7.4.1 Test-vector failures

We encountered several unexpected test-vector failures in our testing of JavaCards. The A1 card produced a wrong ECDH secret in the brain-

poolP512r1 test, as well as the P-521 DHC test from NIST. Similarly, it failed to produce the correct shared secret on the secp521r1, brainpoolP512r1, and brainpoolP512t1 curves. The G1 card produced wrong ECDH secrets on 512-bit curves, yet handled 521-bit ones fine. The G2 card outputs only 20 bytes of the derived secret if the used keypair was generated on the card, even if the requested ECDH version is plain (not hashed) and the curve is larger.

7.4.2 Timing leakage

The ECTester tool was used to discover a group of timing vulnerabilities in ECDSA known as Minerva [105]. This shows the versatility of the tool in uncovering diverse issues like timing attacks which we do not focus on in this work. The functionality of the tool, with respect to test suites and usability was expanded heavily since then.

7.4.3 Psychic signatures

The "psychic signatures" bug, named after Doctor Who's psychic paper, was a bug in ECDSA verification in Java 15–18 that allowed a signature of r, s both equal to zero to pass as valid for any message. This bug was discovered in the Java case by Madden [124] in 2022 and found in other cryptocurrency-related ECDSA implementations in 2021 [145]. A test for such a case was present in the `Signature` test-suite in ECTester since 2018, however the tool — written in Java — only supported Java 8 at the time. Support for newer Java versions was added in 2024.

7.4.4 Domain parameter validation

The `Composite` test suite tests the set up of elliptic curve parameters with composite order n. Curves with composite order are vulnerable to the Pohlig-Hellman attack and any implementation that supports custom curves and wants to avoid it should test primality of the order or use standard named curves. Column 'prime n' in Table 7.2 shows the results for twelve popular ECC libraries and thirteen JavaCards. For eleven cards and eight libraries, we were able to set-up a composite order curve and perform either ECDSA or ECDH. For the rest, we failed to do so. The results for the A1,G1 and I2 cards agree with the findings from [185] that both A1, I2 contain some primality tests on n and G1 does not.

Card	ord P	ord G	prime n
N1	✓	✓	✓
N2	✓	✓	✓
N3	✓	✓	✓
N4	✓	✓	✓
N6	✓	✓	✓
N9	✓	✓	✓
I1	✓	✓	✓
I2	✓	✓	✗
A1	✗	✗	✗
G1	✗	✗	✓
S2	✓	✓	✓
F1	✓	✓	✓
F2	✓	✓	✓

Library	ord P	ord G	prime n
BoringSSL	✓	✓	✓
Botan	✓	✓	✓
BouncyCastle	✓	✓	✓
Crypto++	✓	✓	✓
Intel Crypto	✓	✗	✓
libgcrypt	✗	✗	✗
LibreSSL	✓	✗	✓
libtomcrypt	✓	✗	✗
mbedTLS	✓	✓	✓
Nettle	✗	✗	✗
OpenSSL	✓	✗	✓
SunEC	✗	✗	✗

Table 7.2: Results of ECTester with invalid parameters on popular libraries and JavaCards. The symbols ✓, ✗ denote ECTester managed (or failed) to pass a curve with composite order (prime n) or a public key or generator with invalid order (ord P and ord G).

Next, we tested the validation of the generator point. Any generator of the prime subgroup should have prime order n. Since any point on a prime order curve will be a generator, we selected a curve with cofactor $h > 1$ that we concealed from the implementation (i.e., set to $h = 1$ in the parameters). We set the curve order to n and tested ECDH and ECDSA with a generator of order nh. Additionally, we used the secp256r1 curve and changed the claimed prime order n to another prime. We were successful for all except for two cards (column ord G in Table 7.2) and for five libraries. Inspection of the code showed that the three libraries that support custom curves and rejected the generator (OpenSSL, LibreSSL and Intel Crypto) correctly check the order of the generator.

7.4.5 Cofactor validation

The `Cofactor` test suite tests how implementations deal with public key points in the wrong subgroup. This is relevant for the `Derive` subroutine in ECDH and Verify in ECDSA. We selected curves with cofactor $h > 1$ and gave the implementations public key points of an incorrect order nh (the expected public key should have order n). We considered two options to fully understand the behavior with incorrect

options: conceal the cofactor in the curve specification and pretend that $h = 1$ or transparently present the true cofactor h. In the first case, all of the cards except for A1 and G1 accepted the point as well as all of the libraries that supported custom curves with the addition of libtomcrypt (see Table 7.2). The libtomcrypt library (the latest version v1.18.2) did not support cofactor curves, but it also did not validate that the point is lying on the curve. Hence, a small order point from a different curve could be used. The lack of the check makes the library vulnerable to the invalid curve attack. Furthermore, we observed that repeated runs on the cards produced different results. For the fixed input point P of order nh, the results were of the form $P, P + nP, \ldots, P + (h-1)nP$. This pointed to a randomization of the scalar multiplication that we will explore in the further sections.

Then we repeated the same tests with a point of order nh but with correctly set cofactor h. Surprisingly, all of the cards continued to accept the point. One explanation might be that the user is supposed to correctly choose the correct type of ECDH depending on the cofactor and in case of $h > 1$ use the Cofactor Diffie-Hellman (CDH) protocol. This is what we saw in the OpenSSL library, which offers both ECDH and CDH, leaving the choice up to the user by setting the `ecdh-cofactor-mode` flag. Crypto++ and Intel Crypto implement CDH as well and Botan uses the inverse version of CDH to be compatible with the regular ECDH. BouncyCastle implements the cofactor check, but offers the option to omit it by setting a corresponding flag for the validation function. MbedTLS states in the documentation that the cofactor check "is expensive, is not required by standards, and should not be necessary if the group used has a small cofactor". For libgcrypt and LibreSSL we were not able to confidently determine their approach.

Next, we tested the implementations with points of small order h. We observed that the libraries and cards very often did not successfully finish with a (correct) ECDH secret given a point of small order as the public key. This is not surprising as when the scalar is divisible by the order, the result of the scalar multiplication is the infinity point that cannot be transformed to an affine form and thus an ECDH secret. If the input point P has order l this happens with probability $\frac{1}{l}$ given random scalars. This is indeed what we saw in most of the libraries (Figure 7.2). However, three libraries, libtomcrypt, BouncyCastle, mbedTLS had different distributions. For instance, while the rest of the libraries erred in roughly $\frac{1}{3}$ of the cases

for points of order 3 and very rarely for points of order 37, mbedTLS erred almost always for the same points of order 3 and points of order 37.

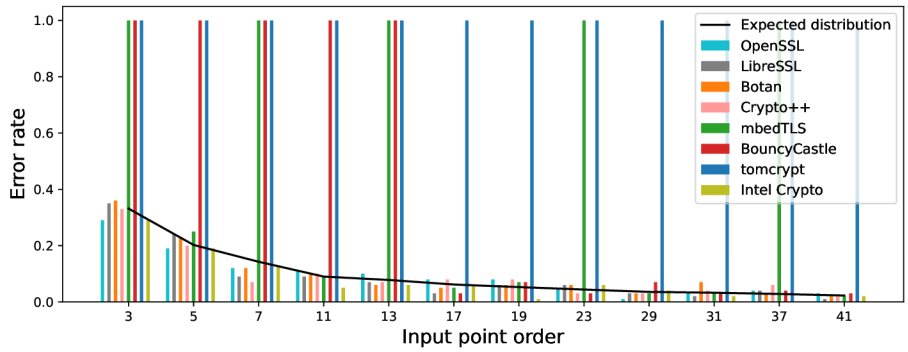


Figure 7.2: Error distribution of ECDH runs given an input point of low order.

Closer inspection of BouncyCastle and mbedTLS revealed a simple explanation. BouncyCastle uses the Window-NAF multiplier for scalar multiplication which begins with the precomputation of the a few small multiples of the input point that are then transformed to an affine form. Similarly, mbedTLS uses the Comb multiplier with a precomputation. The transformation to the affine form then causes the errors.

For libtomcrypt, the explanation turned out to be in the formula that is used for point addition (`add-1998-hnm`). If either of the input points has the projective z-coordinate equal to zero then the result satisfies the same regardless of the second points. In particular, if one of the input points is $\mathcal{O}$ (which is defined as $x = 0$, $y = 1$, $z = 0$) then the result also has $z = 0$. This means that if such a point appears during the scalar multiplication, it propagates through the scalar multiplication till the end where the implementation attempts to transform it to an affine point resulting in an error. Since the input point, say P, has low order h, very likely a multiple of the point, mP, with m divisible by h appears during the scalar multiplication. Consequently, the point mP is the neutral point $\mathcal{O}$ which causes the error.

We observed similar behavior with the cards including `N1` and `N3`, though some of the cards stopped responding and so we did not do further measurements. The frequency of the errors was significantly higher than the expected $\frac{1}{l}$. We concluded that the implementations on the card suffer from the same problem with incomplete formulas as we saw in libtomcrypt and they do not handle the infinity point well. Moreover, the errors did not appear deterministically across multiple

measurements with the same input including a fixed key. This agreed with the observation that some sort of (scalar) randomization is present on the card that we also saw in the results of the cofactor testing.

As a final note for this part, we will comment on the vulnerability to the small subgroup attack. It is known that a few bits of a private static ECDH key can be recovered using this attack. However, the lack of cofactor validation makes the verification of ECDSA signatures also vulnerable by the following simple attack. To verify a valid ECDSA signature (r, s) with a corresponding public key Q, one checks that the x-coordinate of the point $s^{-1}H(m)G + s^{-1}rQ$ is equal to r modulo n. Let T be a point of order 2 (i.e., we assume that the cofactor is even). Assuming that $s^{-1}r$ is an even number, the point $Q + T$ will also pass as a public key for the verification of the signature (r, s). We experimentally confirmed this on the `N1`, `N2`, `N3`, `N9`, and `N4` cards. Attempts on `I1`, `I2` failed, and so we concluded that the cards have some check (here we did not conceal the cofactor) or the verification is done in a different way. Note that the lack of cofactor validation for verification means that the signatures are not bound to the public key Q, i.e., the signature scheme is not *strongly binding* as defined in [50].

7.4.6 $n > p$ overflow

The `Edge-Cases` test suite revealed that the `N4` card did not support all the private keys from the full interval $[1, n - 1]$ for the secp160r1 curve. This curve has a 160-bit prime $p = 2^{160} - 2^{31} - 1$ and 161-bit order $n = p + 1 + t$, where t has 81 bits. Based on the tests in the suite, we concluded that the private key must have 160 bits for the implementation to accept it. This results in a small bias as the values from the interval $[2^{160}, n - 1]$ are never used. Assuming that the implementation applies the same limit on the signature nonces, this would lead to systematic nonce leakage of 1 bit, though with the negligible probability $< 2^{-79}$.

7.5 Reverse-engineering scalar randomization

ECTester results on JavaCards presented in the previous section show that the tested cards usually do not validate the domain parameters and the public point. Incorrect order of the curve, cofactor, or generator with a wrong order can be passed to the card and used in ECC protocols

implemented on the card. Observation of the test results already gave us evidence of side-channel countermeasures such as scalar randomization or improper handling of the infinity point. In this section, we push these ideas further and design methods for reverse-engineering of scalar randomization countermeasures (and random mask sizes and values) based on the behavior of the implementation under invalid inputs.

We will target a scalar multiplication kP using an unknown multiplier, where P is a point (either the generator for `KeyGen` and `Sign` or the public key point for `Derive`) and k is the used scalar (either the nonce or the private key). The computation of kP is randomized by an unknown scalar randomization ρ. We denote the randomization as $\rho(k, P)$, where under valid inputs $\rho(k, P) = kP$. We will assume that k and P are either known or can be set, but this will depend on the individual tests presented below. Finally, we will assume that ρ is one of the scalar randomization techniques described in Section 2.2.1. We will also briefly comment on the point blinding at the end of this section.

The general idea is to set up the curve and the point P such that $\rho(k, P) \neq kP$ and the value of $\rho(k, P)$ leaks information about the randomization ρ. This will very often require to conceal the true subgroup order n and the cofactor h from the implementation. We will denote $\overline{n}$, $\overline{h}$, $\overline{G}$, and $\overline{P}$ the claimed (possibly incorrect) order, cofactor, generator, and public key, respectively. Table 7.3 summarizes the setting of the parameters for our tests.

Test	Target	$\lvert E(\mathbb{F}_p)\rvert$	n^*	$\overline{n}$	$\overline{h}$	ord($\overline{G}$)	ord($\overline{P}$)	k
$3n$	`Derive`	$3n$	✓	n	1	n	$3n$	fixed
	`Sign/KeyGen`					$3n$	n	any
composite	`Derive`	n	✗	n	1	n	n	fixed
	`Sign/KeyGen`							any
$k = 10$	`Derive`	n	✓	n	1	n	n	10
$n + \epsilon$	`Derive`	n	✗	$n + \epsilon$	1	n	n	fixed
	`Sign/KeyGen`							any
EPA	`Derive`	$373n$	✓	n	1	n	373	fixed

Table 7.3: Setting of ECC parameters for our reverse-engineering of scalar randomization.

* Denotes whether n is a prime or not.

Test	Mask	GSR	Additive split	Euclidean split	Multiplic. split	None
$3n$		$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	$\frac{1}{2}, \frac{1}{2}, 0$	$1, 0, 0$	$\frac{5}{9}, \frac{2}{9}, \frac{2}{9}$	$1, 0, 0$
composite		100%	100%	100%	$< 100\%^{\dagger}$	100%
$k = 10$		$< 100\%^{*}$	100%	$< 100\%^{*}$	100%	100%
$n + \epsilon$	size	Yes	No	No	Yes	-
	value	100%	-	-	$\approx 10\%$	-
EPA		> 0	> 0	> 0	> 0	0

Table 7.4: Behavior of our tests with the common scalar randomization techniques.

* Depends on the scalar multiplier.

† Depends on the field inversion algorithm.

7.5.1 Test $3n$

The test uses an elliptic curve of order $3n$, where n is a large prime, and an input point P of order $3n$. We will show that $\rho(k, P) = kP + snP$ for $s \in \{0, 1, 2\}$ for each of the scalar randomization technique ρ. More importantly, the distribution of s is different for each technique ρ, giving us a unique fingerprint for each of them. The test will do multiple measurements and output the distribution (three probabilities for the three possible values of s). This measured distribution is then compared with the theoretical distributions that we will now derive. The summary of the theoretical distributions is shown in Table 7.4.

For GSR, $\rho(k, P) = (k + rn)P = (k + sn)P$, where s is equal to $r \pmod 3$ and thus uniformly sampled from $\{0, 1, 2\}$, i.e., each with probability $\frac{1}{3}$. For the additive splitting, $\rho(k, P) = (k - r)P + rP$. If $k > r$ then $\rho(k, P) = kP$. In the opposite case, the negative scalar $(k - r)$ is reduced modulo n and $\rho(k, P) = (k + n)P$. Assuming $k \sim n \sim r$, these two cases should happen with $\frac{1}{2}$ probability each. It could also be the case that the negative integer $(k - r)$ is not reduced (and perhaps the sign of P is changed instead), in which case $\rho(k, P)$ would be always kP. The Euclidean splitting does not use the value of the order n in any way and so $\rho(k, P) = kP$.

Finally, the multiplicative split will satisfy $\rho(k, P) = k'rP$, where $k' = kr^{-1} \pmod n$ and r^{-1} is the inverse of $r \bmod n$. For some integer t, we can write $\rho(k, P) = (kr^{-1} + tn)rP$. Since P has order $3n$, $\rho(k, P) = kP + snP$ for $s \in \{0, 1, 2\}$, where s satisfies $(kr^{-1} + tn)r = k + sn \pmod 3$. Since r is a random integer, all three values r, r^{-1} and t follow

a uniform distribution modulo 3. Assuming that $k \pmod 3$ is fixed, $(kr^{-1} + tn)$ follows this distribution as well. Then the distribution of the product $(kr^{-1} + tn)r$, and consequently of s, is $0, 1, 2$ with probabilities $\frac{5}{9}, \frac{2}{9}, \frac{2}{9}$, respectively. Note that this holds for a fixed $k \pmod 3$ and so for `KeyGen` and `Sign`, the nonce and private key must be grouped based on the value of $k \pmod 3$ when observing the distribution.

7.5.2 Test composite

We can use a composite order curve to detect the inversion $r^{-1} \pmod n$ computed in the multiplicative splitting randomization, but not others. If r and n share a nontrivial factor, the computation of $r^{-1} \pmod n$ can then either lead to an infinite loop, an error, or an undefined value, as the inverse does not exist. If the extended Euclidean algorithm is used, the result will be incorrect or produce an error with probability $\frac{\phi(n)}{n}$. If the Fermat's little theorem is used with $r^{-1} = r^{n-2} \pmod n$ then it is unlikely that any of the inversions will be correct, though errors are not expected. The measurement is repeated and the output of the test is the rate of successful results. The rest of the countermeasures do not use the primality of the order, and so they will have a 100% success rate. A success rate lower than 100% is indicative of the multiplicative splitting.

Beware that in `Sign` this test will detect the inversion of the nonce. To determine whether we are detecting any further inversions (in the multiplicative splitting), we have to consider the expected frequency of errors and compare them with the measurements.

Note that the frequency of the errors can be adjusted for both inversion algorithms. To modify the error rate of the inversion using Fermat's little theorem, we can use Carmichael numbers that will satisfy $r^{n-2} = r^{-1} \pmod n$ if and only if r and n are coprime. The probability of error is then the same as for the Euclidean algorithm: $\frac{\phi(n)}{n}$. By selecting n with a different number of factors we can increase or decrease the probability.

7.5.3 Test $k = 10$

This test is designed for `Derive` with key $k = 10$ to detect the GSR countermeasure, where $\rho(k, P) = (k + rn)P = (10 + rn)P$. Simulations with the pyecsca tool showed that scalar multipliers tend to compute rnP as an intermediate value while building up the resulting $10P + rnP$. Naturally, this tends to happen for any small k. This can lead to an

error if the implementation does not handle $rnP = \mathcal{O}$ well. At the same time, other countermeasures do not use scalars larger than n and are therefore unlikely to cause a similar error. The only assumption of this test is that we can fix $k = 10$ and so this can be set up on any curve including secp256r1, which we used. The result of the test is the rate of the correctly computed shared secrets. However, not every multiplier has to cross the point rnP and so a 100% success rate does not necessarily mean that GSR is not present in the implementation. Other small scalars can be used to increase the confidence.

7.5.4 Test $n + \epsilon$

The $n + \epsilon$ test is aimed at reverse-engineering the randomization mask used in GSR or in the multiplicative splitting. It works under the assumption that the random mask r is sampled from $[1, 2^b]$ where b is much smaller than $\log_2(n)$. The idea is to set a curve with order n and claim a different order $n + \epsilon$ for small ϵ, where $n + \epsilon$ is a prime to pass any primality test. The GSR randomization then computes $\rho(k, P) = (k + r(n + \epsilon)) = (k + r\epsilon)P = dP$, where $d = k + r\epsilon \pmod{n}$ and P has order n. Since r is significantly smaller than n then the equality $d = k + r\epsilon$ holds over integers, giving us also the mask as $r = \frac{d-k}{\epsilon}$. The only unknown (besides the mask) is the discrete logarithm d of $\rho(k, P)$ with respect to P. If we set a weak curve with composite n, we can easily find the discrete logarithm d and find the mask r. We used a curve with order n equal to a product of 32-bit primes. Note that we need access to the result point $(k + \epsilon r)P$, which holds for `KeyGen`, `Sign`, and plain `Derive` where the point can be recovered from the shared secret.

The multiplicative splitting behaves with the $n+\epsilon$ test in the following way. The randomization is $\rho(k, P) = k'rP$, where $k'r = k \pmod{n + \epsilon}$. The last modular equality can be written as $k'r = k + t(n + \epsilon)$ for some integer t. We estimate the size of t by comparing the sizes of both sides of the equality. Specifically, $k'r \sim tn$ where $k' < n + \epsilon$ and so t is expected to be smaller than r and consequently significantly smaller than n. If we find the discrete logarithm d of $k'rP$ with respect to P, we get $d = k'r = k + \epsilon t \pmod{n}$. Both k and d are smaller than n and $t\epsilon$ is smaller than n (here we use the assumption on the size of r and t), and so the equality $d = k + \epsilon t$ holds in the integers. We can then compute $t = \frac{d-k}{\epsilon}$. The size of t alone gives us a reasonable guess on the size of r as on average $k' \sim n$, and so r should be only slightly larger

than t. To recover the value of r, we need to use the fact that r divides $k + t(n + \epsilon)$. We factor $k + t(n + \epsilon)$ and look for all divisors of similar size as t. Unfortunately, this only gives us a set of candidates for r that cannot be verified. To understand how often we can expect to have just one candidate, we simulated this test for different sizes of the mask r in the range that we saw in the cards: 32, 64, 96, 128, and 160 bits. The probability of a single candidate for r was 10%, 9.8%, 7.2%, 8.4% and 7.8%, respectively for individual bit-lengths of r. Hence, to collect a single mask value, it is expected to be enough to do 13 measurements.

The reason why $n + \epsilon$ is chosen as a prime is to pass any primality tests the implementation can have on the order. Other conditions can be imposed on the value as well. For instance, take the case of the implementation of `Derive` that does not check order primality but checks that the generator vanishes when multiplied by the claimed order $n + \epsilon$. We can then select ϵ such that $n + \epsilon$ and n share a factor f and set $\overline{G}$ as any point of order f. Such a generator will pass the check and the rest of the method remains the same.

7.5.5 Test EPA

The last test leverages the observation of ECTester that a point of small order can be passed to the implementations leading to errors during the scalar multiplication. The occurrence of the errors depends on the scalar and changes with each randomization. Two runs of the same algorithm with the same setting (including a fixed scalar), one with an error and the other without, is clear evidence of randomization. As the input point P, we select a point of small order l so that the point $lP = \mathcal{O}$ sporadically appears during the scalar multiplication regardless of the multiplier, causing an error. Nevertheless, one might accidentally select a point of order l that always/never causes an error for some multipliers so it is better to use several points of different orders. Formally, we will define the result of the test to be the variance in the measured boolean output (error or no error). Nonzero variance indicates that a scalar randomization is present. The opposite (zero variance) does not necessarily mean a lack of randomization. The implementation can handle the infinity point well as discussed in Section 7.4. Another possibility is that the number of measurements or the number of used points was not enough to detect a small variance. This test is applicable only to `Derive` as `Sign` and `KeyGen` contain randomization by design (generation of nonces and private keys).

Note that the name of the test comes from the exceptional procedure attack by Izu et al. [102] which in the same principle invokes errors in the scalar multiplication to gain information about the private key.

Remark. The presented tests were designed to distinguish the behavior of different scalar randomization techniques. All of the tests use correctly defined points on the claimed curve (Table 7.3) and use their order to leak information about the techniques. This is invariant under the coordinate and curve randomization. Hence, the tests will work even if these countermeasures are implemented together with the scalar randomization. Similarly, the point blinding, as described in Section 2.2.1, uses random points on the curve for randomization and not the order n or the cofactor. For any set up of the tests, $\rho(k, P)$ will always correctly compute kP under point blinding and so our tests will be unaffected by point blinding as well. The exception is the EPA test which uses exceptional points of the formulas to produce points not on the curve and consequently errors. The EPA test can reveal point blinding with the same principle as with the scalar randomization.

7.5.6 Scalar randomization on JavaCard smartcards

Table 7.5 presents the results of our reverse-engineering tests on thirteen JavaCard smartcards. Overall we managed to reverse-engineer the scalar randomization on every tested card except for `A1`, which refused the inputs of all five tests. On most of the cards, we managed to recover the randomization technique for all three functions `Derive`, `Sign`, and `KeyGen`, which turned out to all use the same technique. On six of the cards, we recovered the value of the randomization mask. We intentionally did not run all the tests on all the cards to avoid breaking the cards when the randomization technique was already clear, as some of the tests were likely triggering internal fault attack detection mechanisms that were disabling the cards. This was mainly the case of test composite and test EPA. Tests $3n$ and $n + \epsilon$, which on their own were often able to recover the randomization technique and the mask, were causing no errors. Overall, we disabled only four cards during the testing.

Most of the cards use GSR, as the results of test $3n$ follow the uniform distribution $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$ distribution (recall Table 7.4). The GSR was also confirmed by the $k = 10$ test on four cards. `N4` uses multiplicative splitting as it exhibits the $\frac{5}{9}, \frac{2}{9}, \frac{2}{9}$ distribution. There was also evidence

Card	Target	$3n$	Composite	$k = 10$	EPA	ρ	Mask
N1	Derive	$0.34, 0.33, 0.32$	100%	86%	>0	GSR	✗
	Sign	$0.31, 0.31, 0.38$	83%	-		GSR	160
	KeyGen	$0.32, 0.33, 0.35$	100%	-		GSR	160
N2	Derive	$0.37, 0.33, 0.30$	100%	98%		GSR	✗
	Sign	$0.37, 0.31, 0.31$	83%	-		GSR	160
	KeyGen	$0.35, 0.35, 0.31$	100%	-		GSR	160
N3	Derive	$0.33, 0.32, 0.35$	100%	98%		GSR	32
	Sign	$0.31, 0.30, 0.39$	85%	-		GSR	160
	KeyGen	✗	✗	-		✗	✗
N4	Derive	$0.22, 0.56, 0.22$	82%	100%		Mult	64
	Sign	$0.23, 0.23, 0.54$	-	-		Mult	?
	KeyGen	✗	✗	-		✗	✗
N6	Derive	$0, 0, 1$	100%	100%	□	Euc.?	2
	Sign	$0, 0.52, 0.48$	71%	-		Euc.?	2
	KeyGen	$0, 0.51, 0.49$	100%	-		Euc.?	2
N9	Derive	$0.32, 0.34, 0.35$	100%	99%		GSR	✗
	Sign	$0.29, 0.35, 0.35$	84%	-		GSR	160
	KeyGen	$0.34, 0.33, 0.32$	100%	-		GSR	160
I1	Derive	$0.37, 0.32, 0.31$	100%	100%		GSR	✗
	Sign	$0.30, 0.33, 0.37$	92%	-		GSR	✗
	KeyGen	$0.35, 0.30, 0.35$	100%	-		GSR	✗
I2	Derive	$0.31, 0.36, 0.33$	✗	100%		GSR	✗
	Sign	$0.32, 0.32, 0.36$	✗	-		GSR	64
	KeyGen	$0.32, 0.33, 0.35$	✗	-		GSR	32
A1	Derive	✗	✗	100%		✗	✗
	Sign	✗	✗	-		✗	✗
	KeyGen	✗	✗	-		✗	✗
G1	Derive	✗	99%	100%		Mult	✗
	Sign	✗	✗	-		✗	✗
	KeyGen	✗	✗	-		✗	✗
S2	Derive	$1, 0, 0$	100%	100%	0	None	-
	Sign	$1, 0, 0$	80%	-		None	-
	KeyGen	□	□	□	□	□	□
F1	Derive	$0.33, 0.34, 0.33$	100%	100%		GSR	✗
	Sign	$0.33, 0.33, 0.35$	78%	-		GSR	✗
	KeyGen	$0.34, 0.31, 0.36$	100%	-		GSR	✗
F2	Derive	$1, 0, 0$	100%	100%	0	None	-
	Sign	✗	✗	-		✗	✗
	KeyGen	✗	✗	-		✗	✗

Table 7.5: Results of our reverse-engineering of scalar randomization on JavaCards.

of the multiplicative splitting on G1 by the composite test, but the errors did not appear close to the expected frequency and so the result was not conclusive.

The test $3n$ also showed that S2 and F2 do not use scalar randomization at all. EPA test measured high-frequency errors for F2 and N1. In the case of F2, they were constant for fixed inputs, which confirmed no randomization. On N1 the errors were non-deterministic, changing for fixed inputs, confirming the existence of randomization. For both N1 and F2, the errors appeared more frequently than the expected distribution, i.e., for points of order $l = 3$ or $l = 5$ the errors appeared almost always and not with the probability $\frac{1}{l}$. As discussed in Section 7.4 this points to improper handling of the infinity point. On S2 the errors appeared with the expected probability $\frac{1}{l}$.

The results of test $3n$ on N6 point to either Euclidean splitting or no scalar randomization at all. Furthermore, the $n + \epsilon$ mask recovery revealed the implementation adds n to the scalar if the scalar is odd and adds $2n$ otherwise. Initial runs of the EPA test on `Derive` showed some evidence of randomization, but the card soon stopped responding. We concluded that it is most likely the Euclidean splitting, but further measurements would be needed for a definitive answer.

Apart from N6, our tests disabled one N3, one N4 and one S2 card. The symbol ▭ in Table 7.5 denotes that the card stopped responding before we had a chance to run the test.

Mask recovery. For N1, N2, N3, N4, N9 and I2, we recovered the value of the randomization mask using the $n + \epsilon$ test for both the GSR and the multiplicative splitting. The sizes of the masks were the expected 32 and 64 bits (a trade-off between security and efficiency) as well as 160 bits. One explanation for a 160-bit mask is that many standard curves have orders n with special form close to a power of two. GSR blinding with rn for small r might not mask the whole scalar as shown in [75].

Note that on I2 and N3 the sizes of the mask differed between the different functions. The $n + \epsilon$ test failed on `Sign` on N4 for unknown reasons that might be worth investigating further. The $3n$ test shows that multiplicative splitting is used on this card. Recall that if k is the used nonce, d is the discrete logarithm between the generator G and the output point kG then $d = k + \epsilon t$ for some integer t of size similar to the size of the mask. Using this equality, we can recover t and then use it to get the candidate set for the mask. This approach was successful in

`Derive` and showed that the mask has 64 bits. However, for `Sign`, the term $d - k = 92t$ turned out to be a large 256-bit number and often not divisible by 92, making the equality incorrect in this case. Yet, the results showed some signal as $d - k$ was divisible by 92 much more often than a random integer (roughly 12% of time). Perhaps the multiplicative splitting was combined with some other countermeasure that we were not able to recover.

7.6 Impact, limitations and discussion

Impact. Our findings have impacts on several levels. First, it is now possible to mount the learning phase of profiled attacks on black-box ECC targets using scalar randomization via our mask recovery techniques if the targets allow the setting of custom domain parameters. This is the case for the JavaCard platform, as we demonstrated reverse-engineering and mask recovery on several certified smartcards. The learning phase will have to be performed on a different (non-standard) curve than the attack phase, but the curve can be made similar to the target one (same prime, same a parameter). Second, knowledge of scalar randomization countermeasures can no longer be assumed secret in security evaluations of JavaCard smartcards unless they also mitigate the reverse-engineering techniques, including the ones listed in this chapter. This may impact the JIL attack rating of certified (or to be certified) products [183]. Finally, our findings open up possibilities for more reverse-engineering. Perhaps it is possible to reverse-engineer scalar multipliers even if scalar randomization countermeasures are present.

Limitations. While we managed to recover the scalar randomization on almost all of the cards, several of the tests failed. In some cases, the cards explicitly validated the parameters (e.g., the primality of the order). In others, the implementation errored out on invalid inputs and gave us no meaningful information. Moreover, some of the cards likely have an internal fault detection, as repeated errors caused the cards to stop responding temporarily or even permanently. We encountered this behavior for the composite and EPA tests, but not for $3n$ and $n + \epsilon$.

The $n + \epsilon$ test is capable of fully recovering the mask value for GSR. However, for the multiplicative splitting, the test only offers a set of candidates, any of which can be the true masking value. In roughly 10% of the cases, there is only one (true) candidate.

One of the main limitations of our work is the presence of a combination of countermeasures. We have designed individual tests to distinguish the four scalar randomization techniques. Any implementation that uses two or more together may fool our tests. This might have been the case for the results on N4 with `Sign`. The $3n$ test indicated the multiplicative splitting, but the $n + \epsilon$ test gave nonsensical results for the mask.

We have implemented countermeasures composed of two scalar randomization techniques (described in our repository) using pyecsca. This amounted to 15 combinations, since each pair of the four main techniques could be combined in multiple ways. This is not an exhaustive list. We only aim to provide a better understanding of our limitations and to possibly open up paths for future work. We ran test $3n$, test composite and test $k = 10$ on each combination with the results in Appendix B.3. For most combinations, test $3n$ matched the expected distribution of one underlying technique. For instance, the test on all three combinations of GSR and the multiplicative splitting (GM-1, GM-2, GM-3) output the distribution $\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$ — expected for the multiplicative splitting. There were two exceptions though. First, one combination of the additive and multiplicative splitting slightly deviated from $\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$. Closer examination showed that it rather follows the distribution $\frac{8}{27}, \frac{8}{27}, \frac{11}{27}$. Second, one combination of GSR and Euclidean splitting followed the distribution $\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$. In all cases, the composite test correctly detected multiplicative splitting, so these two tests together either rule out a single technique or identify one of them. Test $k = 10$ showed no clear pattern.

Mitigations. Our reverse-engineering methods observe the behavior of the implementations under invalid inputs. An obvious countermeasure is proper validation of the domain parameters and inputs, as suggested in [185]. Both the prime p defining the base field and the prime order n need to be verified by a primality test to avoid composite order curves. The generator G and any point must be checked that they belong on the curve and their order is n. The correct way to do so is to multiply and compare $nG = \mathcal{O}$. Since n is prime, this implies that G must have the correct order, and the curve truly contains a subgroup of order n. The order of the public key points should be verified as well to avoid any points outside of the main subgroup. Other solutions to cofactor validation, like cofactor Diffie-Hellman or the inverse trick, rely on the assumption that the cofactor is correct. This precise assumption by implementations is violated by our inputs and made most of our tests work. Specifically, tests

$3n$, $n + \epsilon$, and composite concealed the true cofactor, used points outside of the claimed group, and composite curve order n. When implemented, the mentioned checks would make these tests ineffective.

Primality testing and domain parameter validation of custom curves come with a cost in performance. A clear solution is to restrict the API to standard named curves as they will not contain hidden small subgroups. This avoids costly checks and these curves are optimized for higher performance. Popular examples are the prime order NIST curves or the cofactor curves Curve25519 and Ed25519. Note that this must be a strict API-level limitation and not just an option (as for JavaCard API since version 3.1), since an attacker will simply opt for optional setting of custom domain parameters to make our techniques work.

The EPA and $k = 10$ tests relied on improper handling of the infinity point by the target during scalar multiplication. Not every formula computing point addition $P+Q$ is prepared for special cases like $P = \pm Q$ or $P = \mathcal{O}$, which results in an undefined output that causes an error. One solution is for the implementation to check for these special cases and solve them separately. This might involve dummy operations to preserve constant-time execution. A more straightforward, and secure, solution is to use complete formulas [170]. Note that even these formulas contain exceptional points that are outside of the main subgroup, and so they must be used together with cofactor validation.

It must be stressed that the described mitigations will only thwart methods presented in this chapter and in their current form. Reverse engineering of countermeasures, in general, likely cannot be fully prevented, as evidenced also by the Side Journey to Titan [173] and EUCLEAK papers [172]. The assumption that implementation details can be kept secret relies on security by obscurity in contrary to Kerckhoff's principle. While obscurity typically increases the difficulty of mounting an attack in the short term due to an additional layer of uncertainty about the implementation, it also decreases availability for public scrutiny. The lack of public audits is possibly harming the security in the long run — a timeframe especially relevant for cryptographic smartcards used as, e.g., electronic IDs with the expected lifetime of a decade or more. The advent of open-source cryptographic hardware designs like LowRISC's OpenTitan [156] or TropicSquare's TROPIC01 [206] may offer interesting tests of the feasibility of open-source secure hardware design.

8 Conclusions

This chapter concludes the thesis by synthesizing the main findings and addressing the research questions stated in the introduction. We summarize the insights gained from the analysis of elliptic curve properties, implementations, and attacks, and discuss their implications for the practical security of ECC.

It is probably not surprising that the vision articulated by Miller and Koblitz of a "fertile ground" for cryptographic constructions has evolved into a landscape of remarkable depth and complexity (**RQ I**). What the results of Chapters 3 through 7 provide, however, are concrete measurements of this complexity. Over more than two decades, standardization efforts have produced over 200 elliptic curves recommended for public use (Chapter 3) from 18 different sources. The selection processes and diversity of properties have caused widespread distrust and the belief that additional structure may be hidden. Beyond curve choice, the design space includes nearly 300 addition formulas across various coordinate systems and curve models, whose resistance to side-channel attacks differs significantly, as shown in Chapter 4. In Chapter 5, we further pile on the already extensive list of scalar multiplication algorithms by introducing a new verification algorithm that exploits fast endomorphisms to improve performance. The analysis of open-source libraries in Chapter 6 demonstrates that the diversity of ECC is not confined to academic literature, but is actively reflected in real-world implementations. Faced with this expansive design space, developers rarely restrict themselves to a small set of options; instead, they combine and adapt formulas, coordinates, and algorithms, expanding the space to an estimated 140 000 distinct ECC implementations. Chapter 7 further extends this landscape by examining input validation and side-channel countermeasures on smartcards.

There are several explanations for this diversity. It enables performance optimizations across various platforms and supports a wide range of protocols with different requirements. Furthermore, it provides flexibility in response to newly discovered weaknesses. As shown throughout this thesis, the resulting complexity has concrete consequences for practical

security (**RQ II**). Chapter 3 demonstrates that even standardized curves may exhibit undocumented or poorly understood properties, calling into question the trust placed in their selection. Chapters 4 and 5 make clear that the diversity of the ECC landscape is a double-edged sword: using fast endomorphisms speeds up signature verification, yet at the same time increases vulnerability to side-channel attacks. Similarly, the choice of addition formulas has a direct impact on side-channel resistance, but this depends on the curve, linking implementation security back to the curve selection examined in Chapter 3. Chapters 6 and 7 examine the impact of implementation diversity on black-box ECC implementations. In particular, they investigate whether the diversity combined with obscurity improves resistance against side-channel attacks, which often assume detailed knowledge of the target. The results answer this question in the negative. Chapter 6 demonstrates this by transforming the ZVP attacks studied in Chapter 4 and Chapter 5 into reverse-engineering methods. Chapter 7 demonstrates that improper input validation and overly permissive APIs allow attackers to recover implementation details without side-channel measurements. A central conclusion of these results is that ECC implementations should favor more restrictive and carefully validated frameworks, rather than exposing overly rich or flexible APIs.

Finally, these results should be placed in the broader context of the long-term future of ECC. While large-scale quantum computers would fundamentally break ECC through Shor's algorithm [192], their practicality remains uncertain. ECC continues to provide strong security guarantees in the classical setting for the foreseeable future. At the same time, the experience of ECC has already shaped the transition to post-quantum cryptography. A telling example is the post-quantum protocol SIKE [10], which was itself built on the famously complex elliptic curve isogenies. SIKE illustrated both the promise and the peril of carrying rich algebraic structure into new cryptographic settings: while elegant and compact, an unexpected structural attack led to its complete break [47]. This mirrors, in a different setting, the central theme of this thesis that additional structure can be a powerful enabler of efficiency, but also a source of vulnerability.

As cryptography migrates toward post-quantum primitives, the challenges explored in this work remain highly relevant. Security will depend not only on asymptotic hardness assumptions, but equally on careful parameter selection, robust implementation, and constrained interfaces that limit how algebraic structure can be exploited.

Appendices

A Academic publications

This appendix contains the list of publications that formed the basis of this dissertation. In each of these works, I served either as the primary author or as a major contributing (second) author. Each publication includes an estimated percentage of my contribution, along with a qualitative description of my role.

***Publication 1*: DiSSECT: Distinguisher of Standard and Simulated Elliptic Curves via Traits**

- Published at AfricaCrypt 2022: International Conference on Cryptology in Africa. (GGS B).
- Author's contribution: 40% (total number of authors: 5). I reviewed and analyzed the relevant ECC standards. I implemented the curve generation and carried out the data collection. The analysis of the results and the writing of the paper were performed collaboratively with the co-authors.

Bibliographic reference: V. Sedlacek, V. Suchanek, A. Dufka, M. Sys, and V. Matyas. "DiSSECT: Distinguisher of Standard and Simulated Elliptic Curves via Traits". In: *Progress in Cryptology – AFRICACRYPT 2022*. Ed. by L. Batina and J. Daemen. Springer Nature Switzerland, Cham, pp. 493–517. ISBN: 978-3-031-17433-9.

***Publication 2*: Decompose and conquer: ZVP attacks on GLV curves**

- Published at ACNS 2025: Applied Cryptography and Network Security (CORE B).
- Author's contribution: 60% (total number of authors: 3). The initial idea to exploit small endomorphisms originated from Sorina Ionica and was refined through joint discussions and preliminary investigations with the co-authors. I designed and implemented the attacks presented

in the paper. I developed the tool used for formula analysis and analyzed the results. I wrote the paper, with input and revisions from the co-authors.

Bibliographic reference: V. Suchanek, V. Sedlacek, and M. Sys. "Decompose and Conquer: ZVP Attacks on GLV Curves". In: *Applied Cryptography and Network Security: 23rd International Conference, ACNS 2025, Munich, Germany, June 23–26, 2025, Proceedings, Part II*. Springer-Verlag, Munich, Germany, 2025, pp. 49–73. ISBN: 978-3-031-95763-5. DOI: 10.1007/978-3-031-95764-2_3 .

Publication 3: Faster signature verification with 3-dimensional decomposition

- Accepted to ACNS 2026: Applied Cryptography and Network Security (CORE B).
- Author's contribution: 70% (total number of authors: 3). I conceived the main idea and designed all algorithms for signature verification. The lattice reduction algorithms were developed collaboratively with the co-authors. I implemented all the algorithms and performed the measurements. I wrote the paper, with input from the co-authors.

Bibliographic reference: V. Suchanek, M. Sys, and Ł. Chmielewski. *Faster signature verification with 3-dimensional decomposition*. Accepted to: International Conference on Applied Cryptography and Network Security – ACNS 2026.

Publication 4: pyecsca: Reverse engineering black-box elliptic curve cryptography via side-channel analysis

- Published at CHES 2024: Transactions on Cryptographic Hardware and Embedded Systems (CORE A).
- Author's contribution: 30% (total number of authors: 5). Most of the tool was developed prior to my involvement. The analysis of libraries and the design of the techniques was done collaboratively with Ján Jančár. The paper was written by all authors.

Bibliographic reference: J. Jancar, V. Suchanek, P. Svenda, V. Sedlacek, and Ł. Chmielewski. "pyecsca: Reverse engineering black-box elliptic curve cryptography via side-channel analysis". *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024:4, 2024, pp. 355–381. DOI: 10.46586/TCHES.V2024.I4.355-381 .

***Publication 5*: ECTester: Reverse-engineering side-channel countermeasures of ECC implementations**

- Published at CHES 2025: Transactions on Cryptographic Hardware and Embedded Systems (CORE A).
- Author's contribution: 35% (total number of authors: 5). The original ECTester tool was implemented prior to my involvement. I performed the testing of the libraries and smartcards presented in the paper, with significant input from Ján Jančár. Based on the obtained results, I designed the reverse-engineering techniques and carried out the measurements. Their implementation and integration was done together with Ján Jančár. The overall discussion and writing of the manuscript were conducted collaboratively by all authors.

Bibliographic reference: V. Suchanek, J. Jancar, J. Kvapil, P. Svenda, and Ł. Chmielewski. "ECTester: Reverse-engineering side-channel countermeasures of ECC implementations". *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025:4, 2025, pp. 290–316.

B Supplementary Algorithms, Tables, and Figures

B.1 DiSSECT: List of traits

The following is a list of all traits used for curve analysis in Chapter 3. Each trait takes as an input an ordinary elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + ax + b$ of order n.

Trait name	**Input**	**Output**
`cofactor`	int r	Tuple (n_1, n_2) such that the group $E(\mathbb{F}_{p^r})$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ and $n_1 \mid n_2$ ($n_1 = 1$ for cyclic groups).
`discriminant`		The factorization of $D = t^2 - 4p = v^2 d_K$, where d_K is the discriminant of the endomorphism algebra of E.
`twist order`	int r	The factorization of the cardinality of the quadratic twist of $E(\mathbb{F}_{p^r})$.
`kn fact.`	int k	The factorizations of $kn + 1$, $kn - 1$
`torsion extension`	prime l	$k_1, k_2, k_2/k_1$, where k_1, k_2 are the smallest integers satisfying $E[l] \cap E(\mathbb{F}_{p^{k_1}}) \neq \emptyset$ and $E[l] \subseteq E(\mathbb{F}_{p^{k_2}})$.
`conductor`	int r	The factorization of D_r/D_1, where $D_r = t_r^2 - 4p^r$ and t_r is the trace of Frobenius of $E/\mathbb{F}_{p^r}$.
`embedding`		The ratio $\phi(r)/e$, where $r \mid n$ is the order of the prime order subgroup and e is the multiplicative order of $q \pmod r$.
`class number`		Upper bound [66] and lower bound [154] on the class number of the endomorphism algebra of E.

`small prime order`	prime l	The ratio $\phi(n)/m$ where m is the multiplicative order of $l \pmod n$.
`division pol.`	prime l	The factorization of the l-th division polynomial.
`volcano`	prime l	The depth and the degree of the l-volcano.
`isogeny extension`	prime l	$i_1, i_2, i_2/i_1$ where i_1, i_2 are the smallest integers such that there exists a $\mathbb{F}_{p^{i_1}}$-rational l-isogeny and $l+1$ $\mathbb{F}_{p^{i_2}}$-rational l-isogenies from E.
`trace fact.`	int r	A factorization of the trace of Frobenius of $E/\mathbb{F}_{p^r}$.
`isogeny neighbors`	prime l	A number of roots of $\Phi_l(j(E), x)$ where Φ_l is the l-th modular polynomial.
`q torsion`		Torsion order of $E'(\mathbb{Q})$ where E' is is given by the same equation $y^2 = x^3 + ax + b$.
`hamming x`	int k	A number of points on E with the Hamming weight of the x-coordinate equal to k.
`square 4p-1`		The factorization of square-free parts of $4p-1$ and $4n-1$ where n is the order of the generator point of E
`pow distance`		The distances of n to the nearest power of 2 and multiples of 32 and 64.
`multiples x`	int k	The x-coordinate of $\frac{1}{k}G$.
`x962 inv.`		The value $r = \frac{a^3}{b^2}$.
`brainpool overlap`		$a_{160} - b_{-160}$ where a_{160} are the $s-160$ rightmost bits of a and b_{-160} are the $s-160$ leftmost bits of b.
`weierstrass`		The parameters a, b.

B.2 Algorithms and figures for Chapter 4

Algorithm 8: ZVP-GLV attack on the Straus-Shamir trick.

input : A set K_{i+1} of candidates pairs for $d_0^{\geq i+1}, d_1^{\geq i+1}$
output : A set K_i of candidate pairs for $d_0^{\geq i}, d_1^{\geq i}$

$K_i, U_i \leftarrow \{\}, \{\}$
foreach $(\kappa_0, \kappa_1) \in K_{i+1}$ **do**
  for $(g_0, g_1) \in \{(1,0), (-1,0), (1,1), (-1,-1)\}$ **do**
    $D \leftarrow \mathrm{DCP}_f(g_0\kappa_0 + \lambda(g_0\kappa_1 - g_1\kappa_0))$
    Find a solution point P to D using Lemma 1
    if no solution **then** add $(2\kappa_0 + g_0, 2\kappa_1 + g_1)$ to U_i
    else if $\mathcal{O}(P) = 1$ **then** add $(2\kappa_0 + g_0, 2\kappa_1 + g_1)$ to K_i
if K_i is empty **then return** U_i **else return** K_i

Algorithm 9: ZVP-GLV attack on the regular w-NAF interleaving.

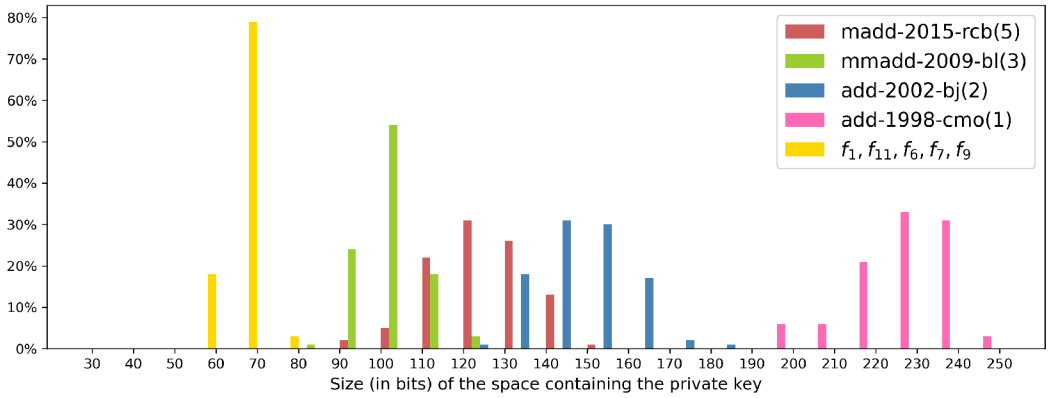
input : A set K_{i+1} of candidates pairs for $d_0^{\geq i+1}, d_1^{\geq i+1}$
output : A set K_i of candidate pairs for $d_0^{\geq i}, d_1^{\geq i}$

$K_i, U_i \leftarrow \{\}, \{\}$
foreach $(\kappa_0, \kappa_1) \in K_{i+1}$ **do**
  for $g_0 \in \{\pm 1, \ldots, \pm(2^w - 1)\}$ **do**
    $D \leftarrow \mathrm{DCP}_f(2^w\kappa_0 + 2^w\lambda\kappa_1, g_0)$
    Find a solution point P to D using Lemma 1
    if no solution **then** add $(2^w\kappa_0 + g_0, 2^w\kappa_1)$ to U_i
    else if $\mathcal{O}(P) = 1$ **then** add $(2^w\kappa_0 + g_0, 2^w\kappa_1)$ to K_i
if K_i is empty **then return** U_i **else return** K_i

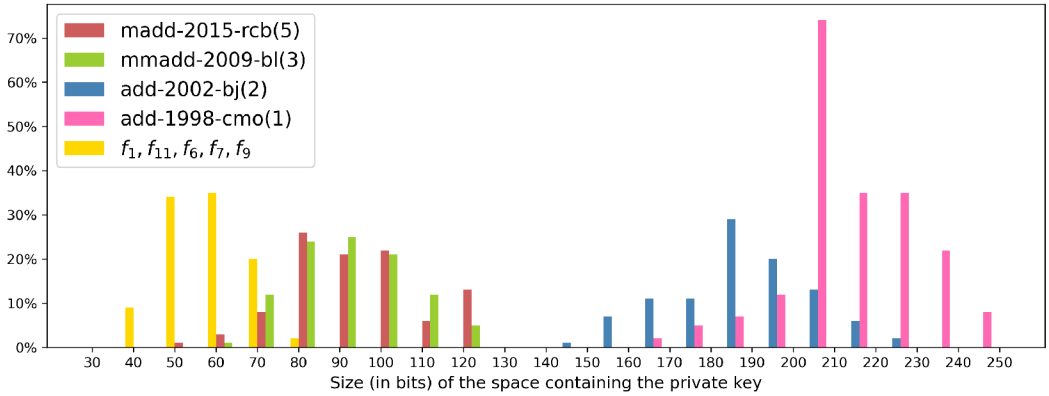
Algorithm 10: ZVP-GLV attack on regular w-NAF interleaving.

input : Oracle $\mathcal{O}^*$, integer t, polynomial f
output : A set C_i of candidates pairs of w-NAF bits $d_0^{(i)}, d_1^{(i)}$

$\mathcal{P}, \mathcal{G} \leftarrow \{\}, \{\pm 1, \ldots, \pm(2^w - 1)\}^2$
for $(g_0, g_1) \in \mathcal{G}$ **do** put in $\mathcal{P}$ any solution point P to $\mathrm{DCP}_f(g_0, g_1)$
for $P \in \mathcal{P}$ **do** $\mathcal{T}_P \leftarrow \{(g_0, g_1) \in \mathcal{G} \mid P \text{ solves } \mathrm{DCP}_f(g_0, g_1)\}$
for $i = 0$ *to* $l - 1$ **do**
  $C_i \leftarrow \mathcal{G}$
  for $P \in \mathcal{P}$ **do**
    if $\mathcal{O}^*(P)_i = 1$ **then** $C_i \leftarrow C_i \cap \mathcal{T}_P$ **else** $C_i \leftarrow C_i \setminus \mathcal{T}_P$
return $C_0, \ldots, C_{l-1}$



(a) $w = 3$



(b) $w = 5$

Figure B.1: Histogram of the number of unknown bits after the ZVP-GLV attack on the regular 3-NAF and 5-NAF interleaving algorithm. The attack was performed on 100 scalars for four different addition formulas and one fictitious formula containing IV polynomials with a high success rate.

B.3 Tables for Chapter 7

Library	Supported versions	Evaluated version
BoringSSL	r76bb141–r2eb2889, rde43457–r2be18f5	r67422ed
Botan	2.0.0–2.19.5	2.19.1
BouncyCastle	1.69–1.80	1.77
Crypto++	6.1.0–8.9.0	8.6.0
Intel Crypto	2020–2021.2, 2021.5–2021.12.1	2021.7
libgcrypt	1.8.0–1.10.3	1.9.4
LibreSSL	2.6.2–4.0.0	3.9.0
libtomcrypt	1.18.0-rc4–1.18.2	1.18.2
mbedTLS	2.7.19–3.6.2	3.5.2
Nettle	3.1.1, 3.{4, 5, 8}.1, 3.7.{1, 2, 3}	3.7.3
OpenSSL	1.1.0–1.1.0j, 1.1.1–3.4.1	3.2.0-dev
SunEC	OpenJDK 15–23	17

Table B.2: The version ranges of libraries currently supported by ECTester. The third column shows the versions used for the results presented in Chapter 7. As BoringSSL is not versioned we test roughly every 40^{th} commit, for about 100 times.

Combination	Test $3n$	Test composite	Test $k = 10$
GA-1, GA-2	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	100%	100%
GM-1, GM-2, GM-3	$\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$	91%	100%
GE-1	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	100%	100%
GE-2	$\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$	100%	0%
GE-3, GE-4	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	100%	0%
AM-1	$\frac{8}{27}, \frac{8}{27}, \frac{11}{27}$	84%	100%
AM-2	$\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$	90%	100%
AE-1	$\frac{1}{2}, \frac{1}{2}, 0$	100%	100%
AE-2	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	100%	100%
ME-1	$\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$	91%	46%
ME-2	$\frac{2}{9}, \frac{2}{9}, \frac{5}{9}$	84%	0%

Table B.3: The results of our RE tests on the combinations of randomization techniques. The precise description of the algorithms can be found in our repository. Each algorithm is labeled XY-N, where X,Y are the first letters of the four used techniques (GSR, Additive, Multiplicative and Euclidean splitting) and N is a numerical identifier.

Bibliography

Authored publications

VS1. J. Jancar, V. Suchanek, P. Svenda, V. Sedlacek, and Ł. Chmielewski. "pyecsca: Reverse engineering black-box elliptic curve cryptography via side-channel analysis". *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024:4, 2024, pp. 355–381. DOI: 10.46586/TCHES.V2024.I4.355-381.

VS2. V. Sedlacek, V. Suchanek, A. Dufka, M. Sys, and V. Matyas. "DiSSECT: Distinguisher of Standard and Simulated Elliptic Curves via Traits". In: *Progress in Cryptology – AFRICACRYPT 2022*. Ed. by L. Batina and J. Daemen. Springer Nature Switzerland, Cham, pp. 493–517. ISBN: 978-3-031-17433-9.

VS3. V. Suchanek, J. Jancar, J. Kvapil, P. Svenda, and Ł. Chmielewski. "ECTester: Reverse-engineering side-channel countermeasures of ECC implementations". *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025:4, 2025, pp. 290–316.

VS4. V. Suchanek, V. Sedlacek, and M. Sys. "Decompose and Conquer: ZVP Attacks on GLV Curves". In: *Applied Cryptography and Network Security: 23rd International Conference, ACNS 2025, Munich, Germany, June 23–26, 2025, Proceedings, Part II*. Springer-Verlag, Munich, Germany, 2025, pp. 49–73. ISBN: 978-3-031-95763-5. DOI: 10.1007/978-3-031-95764-2_3.

VS5. V. Suchanek, M. Sys, and Ł. Chmielewski. *Faster signature verification with 3-dimensional decomposition*. Accepted to: International Conference on Applied Cryptography and Network Security – ACNS 2026.

External references

1. T. Akishita and T. Takagi. "Zero-Value Point Attacks on Elliptic Curve Cryptosystem". In: *Information Security*. Ed. by C. Boyd and W. Mao. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 218–233. ISBN: 978-3-540-39981-0.
2. E. K. Alekseev, V. Nikolaev, and S. V. Smyshlyaev. "On the security properties of Russian standardized elliptic curves". *Mathematical questions of cryptography* 9:3, 2018, pp. 5–32.
3. F. Amiel, B. Feix, and K. Villegas. "Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms". In: *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*. Ed. by C. M. Adams, A. Miri, and M. J. Wiener. Vol. 4876. Lecture Notes in Computer Science. Springer, 2007, pp. 110–125. DOI: 10.1007/978-3-540-77360-3_8.
4. ANSI. *American National Standard X9.62-1998, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)*. Preliminary draft. Accredited Standards Committee X9, 1998.
5. ANSI. *American National Standard X9.62-2005, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)*. Standard. Accredited Standards Committee X9, 2005.
6. A. Antipa, D. Brown, R. Gallant, R. Lambert, R. Struik, and S. Vanstone. "Accelerated Verification of ECDSA Signatures". In: *Selected Areas in Cryptography*. Ed. by B. Preneel and S. Tavares. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 307–318. ISBN: 978-3-540-33109-4.
7. A. Antipa, D. R. L. Brown, A. Menezes, R. Struik, and S. A. Vanstone. "Validation of Elliptic Curve Public Keys". In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. Ed. by Y. Desmedt. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 211–223. DOI: 10.1007/3-540-36288-6_16.

8. D. F. Aranha, P. S. L. M. Barreto, G. C. C. F. Pereira, and J. E. Ricardini. *A note on high-security general-purpose elliptic curves*. Cryptology ePrint Archive, Report 2013/647. https://eprint.iacr.org/2013/647. 2013.
9. C. N. Arguello, H. Searle, S. Rampazzi, and K. R. B. Butler. "A Practical Methodology for ML-Based EM Side Channel Disassemblers". *CoRR* abs/2206.10746, 2022. DOI: 10.48550/arXiv.2206.10746.
10. R. Azarderakhsh, M. Campagna, C. Costello, L. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, et al. "Supersingular isogeny key encapsulation". *Submission to the NIST Post-Quantum Standardization project*, 2017.
11. L. Babai. "On Lovász' lattice reduction and the nearest lattice point problem". *Combinatorica* 6, 1986, pp. 1–13.
12. D. Bae and J. Ha. "Implementation of Disassembler on Microcontroller Using Side-Channel Power Consumption Leakage". *Sensors* 22:15, 2022, p. 5900. DOI: 10.3390/S22155900.
13. T. Baignères, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain. *Trap Me If You Can – Million Dollar Curve*. Cryptology ePrint Archive, Paper 2015/1249. 2015. URL: https://eprint.iacr.org/2015/1249.
14. D. V. Bailey, L. Batina, D. J. Bernstein, P. Birkner, J. W. Bos, H.-C. Chen, C.-M. Cheng, G. van Damme, G. de Meulenaer, L. J. D. Perez, J. Fan, T. Güneysu, F. Gurkaynak, T. Kleinjung, T. Lange, N. Mentens, R. Niederhagen, C. Paar, F. Regazzoni, P. Schwabe, L. Uhsadel, A. V. Herrewege, and B.-Y. Yang. *Breaking ECC2K-130*. Cryptology ePrint Archive, Paper 2009/541. 2009. URL: https://eprint.iacr.org/2009/541.
15. P. S. L. M. Barreto, B. Lynn, and M. Scott. "Constructing Elliptic Curves with Prescribed Embedding Degrees". In: *Security in Communication Networks*. Ed. by S. Cimato, G. Persiano, and C. Galdi. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 257–267. ISBN: 978-3-540-36413-9.
16. L. Batina, Ł. Chmielewski, B. Haase, N. Samwel, and P. Schwabe. "SoK: SCA-secure ECC in software – mission impossible?" *IACR TCHES* 2023:1, 2023, pp. 557–589. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i1.557-589.

17. A. Bauer, É. Jaulmes, E. Prouff, J. Reinhard, and J. Wild. "Horizontal collision correlation attack on elliptic curves – Extended Version". *Cryptogr. Commun.* 7:1, 2015, pp. 91–119. DOI: 10.1007/s12095-014-0111-8.

18. R. Bellman and E. Straus. "Addition chains of vectors (problem 5125)". *The American Mathematical Monthly* 71:7, 1964, pp. 806–808.

19. D. Belyavsky, B. B. Brumley, J. Chi-Dominguez, L. Rivera-Zamarripa, and I. 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*. ACM, 2020, pp. 760–771. DOI: 10.1145/3427228.3427291.

20. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. "High-speed high-security signatures". *Journal of Cryptographic Engineering* 2:2, 2012, pp. 77–89. ISSN: 2190-8516. DOI: 10.1007/s13389-012-0027-1.

21. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. "High-speed high-security signatures". *Journal of cryptographic engineering* 2:2, 2012, pp. 77–89.

22. D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. "Elligator: Elliptic-curve points indistinguishable from uniform random strings". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 967–980.

23. D. J. Bernstein, T. Lange, and R. Niederhagen. "Dual EC: A standardized back door". In: *The New Codebreakers*. Springer, 2016, pp. 256–281.

24. D. J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006*. Ed. by M. Yung, Y. Dodis, A. Kiayias, and T. Malkin. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 207–228. ISBN: 978-3-540-33852-9.

25. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. "Twisted Edwards Curves". In: *Progress in Cryptology – AFRICACRYPT 2008*. Ed. by S. Vaudenay. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 389–405. ISBN: 978-3-540-68164-9.

26. D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, E. Lambooij, T. Lange, R. Niederhagen, and C. Vredendaal. "How to Manipulate Curve Standards: A White Paper for the Black Hat http://bada55.cr.yp.to". In: *Proceedings of the Second International Conference on Security Standardisation Research - Volume 9497*. SSR 2015. Springer-Verlag, Tokyo, Japan, 2015, pp. 109–139. ISBN: 9783319271514. DOI: 10.1007/978-3-319-27152-1_6.

27. D. J. Bernstein, C. Chuengsatiansup, and T. Lange. "Curve41417: Karatsuba Revisited". In: *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems — CHES 2014 - Volume 8731*. Springer-Verlag, Berlin, Heidelberg, 2014, pp. 316–334. ISBN: 9783662447086. DOI: 10.1007/978-3-662-44709-3_18.

28. D. J. Bernstein and T. Lange. *Explicit-Formulas Database*. 2007. URL: https://hyperelliptic.org/EFD/ (visited on 08/03/2026).

29. D. J. Bernstein and T. Lange. *SafeCurves: choosing safe curves for elliptic-curve cryptography*. 2017. URL: https://safecurves.cr.yp.to/ (visited on 08/03/2026).

30. D. J. Bernstein and T. Lange. "Safe curves for elliptic-curve cryptography". *IACR Cryptol. ePrint Arch.*, 2024, p. 1265. URL: https://eprint.iacr.org/2024/1265.

31. I. Biehl, B. Meyer, and V. Müller. "Differential Fault Attacks on Elliptic Curve Cryptosystems". In: *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*. Ed. by M. Bellare. Vol. 1880. Lecture Notes in Computer Science. Springer, 2000, pp. 131–146. DOI: 10.1007/3-540-44598-6_8.

32. E. Biham and O. Dunkelman. "Cryptanalysis of the A5/1 GSM Stream Cipher". In: *Progress in Cryptology —INDOCRYPT 2000*. Ed. by B. Roy and E. Okamoto. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 43–51. ISBN: 978-3-540-44495-4.

33. B. Black, J. Bos, C. Costello, P. Longa, and M. Naehrig. *Elliptic Curve Cryptography (ECC) Nothing Up My Sleeve (NUMS) Curves and Curve Generation*. Internet-Drafts are working documents of the Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-black-numscurves-02. 2014. (Visited on 08/03/2026).

34. D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing". In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by C. Boyd. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 514–532. ISBN: 978-3-540-45682-7.
35. A. D. Booth. "A signed binary multiplication technique". *The Quarterly Journal of Mechanics and Applied Mathematics* 4:2, 1951, pp. 236–240. ISSN: 0033-5614. DOI: 10.1093/qjmam/4.2.236.
36. J. W. Bos, C. Costello, H. Hisil, and K. Lauter. "High-performance scalar multiplication using 8-dimensional GLV/GLS decomposition". In: *Cryptographic Hardware and Embedded Systems-CHES 2013: 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings 15*. Springer. 2013, pp. 331–348.
37. S. Bowe. *BLS12-381: New zk-SNARK Elliptic Curve Construction*. https://electriccoin.co/blog/new-snark-curve/. 2017. (Visited on 08/03/2026).
38. S. Bowe, J. Grigg, and D. Hopwood. *Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. https://ia.cr/2019/1021. 2019. (Visited on 08/03/2026).
39. M. Brengel and C. Rossow. "Identifying Key Leakage of Bitcoin Users". In: *Research in Attacks, Intrusions, and Defenses*. Ed. by M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis. Springer International Publishing, Cham, 2018, pp. 623–643.
40. M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.
41. R. Bröker. "Constructing elliptic curves of prescribed order". PhD thesis. Thomas Stieltjes Institute for Mathematics, 2006.
42. D. R. Brown. *The exact security of ECDSA*. Faculty of Mathematics, University of Waterloo, 2000.
43. D. R. Brown. "Generic groups, collision resistance, and ECDSA". *Designs, Codes and Cryptography* 35:1, 2005, pp. 119–152.

44. B. B. Brumley. "Faster software for fast endomorphisms". In: *Constructive Side-Channel Analysis and Secure Design: 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers 6*. Springer. 2015, pp. 127–140.

45. B. B. Brumley, M. Barbosa, D. Page, and F. Vercauteren. "Practical Realisation and Elimination of an ECC-Related Software Bug Attack". In: *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*. Ed. by O. Dunkelman. Vol. 7178. Lecture Notes in Computer Science. Springer, 2012, pp. 171–186. DOI: 10.1007/978-3-642-27954-6_11.

46. C2SP. *Wycheproof*. 2016. URL: https://github.com/C2SP/wycheproof (visited on 08/03/2026).

47. W. Castryck and T. Decru. "An Efficient Key Recovery Attack on SIDH". In: *Advances in Cryptology – EUROCRYPT 2023*. Ed. by C. Hazay and M. Stam. Springer Nature Switzerland, Cham, 2023, pp. 423–447. ISBN: 978-3-031-30589-4.

48. Certicom Research. *Test Vectors for SEC 1*. Standards for Efficient Cryptography Group. 1999. URL: http://rfc.nop.hu/secg/gec2.pdf (visited on 08/03/2026).

49. Certicom Research. *Standards for Efficient Cryptography Group*. https://secg.org/. (Visited on 08/03/2026).

50. K. Chalkias, F. Garillot, and V. Nikolaenko. "Taming the Many EdDSAs". In: *Security Standardisation Research - 6th International Conference, SSR 2020, London, UK, November 30 - December 1, 2020, Proceedings*. Ed. by T. van der Merwe, C. J. Mitchell, and M. Mehrnezhad. Vol. 12529. Lecture Notes in Computer Science. Springer, 2020, pp. 67–90. DOI: 10.1007/978-3-030-64357-7_4.

51. S. Checkoway, J. Maskiewicz, C. Garman, J. Fried, S. Cohney, M. Green, N. Heninger, R. Weinmann, E. Rescorla, and H. Shacham. "A Systematic Analysis of the Juniper Dual EC Incident". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pp. 468–479.

52. J. H. Cheon. "Security Analysis of the Strong Diffie-Hellman Problem". In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by S. Vaudenay. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 1–11. ISBN: 978-3-540-34547-3.

53. D. Chudnovsky and G. Chudnovsky. "Sequences of Numbers Generated by Addition in Formal Groups and New Primality and Factorization Tests". 7:4, 1986, pp. 385–434. ISSN: 0196-8858. DOI: 10.1016/0196-8858(86)90023-0.

54. D. V. Chudnovsky and G. V. Chudnovsky. "Sequences of numbers generated by addition in formal groups and new primality and factorization tests". *Advances in Applied Mathematics* 7:4, 1986, pp. 385–434.

55. M. Ciet and M. Joye. "(Virtually) Free Randomization Techniques for Elliptic Curve Cryptography". In: *Information and Communications Security, 5th International Conference, ICICS 2003, Huhehaote, China, October 10-13, 2003, Proceedings*. Ed. by S. Qing, D. Gollmann, and J. Zhou. Vol. 2836. Lecture Notes in Computer Science. Springer, 2003, pp. 348–359. DOI: 10.1007/978-3-540-39927-8_32.

56. C. Clavier. *Side Channel Analysis for Reverse Engineering (SCARE) – An Improved Attack Against a Secret A3/A8 GSM Algorithm*. Cryptology ePrint Archive, Report 2004/049. https://eprint.iacr.org/2004/049. 2004.

57. C. Clavier, Q. Isorez, D. Marion, and A. Wurcker. "Complete reverse-engineering of AES-like block ciphers by SCARE and FIRE attacks". *Cryptogr. Commun.* 7:1, 2015, pp. 121–162. DOI: 10.1007/s12095-014-0112-7.

58. C. Clavier and M. Joye. "Universal Exponentiation Algorithm". In: *Cryptographic Hardware and Embedded Systems – CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*. Ed. by Ç. K. Koç, D. Naccache, and C. Paar. Vol. 2162. Lecture Notes in Computer Science. Springer, 2001, pp. 300–308. DOI: 10.1007/3-540-44709-1_25.

59. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography, Second Edition*. Chapman & Hall/CRC, 2012. ISBN: 1439840008.

60. H. Cohen, A. Miyaji, and T. Ono. "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates". In: *Advances in Cryptology — ASIACRYPT'98*. Ed. by K. Ohta and D. Pei. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 51–65. ISBN: 978-3-540-49649-6.

61. J. Coron. "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems". In: *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*. Ed. by Ç. K. Koç and C. Paar. Vol. 1717. Lecture Notes in Computer Science. Springer, 1999, pp. 292–302. DOI: 10.1007/3-540-48059-5_25.

62. C. Costello, P. Longa, and M. Naehrig. *A brief discussion on selecting new elliptic curves*. https://www.microsoft.com/en-us/research/publication/a-brief-discussion-on-selecting-new-elliptic-curves/. Position paper presented at the NIST Workshop on Elliptic Curve Cryptography Standards (http://www.nist.gov/itl/csd/ct/ecc-workshop.cfm). 2015. (Visited on 08/03/2026).

63. V. Cristiani, M. Lecomte, and T. Hiscock. "A Bit-Level Approach to Side Channel Based Disassembling". In: *CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*. Vol. 11833. LNCS. Springer, 2019, pp. 143–158. DOI: 10.1007/978-3-030-42068-0_9.

64. J. Danger, S. Guilley, P. Hoogvorst, C. Murdica, and D. Naccache. "A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards". *J. Cryptographic Engineering* 3:4, 2013, pp. 241–265. DOI: 10.1007/s13389-013-0062-6.

65. R. Daudigny, H. Ledig, F. Muller, and F. Valette. "SCARE of the DES". In: *ACNS 05*. Vol. 3531. LNCS. Springer, Heidelberg, 2005, pp. 393–406. DOI: 10.1007/11496137_27.

66. H. Davenport. *Multiplicative number theory*. Vol. 74. Springer Science & Business Media, 2013.

67. P. De Rooij. "Efficient exponentiation using precomputation and vector addition chains". In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1994, pp. 389–399.

68. L. Di and L. Yan. *Introduction to the Commercial Cryptography Scheme in China*. https://icmconference.org/wp-content/uploads/C23Introduction-on-the-Commercial-Cryptography-Scheme-in-China-20151105.pdf. 2015. (Visited on 08/03/2026).

69. L. Eagen, Y. E. Housni, S. Masson, and T. Piellard. *Fast elliptic curve scalar multiplications in SN(T)ARK circuits*. Cryptology ePrint Archive, Paper 2025/933. 2025. URL: https://eprint.iacr.org/2025/933.

70. *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*. Technical report. IETF RFC 5639, 2010.

71. T. Eisenbarth, C. Paar, and B. Weghenkel. "Building a Side Channel Based Disassembler". *Trans. Comput. Sci.* 10, 2010, pp. 78–99. DOI: 10.1007/978-3-642-17499-5_4.

72. J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. 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*. IEEE, 2010, pp. 76–87. DOI: 10.1109/HST.2010.5513110.

73. J. Fan and I. 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*. Vol. 6805. LNCS. Springer, 2012, pp. 265–282. DOI: 10.1007/978-3-642-28368-0_18.

74. A. Faz-Hernández, P. Longa, and A. H. Sánchez. "Efficient and Secure Algorithms for GLV-Based Scalar Multiplication and Their Implementation on GLV-GLS Curves". In: *Topics in Cryptology – CT-RSA 2014*. Ed. by J. Benaloh. Springer International Publishing, Cham, 2014, pp. 1–27. ISBN: 978-3-319-04852-9.

75. B. Feix, M. Roussellet, and A. Venelli. "Side-Channel Analysis on Blinded Regular Scalar Multiplications". In: *Progress in Cryptology – INDOCRYPT 2014 – 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*. Ed. by W. Meier and D. Mukhopadhyay. Vol. 8885. Lecture Notes

in Computer Science. Springer, 2014, pp. 3–20. DOI: 10.1007/978-3-319-13039-2_1.

76. *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION 186 Digital Signature Standard (DSS)*. Standard. National Institute for Standards and Technology, 1994.

77. T. Firmware. *Mbed TLS*. https://github.com/Mbed-TLS/mbedtls. 2024. (Visited on 08/03/2026).

78. P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. "Fault Attack on Elliptic Curve Montgomery Ladder Implementation". In: *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2008, pp. 92–98. DOI: 10.1109/FDTC.2008.15.

79. G. Frey and H.-G. Rück. "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves". *Mathematics of Computation* 62:206, 1994, pp. 865–874.

80. S. D. Galbraith, X. Lin, and M. Scott. "Endomorphisms for Faster Elliptic Curve Cryptography on a Large Class of Curves". In: *Advances in Cryptology - EUROCRYPT 2009*. Ed. by A. Joux. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 518–535. ISBN: 978-3-642-01001-9.

81. R. Gallant, R. Lambert, and S. Vanstone. "Improving the parallelized Pollard lambda search on anomalous binary curves". *Mathematics of Computation* 69:232, 2000, pp. 1699–1705.

82. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms". In: *Advances in Cryptology — CRYPTO 2001*. Ed. by J. Kilian. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 190–200. ISBN: 978-3-540-44647-7.

83. S. Gao, E. Oswald, and D. 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*. Springer-Verlag, Trondheim, Norway, 2022, pp. 284–311. ISBN: 978-3-031-07081-5. DOI: 10.1007/978-3-031-07082-2_11.

84. C. P. Garcia and S. Sovio. "Size, Speed, and Security: An Ed25519 Case Study". In: *NordSec 2021, Virtual Event, November 29-30, 2021*. Vol. 13115. LNCS. Springer, 2021, pp. 16–30. DOI: 10.1007/978-3-030-91625-1_2.

85. J. van Geest and I. Buhan. "A Side-Channel Based Disassembler for the ARM-Cortex M0". In: *ACNS 2022 Satellite Workshops, Rome, Italy, June 20-23, 2022*. Vol. 13285. LNCS. Springer, 2022, pp. 183–199. DOI: 10.1007/978-3-031-16815-4_11.

86. Google. *Paranoid Crypto*. 2016. URL: https://github.com/google/paranoid_crypto (visited on 08/03/2026).

87. L. Goubin. "A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems". In: *Public Key Cryptography — PKC 2003*. Ed. by Y. G. Desmedt. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 199–211. ISBN: 978-3-540-36288-3.

88. J. Groth. "On the size of pairing-based non-interactive arguments". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2016, pp. 305–326.

89. A. Guillevic. *Pairing-friendly curves*. URL: https://people.irisa.fr/Aurore.Guillevic/Pairing_friendly_curves.html (visited on 08/03/2026).

90. S. Guilley, L. Sauvage, J. Micolod, D. Réal, and F. Valette. "Defeating Any Secret Cryptography with SCARE Attacks". In: *LATINCRYPT 2010*. Vol. 6212. LNCS. Springer, Heidelberg, 2010, pp. 273–293. DOI: 10.1007/978-3-642-14712-8_17.

91. T. C. Hales. "The NSA back door to NIST". *Notices of the AMS* 61:2, 2013, pp. 190–192.

92. M. Hamburg. *Fast and compact elliptic-curve cryptography*. Cryptology ePrint Archive, Paper 2012/309. https://eprint.iacr.org/2012/309. 2012. URL: https://eprint.iacr.org/2012/309.

93. M. Hamburg. *A note on high-security general-purpose elliptic curves*. Cryptology ePrint Archive, Report 2015/625. https://eprint.iacr.org/2015/625.pdf. 2015.

94. M. Hamburg. *Ed448-Goldilocks, a new elliptic curve*. Cryptology ePrint Archive, Paper 2015/625. 2015. URL: https://eprint.iacr.org/2015/625.

95. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Professional Computing. Springer, 2004. ISBN: 0-387-95273-X. DOI: 10.1016/s0012-365x(04)00102-5.

96. D. Harkins. *RFC 6932: Brainpool Elliptic Curves for the Internet Key Exchange (IKE) Group Description Registry*. 2013.

97. D. Hopwood. *The pasta curves*. https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/. 2020. (Visited on 08/03/2026).

98. M.-D. Huang and W. Raskind. "Signature Calculus and Discrete Logarithm Problems". In: *Algorithmic Number Theory*. Ed. by F. Hess, S. Pauli, and M. Pohst. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 558–572. ISBN: 978-3-540-36076-6.

99. IEEE. *IEEE Standard Specifications for Public-Key Cryptography*. Standard. IEEE Std 1363-2000 Working Group, 2000.

100. ISO/IEC 15946. *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 5: Elliptic curve generation*. 2017.

101. V. V. Iyer, A. Thimmaiah, M. Orshansky, A. Gerstlauer, and A. E. Yilmaz. "A Hierarchical Classification Method for High-accuracy Instruction Disassembly with Near-field EM Measurements". *ACM Trans. Embed. Comput. Syst.* 23:1, 2024, 10:1–10:21. DOI: 10.1145/3629167.

102. T. Izu and T. Takagi. "Exceptional Procedure Attack on Elliptic Curve Cryptosystems". In: *Public Key Cryptography – PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. Ed. by Y. Desmedt. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 224–239. DOI: 10.1007/3-540-36288-6_17.

103. T. Jager, J. Schwenk, and J. Somorovsky. "Practical Invalid Curve Attacks on TLS-ECDH". In: *Computer Security – ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*. Ed. by G. Pernul, P. Y. A. Ryan, and E. R. Weippl. Vol. 9326. Lecture Notes in Computer Science. Springer, 2015, pp. 407–425. DOI: 10.1007/978-3-319-24174-6_21.

104. J. 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/ (visited on 08/03/2026).

105. J. Jancar, V. Sedlacek, P. Svenda, and M. Sýs. "Minerva: The curse of ECDSA nonces; Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces". *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020:4, 2020, pp. 281–308. DOI: 10.13154/TCHES.V2020.I4.281-308.

106. A. Janovsky, J. Jancar, P. Svenda, Ł. Chmielewski, J. Michalik, and V. Matyas. "sec-certs: Examining the security certification practice for better vulnerability mitigation". *Comput. Secur.* 143, 2024, p. 103895. DOI: 10.1016/J.COSE.2024.103895.

107. D. Jao, S. D. Miller, and R. Venkatesan. "Ramanujan graphs and the random reducibility of discrete log on isogenous elliptic curves". *Cryptology ePrint Archive*, 2004.

108. Journal officiel de la republique francaise. *Avis relatif aux parametres de courbes elliptiques definis par l'Etat francais*. 2011. URL: https://www.legifrance.gouv.fr/download/pdf?id=QfYWtPSAJVtAB_c6Je5tAv0OOY2r1ad3LaVVmnStGvQ= (visited on 08/03/2026).

109. M. Joye and M. Tunstall. "Exponent Recoding and Regular Exponentiation Algorithms". In: *Progress in Cryptology – AFRICACRYPT 2009*. Ed. by B. Preneel. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 334–349. ISBN: 978-3-642-02384-2.

110. S. S. Keller. "The elliptic curve cryptography cofactor Diffie-Hellman (ECC CDH) primitive validation system (ECC_CDHVS)". *NIST Information Technology Laboratory*, 2011. URL: https://csrc.nist.gov/csrc/media/projects/cryptographic-algorithm-validation-program/documents/components/ecccdhvs.pdf (visited on 08/03/2026).

111. A. Kerckhoffs. "La cryptographie militaire". *Journal des sciences militaires* IX, 1883, pp. 161–191.

112. T. Kim and R. Barbulescu. "Extended tower number field sieve: A new complexity for the medium prime case". In: *Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptol-*

ogy Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Springer. 2016, pp. 543–571.

113. A. H. Koblitz, N. Koblitz, and A. Menezes. "Elliptic curve cryptography: The serpentine course of a paradigm shift". *Journal of Number theory* 131:5, 2011, pp. 781–814.
114. N. Koblitz. "Elliptic curve cryptosystems". *Math. Comp.* 48:177, 1987, pp. 203–209. ISSN: 0025-5718. DOI: 10.2307/2007884.
115. N. Koblitz and A. Menezes. *Another Look at Generic Groups*. Cryptology ePrint Archive, Paper 2006/230. 2006. URL: https://eprint.iacr.org/2006/230.
116. N. Koblitz and A. Menezes. "A riddle wrapped in an enigma". *IEEE Security & Privacy* 14:6, 2016, pp. 34–42.
117. P. Kocher, J. Jaffe, and B. Jun. "Differential power analysis". In: *Advances in Cryptology – CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer. 1999, pp. 388–397.
118. M. Kraitchik. *Théorie des nombres*. Vol. 1. Gauthier-Villars, 1922.
119. A. K. Lenstra, H. W. Lenstra, and L. Lovász. "Factoring polynomials with rational coefficients". *Mathematische annalen* 261:ARTICLE, 1982, pp. 515–534.
120. C. H. Lim and P. J. Lee. "A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup". In: *Advances in Cryptology – CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. Ed. by B. S. K. Jr. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 249–263. DOI: 10.1007/BFB0052240.
121. M. Lochter, J. Merkle, J.-M. Schmidt, and T. Schütze. *Requirements for Standard Elliptic Curves*. Cryptology ePrint Archive, Paper 2014/832. 2014. URL: https://eprint.iacr.org/2014/832.
122. P. Longa and F. Sica. "Four-dimensional Gallant-Lambert-Vanstone scalar multiplication". In: *Advances in Cryptology – ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*. Springer. 2012, pp. 718–739.

123. O. L. Macsorley. "High-Speed Arithmetic in Binary Computers". *Proceedings of the IRE* 49:1, 1961, pp. 67–91. DOI: 10.1109/JRPROC.1961.287779.

124. N. Madden. *Psychic signatures in Java*. 2022. URL: https://neilmadden.blog/2022/04/19/psychic-signatures-in-java/ (visited on 08/03/2026).

125. J. Maillard, T. Hiscock, M. Lecomte, and C. 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*. IEEE, 2022, pp. 472–479. DOI: 10.1109/DSD57027.2022.00069.

126. S. Masson, A. Sanso, and Z. Zhang. "Bandersnatch: a fast elliptic curve built over the BLS12-381 scalar field". *Des. Codes Cryptography* 92:12, 2024, pp. 4131–4143. ISSN: 0925-1022. DOI: 10.1007/s10623-024-01472-0.

127. G. Maxwell. *The most repeated R value on the blockchain*. https://bitcointalk.org/index.php?topic=1118704.0. 2015. (Visited on 08/03/2026).

128. A. Menezes, S. Vanstone, and T. Okamoto. "Reducing elliptic curve logarithms to logarithms in a finite field". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*. STOC '91. Association for Computing Machinery, New Orleans, Louisiana, USA, 1991, pp. 80–89. ISBN: 0897913973. DOI: 10.1145/103418.103434.

129. A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 2018. ISBN: 9780429466335.

130. J. Merkle and M. Lochter. *RFC 7027: Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS)*. 2013.

131. G. D. Micheli and N. Heninger. *Recovering cryptographic keys from partial information, by example*. Cryptology ePrint Archive, Paper 2020/1506. 2020. URL: https://eprint.iacr.org/2020/1506.

132. A. Miele and A. K. Lenstra. "Efficient ephemeral elliptic curve cryptographic keys". In: *International Conference on Information Security*. Springer. 2015, pp. 524–547.

133. V. S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology - CRYPTO'85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*. Ed. by H. C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426. DOI: 10.1007/3-540-39799-X_31.

134. MIRACL UK Ltd. *Multiprecision Integer and Rational Arithmetic Cryptographic Library*. https://github.com/miracl/MIRACL. 2018. (Visited on 08/03/2026).

135. A. Miyaji, M. Nakabayashi, and S. Takano. "New Explicit Conditions of Elliptic Curve Traces for FR-Reduction". *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 84, 2001, pp. 1234–1243.

136. P. L. Montgomery. "Modular Multiplication Without Trial Division". *Mathematics of Computation* 44:170, 1985, pp. 519–521. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2007970 (visited on 08/03/2026).

137. P. L. Montgomery. "Speeding the Pollard and Elliptic Curve Methods of Factorization". *Mathematics of Computation* 48:177, 1987, pp. 243–264. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2007888 (visited on 08/03/2026).

138. Mozilla Security. *Cryptofuzz*. github.com/MozillaSecurity/cryptofuzz. 2024. (Visited on 08/03/2026).

139. M. Msgna, K. Markantonakis, and K. Mayes. "Precise Instruction-Level Side Channel Profiling of Embedded Processors". In: *ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings*. Vol. 8434. LNCS. Springer, 2014, pp. 129–143. DOI: 10.1007/978-3-319-06320-1_11.

140. C. Murdica, S. Guilley, J.-L. Danger, P. Hoogvorst, and D. Naccache. "Same Values Power Analysis Using Special Points on Elliptic Curves". In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by W. Schindler and S. A. Huss. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 183–198. ISBN: 978-3-642-29912-4.

141. D. Naccache, D. M'RaÏhi, S. Vaudenay, and D. Raphaeli. "Can DSA be improved? – Complexity trade-offs with the digital signature standard". In: *Advances in Cryptology – EUROCRYPT'94:*

Workshop on the Theory and Application of Cryptographic Techniques Perugia, Italy, May 9–12, 1994 Proceedings 13. Springer. 1995, pp. 77–85.

142. S. Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". *Decentralized Business Review*, 2008, p. 21260.

143. P. Narimani, M. A. Akhaee, and S. Habibi. "Side-Channel based Disassembler for AVR Micro-Controllers using Convolutional Neural Networks". In: *ISCISC 2021, Isfahan, Iran, September 1-2, 2021*. IEEE, 2021, pp. 75–80. DOI: 10.1109/ISCISC53448.2021.9720466.

144. National Institute of Standards and Technology. *FIPS Publication 186-2*. 2000. URL: https://csrc.nist.gov/publications/detail/fips/186/2/archive/2000-01-27 (visited on 08/03/2026).

145. NCC Group. *Arbitrary Signature Forgery in Stark Bank ECDSA Libraries*. https://www.nccgroup.com/us/research-blog/technical-advisory-arbitrary-signature-forgery-in-stark-bank-ecdsa-libraries-cve-2021-43572-cve-2021-43570-cve-2021-43569-cve-2021-43568-cve-2021-43571/. 2021. (Visited on 08/03/2026).

146. M. Nemec, M. Sys, P. Svenda, D. Klinec, and V. Matyas. "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli". In: *ACM CCS 2017*. ACM Press, 2017, pp. 1631–1648. DOI: 10.1145/3133956.3133969.

147. S. Neves and M. Tibouchi. "Degenerate Curve Attacks – Extending Invalid Curve Attacks to Edwards Curves and Other Models". In: *Public-Key Cryptography – PKC 2016 – 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*. Ed. by C. Cheng, K. Chung, G. Persiano, and B. Yang. Vol. 9615. Lecture Notes in Computer Science. Springer, 2016, pp. 19–35. DOI: 10.1007/978-3-662-49387-8_2.

148. P. Q. Nguyen and D. Stehlé. "Low-dimensional lattice basis reduction revisited". *ACM Transactions on algorithms (TALG)* 5:4, 2009, pp. 1–48.

149. Q. Nguyen. *Practical Cryptanalysis of JSON Web Token and Galois Counter Mode's Implementations*. Presented on the Real World Cryptography 2017 conference. 2017. URL: https://research.google/pubs/practical-cryptanalysis-of-json-web-token-and-galois-counter-modes-implementations/ (visited on 08/03/2026).

150. Q. T. M. Nguyen. "0". *IACR Cryptol. ePrint Arch.*, 2021, p. 323. URL: https://eprint.iacr.org/2021/323.

151. NIST. *SP 800-56A Rev. 3: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*. Standard. National Institute for Standards and Technology, 2018. URL: https://doi.org/10.6028/NIST.SP.800-56Ar3.

152. NIST. *SP 800-186: Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters*. Standard. National Institute for Standards and Technology, 2023. URL: https://doi.org/10.6028/NIST.SP.800-186.

153. L. Obrátil, D. Klinec, P. Švenda, and M. Ukrop. *Randomness Testing Toolkit*. https://github.com/crocs-muni/randomness-testing-toolkit. 2015. (Visited on 08/03/2026).

154. J. Oesterle. "Nombres de classes des corps quadratiques imaginaires". *Asterisque* 121:122, 1985, pp. 309–323.

155. OpenCash. *Cashu*. https://cashu.space/. (Visited on 08/03/2026).

156. OpenTitan. *Open source silicon root of trust (RoT)*. URL: https://opentitan.org/ (visited on 08/03/2026).

157. Oracle. *Oracle Java Card technology*. 2023. URL: https://www.oracle.com/java/java-card/ (visited on 08/03/2026).

158. H. Orman. *The OAKLEY Key Determination Protocol*. RFC 2412. 1998. URL: https://rfc-editor.org/rfc/rfc2412.txt (visited on 08/03/2026).

159. J. Park, X. Xu, Y. Jin, D. Forte, and M. 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*. ACM, 2018, 119:1–119:6. DOI: 10.1145/3195970.3196094.

160. M. S. Pedro, M. Soos, and S. Guilley. "FIRE: Fault Injection for Reverse Engineering". In: *WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011*. Vol. 6633. LNCS. Springer, 2011, pp. 280–293. DOI: 10.1007/978-3-642-21040-2_20.

161. G. C. Pereira, M. A. Simplício Jr, M. Naehrig, and P. S. Barreto. "A family of implementation-friendly BN elliptic curves". *Journal of Systems and Software* 84:8, 2011, pp. 1319–1326.

162. N. Pippenger. "On the evaluation of powers and related problems". In: *17th Annual Symposium on Foundations of Computer Science (SFCS 1976)*. IEEE. 1976, pp. 258–263. DOI: 10.1109/SFCS.1976.21.

163. S. Pohlig and M. Hellman. "An Improved Algorithm for Computing Logarithms over $GF(p)$ and Its Cryptographic Significance". *IEEE Transactions on Information Theory* 24:1, 1978, pp. 106–110. DOI: 10.1109/TIT.1978.1055817.

164. V. Popov, S. Leontiev, and I. Kurepkin. *Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms*. 2006. URL: https://rfc-editor.org/rfc/rfc4357.txt (visited on 08/03/2026).

165. T. Pornin. https://crypto.stackexchange.com/questions/60420/what-does-the-special-form-of-the-base-point-of-secp256k1-allow. 2019. (Visited on 08/03/2026).

166. T. Pornin. *Optimized Lattice Basis Reduction In Dimension 2, and Fast Schnorr and EdDSA Signature Verification*. Cryptology ePrint Archive, Paper 2020/454. 2020. URL: https://eprint.iacr.org/2020/454.

167. J. Proos. *Joint sparse forms and generating zero columns when combing*. Technical report. Technical Report CORR 2003-23, Department of Combinatorics and Optimization, 2003.

168. J.-J. Quisquater and D. 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. USENIX Association, San Jose, CA, 2002, p. 6. DOI: 10.5555/1250988.1250994.

169. D. Réal, V. Dubois, A. Guilloux, F. Valette, and M. Drissi. "SCARE of an Unknown Hardware Feistel Implementation". In: *CARDIS 2008, London, UK, September 8-11, 2008*. Vol. 5189. LNCS. Springer, 2008, pp. 218–227. DOI: 10.1007/978-3-540-85893-5_16.

170. J. Renes, C. Costello, and L. Batina. "Complete Addition Formulas for Prime Order Elliptic Curves". In: *Proceedings, Part I, of the 35th Annual International Conference on Advances in Cryptology – EUROCRYPT 2016 – Volume 9665*. Springer-Verlag, Berlin, Heidelberg, 2016, pp. 403–428. ISBN: 9783662498897.

171. M. Rivain and T. Roche. "SCARE of Secret Ciphers with SPN Structures". In: *ASIACRYPT 2013, Part I*. Vol. 8269. LNCS. Springer, Heidelberg, 2013, pp. 526–544. DOI: 10.1007/978-3-642-42033-7_27.

172. T. Roche. "EUCLEAK". *IACR Cryptol. ePrint Arch.*, 2024, p. 1380. URL: https://eprint.iacr.org/2024/1380.

173. T. Roche, V. Lomné, C. Mutschler, and L. Imbert. "A Side Journey To Titan". In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by M. D. Bailey and R. Greenstadt. USENIX Association, 2021, pp. 231–248. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/roche.

174. A. Sanso. *Critical vulnerability in JSON Web Encryption (JWE)*. https://blog.intothesymmetry.com/2017/03/critical-vulnerability-in-json-web.html. 2017. (Visited on 08/03/2026).

175. A. Sanso. *CVE-2017-7781/CVE-2017-10176: Issue with elliptic curve addition in mixed Jacobian-affine coordinates in Firefox/Java*. 2017. URL: https://blog.intothesymmetry.com/2017/08/cve-2017-7781cve-2017-10176-issue-with.html (visited on 08/03/2026).

176. T. Satoh, K. Araki, et al. "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves". *Rikkyo Daigaku sugaku zasshi* 47:1, 1998, pp. 81–92.

177. B. Schneier. *The NSA Is Breaking Most Encryption on the Internet*. https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html. 2013. (Visited on 08/03/2026).

178. R. Schoof. "Counting points on elliptic curves over finite fields". *Journal de Théorie des Nombres de Bordeaux*, 1995, pp. 219–254.

179. M. Scott. *Re: NIST annouces set of Elliptic Curves*. web.archive.org/web/20160313065951/https://groups.google.com/forum/message/raw?msg=sci.crypt/mFMukSsORmI/FpbHDQ6hM_MJ. 1999. (Visited on 08/03/2026).

180. M. Scott. *A new curve*. https://moderncrypto.org/mail-archive/curves/2015/000449.html. 2015. (Visited on 08/03/2026).

181. SECG. *SEC 1: Elliptic Curve Cryptography*. Standard. Standards for Efficient Cryptography Group, 2009. URL: http://www.secg.org/sec1-v2.pdf (visited on 08/03/2026).

182. SECG. *SEC 2: Recommended Elliptic Curve Domain Parameters*. Standard. Standards for Efficient Cryptography Group, 2010. URL: http://www.secg.org/sec2-v2.pdf (visited on 08/03/2026).

183. S. O. G. I. S. Security. *Application of Attack Potential to Smartcards and Similar Devices*. Joint Interpretation Library. 2022. URL: https://www.sogis.eu/documents/cc/domains/sc/JIL-Application-of-Attack-Potential-to-Smartcards-v3.2.pdf.

184. V. Sedlacek, J.-J. Chi-Dominguez, J. Jancar, and B. B. Brumley. "A Formula for Disaster: A Unified Approach to Elliptic Curve Special-Point-Based Attacks". In: *Advances in Cryptology – ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part I*. Springer-Verlag, Singapore, Singapore, 2021, pp. 130–159. ISBN: 978-3-030-92061-6. DOI: 10.1007/978-3-030-92062-3_5.

185. V. Sedlacek, J. Jancar, and P. Svenda. "Fooling Primality Tests on Smartcards". In: *Computer Security – ESORICS 2020 – 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II*. Ed. by L. Chen, N. Li, K. Liang, and S. A. Schneider. Vol. 12309. Lecture Notes in Computer Science. Springer, 2020, pp. 209–229. DOI: 10.1007/978-3-030-59013-0_11.

186. V. Sedlacek, V. Suchanek, and A. Dufka. *DiSSECT:Distinguisher of Standard and Simulated Elliptic Curves via Traits*. https://dissect.crocs.fi.muni.cz/. (Visited on 08/03/2026).

187. I. Semaev. "On computing logarithms on elliptic curves". *Discrete Mathematics and Applications* 6:1, 1996, pp. 69–76.

188. I. Semaev. "Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p". *Mathematics of computation* 67:221, 1998, pp. 353–356.

189. I. Semaev. "A 3-dimensional lattice reduction algorithm". In: *Cryptography and Lattices: International Conference, CaLC 2001 Providence, RI, USA, March 29–30, 2001 Revised Papers*. Springer. 2001, pp. 181–193.

190. I. Semaev. *Summation polynomials and the discrete logarithm problem on elliptic curves*. Cryptology ePrint Archive, Paper 2004/031. https://eprint.iacr.org/2004/031. 2004. URL: https://eprint.iacr.org/2004/031.

191. S. Shen, S. Shen, and X. Lee. *SM2 Digital Signature Algorithm*. Internet-Draft. Work in Progress. 2014. URL: https://datatracker.ietf.org/doc/html/draft-shen-sm2-ecdsa-02 (visited on 08/03/2026).

192. P. W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. IEEE. 1994, pp. 124–134.

193. V. Shoup. "Lower Bounds for Discrete Logarithms and Related Problems". In: *Advances in Cryptology — EUROCRYPT '97*. Ed. by W. Fumy. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 256–266. ISBN: 978-3-540-69053-5.

194. J. H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics 106. Springer-Verlag, 1986. ISBN: 978-0-387-09493-9.

195. N. P. Smart. "The discrete logarithm problem on elliptic curves of trace one". *Journal of cryptology* 12:3, 1999, pp. 193–196.

196. E. S. Smyshlyaeva and V. Fotieva. *Information technology. Cryptographic data security. Parameters of elliptic curves for cryptographic algorithms and protocols*. Federal Agency on Technical Regulating and Metrology. 2016.

197. J. A. Solinas. "Low-weight binary representation for pairs of integers". *Centre for Applied Cryptographic Research, University of Waterloo, Combinatorics and Optimization Research Report CORR 2001-41*, 2001.

198. J. A. Solinas. “Generalized Mersenne Prime”. In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg and S. Jajodia. Springer US, 2011.

199. I. O. for Standardization. “ISO/IEC 15408 Information technology – Security techniques – Evaluation criteria for IT security”. In: *ISO/IEC 15408-1:2022*. ISO/IEC, 2022.

200. J. Stern, D. Pointcheval, J. Malone-Lee, and N. P. Smart. “Flaws in applying proof methodologies to signature schemes”. In: *Annual International Cryptology Conference*. Springer. 2002, pp. 93–110.

201. D. Strobel, F. Bache, D. F. Oswald, F. Schellenberg, and C. 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*. ACM, 2015, pp. 139–144. DOI: 10.5555/2755753.2755784.

202. P. Svenda, R. Kvasnovsky, I. Nagy, and A. Dufka. “JCAlgTest: Robust identification metadata for certified smartcards”. eng. In: *19th International Conference on Security and Cryptography*. INSTICC, Lisabon, 2022, pp. 597–604. ISBN: 978-989-758-590-6. DOI: 10.5220/0000163500003283.

203. M. Tang, Z. Qiu, H. Peng, X. Hu, Y. Mu, and H. Zhang. “Toward reverse engineering on secret S-boxes in block ciphers”. *Sci. China Inf. Sci.* 57:3, 2014, pp. 1–18. DOI: 10.1007/S11432-013-5053-9.

204. C. E. Torkelson. “The Clipper Chip: How Key Escrow Threatens to Undermine the Fourth Amendment”. *Seton Hall L. Rev.* 25, 1994, p. 1142.

205. E. Trichina and A. Bellezza. “Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks”. In: *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by B. S. K. Jr., Ç. K. Koç, and C. Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 98–113. DOI: 10.1007/3-540-36400-5_9.

206. TropicSquare. *TROPIC01 Secure Element*. https://tropicsquare.com/. (Visited on 08/03/2026).

207. L. Valenta, N. Sullivan, A. Sanso, and N. Heninger. "In Search of CurveSwap: Measuring Elliptic Curve Implementations in the Wild". In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 2018, pp. 384–398. DOI: 10.1109/EUROSP.2018.00034.

208. F. Valsorda. *Squeezing a key through a carry bit*. 2017. URL: https://fahrplan.events.ccc.de/congress/2017/Fahrplan/events/9021.html (visited on 08/03/2026).

209. D. Vermoen, M. F. Witteman, and G. Gaydadjiev. "Reverse Engineering Java Card Applets Using Power Analysis". In: *WISTP 2007, Heraklion, Crete, Greece, May 9-11, 2007*. Vol. 4462. LNCS. Springer, 2007, pp. 138–149. DOI: 10.1007/978-3-540-72354-7_12.

210. J. Von Zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

211. D. Wagner. *Chaumian ecash without RSA*. https://cypherpunks.venona.com/date/1996/03/msg01848.html. 1996. (Visited on 08/03/2026).

212. P. Wang and F. Zhang. "Computing elliptic curve discrete logarithms with the negation map". *Information Sciences* 195, 2012, pp. 277–286.

213. L. C. Washington. *Elliptic curves: number theory and cryptography*. Second. Discrete Mathematics and its Applications. Chapman and Hall/CRC, 2008. ISBN: 978-1-4200-7146-7. DOI: 10.1201/9781420071474.

214. S. Weiser, D. Schrammel, L. Bodner, and R. Spreitzer. "Big Numbers - Big Troubles: Systematically Analyzing Nonce Leakage in (EC)DSA Implementations". In: *USENIX Security Symposium*. 2020.

215. P. Whittaker and T. Randall. *Wide Boyz*. 2020. URL: https://www.youtube.com/@WideBoyz (visited on 08/03/2026).

216. Wireless Application Protocol Forum. *Wireless Application Protocol Wireless Transport Layer Security Specification*. https://web.archive.org/web/20170829023257/https://www.wapforum.org/tech/documents/WAP-199-WTLS-20000218-a.pdf. 2000. (Visited on 08/03/2026).

217. P. Wuille. *libsecp256k1*. https://github.com/bitcoin-core/secp256k1. 2013. (Visited on 08/03/2026).

218. P. Wuille, J. Nick, and T. Ruffing. *Schnorr Signatures for secp256k1*. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki. 2020. (Visited on 08/03/2026).