Masaryk University

and

Ca' Foscari University of Venice

On cryptographic weaknesses related to elliptic curves

Doctoral Thesis

Vladimír Sedláček

Advisor (Masaryk University)	**Advisor (Ca' Foscari University)**
Prof. Vashek Matyáš	Prof. Riccardo Focardi
Consultant	**Doctorate Coordinator**
Dr. Marek Sýs	Prof. Riccardo Focardi
Semester: Fall 2021	Cycle: 33
UČO: 408178	Matricola: 956406
	SSD: INF/01 Informatica

Declaration

Hereby I declare that this thesis is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Vladimír Sedláček

Advisor (Masaryk University): Prof. Vashek Matyáš
Consultant (Masaryk University): Dr. Marek Sýs
Advisor (Ca' Foscari University of Venice): Prof. Riccardo Focardi

Acknowledgements

I would like to thank Vashek Matyáš for guidance, freedom, and teaching me some realism, and Riccardo Focardi for the opportunity to learn new topics and appreciate a different culture.

I am also grateful to all my CRoCS colleagues, especially:
Marek for challenging my views on the interplay of math, formalism and intuition,
Petr for countless profound, inspiring and amusing discussions,
Jano for showing me how to push one's limits via an amazing passion and drive,
Dušan for impressive computing and meme-generating skills,
Martin for questioning and improving the environment, and exemplified teaching,
Vojta for sharing the joy and burden that comes with pure math,
Tonda for pragmatically approaching abstract ideas,
Lukáš for helping me explore the concept of practicality,
Matúš for pioneering the double PhD path,
Adam for lightening the mood and offering alternative viewpoints,
Agi for her take on empathy and listening.

My deep thanks also belongs to my family, friends, and Minh, who stood by me during the rough times and helped me preserve my sanity during the long journey. The list of all people who positively influenced my PhD is too long to fit this margin, but you will not be forgotten.

Last but not least, I would also like to thank organizations and projects who supported me in my research - namely, the Ph.D. Talent Scholarship funded by the Brno City Municipality, the Czech Science Foundation project GA20-03426S, the grant from the Cisco University Research Program Fund of Silicon Valley Community Foundation, and "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Abstract

It is possible to write endlessly on elliptic curves. (*This is not a threat.*)

Serge Lang – Elliptic curves: Diophantine analysis

This thesis[1] explores the security of cryptographic systems related to elliptic curves from various angles. We discuss a way of leveraging a special factorization method based on elliptic curves to insert a backdoor into RSA implementations. We try to fool the primality tests applied to elliptic curve parameters that are present on smartcards. We study a flaw in scalar multiplication that allowed us to recover the private key in many real-world signature scheme implementations. We unify several approaches to detecting problems in elliptic curve addition formulas in the side-channel context and come up with a new attack. We propose a new scheme for elliptic curve Schnorr multisignatures, focusing on efficiency and interoperability. Finally, we design a framework that allows for a systematic analysis of standardized curves by comparing them to simulated ones in many different aspects.

We conclude that while elliptic curve cryptography is very useful, one must be very careful to correctly tame its complexity and avoid devastating pitfalls at many levels.

1. The thesis builds upon the author's previous PhD research proposal thesis [Sed20].

Keywords

cryptography, ECC, ECDH, ECDSA, elliptic curve, factorization, formulas, implementations, primality testing, standards

Contents

Mathematical notation

$E(\mathbb{F}_p)$	the group of an elliptic curve E over $\mathbb{F}_p$
$\mathbb{F}_p$	the field with p elements
gcd	greatest common divisor
$[k]$	scalar multiplication by k
$\mathcal{O}$	the neutral element of an elliptic curve
$\mathbb{Q}(\alpha)$	the extension of $\mathbb{Q}$ generated by α
$R/(\alpha_1,\ldots,\alpha_r)$	the ring R modulo the ideal generated by $\alpha_1,\ldots,\alpha_r$
$\#S$	the cardinality of set S
$S/\sim$	the set of equivalence classes of S under $\sim$
$(x:y:z)$	a projective space element representation
$\mathbb{Z}_m, \mathbb{Z}_m^*$	the ring of (invertible) integers modulo m
$:=$	defining equality
$\cong$	is isomorphic to
$\oplus, \times$	the direct sum, product

Used abbreviations

ANSI	American National Standards Institute
API	application programming interface
BIP	Bitcoin improvement proposal
BKZ	block Korkine-Zolotarev
CM	complex multiplication
CRT	Chinese remainder theorem
CVE	common vulnerabilities and exposures
CVP	closest vector problem
DCP	dependent coordinates problem
ECDAA	elliptic curve direct anonymous atestation
(EC)DH	(elliptic curve) Diffie-Hellman
(EC)DLP	(elliptic curve) discrete logarithm problem
(EC)DSA	(elliptic curve) digital signature algorithm
ECC	elliptic curve cryptography
EFD	Efficient Formula Database
eID	electronic identification
EPA	exceptional procedure attack
EPID	enhanced privacy identity
FIPS	Federal Information Processing Standards
FROST	Flexible Round-Optimized Schnorr Threshold Signatures
GOST	Gosudarstvenny Standart
HNP	hidden number problem
HP	the Hilbert polynomial
HSM	hardware secure module
HTTPS	Hypertext Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers
IPSec	Internet Protocol Security
KDF	key derivation function

LLL	the Lenstra-Lenstra-Lovász algorithm
MIRACL	Multiprecision Integer and Rational Arithmetic C/C++ Library
MOV	Menezes-Okamato-Vanstone
MR	Miller-Rabin
MSMB	most significant modular bits
NAF	non-adjacent form
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NUMS	nothing up my sleeve
OSCCA	Office of State Commercial Cryptography Administration
PC/SC	Personal computer / smartcard
PRF	pseudorandom function
RFC	Request for comments
RNG	random number generator
RPA	refined power analysis
RSA	Rivest-Shamir-Adleman
SEA	Schoof-Elkies-Atkin
SECG	Standards for Efficient Cryptography Group
SHINE	smartcard highly-interoperable nonce encryption scheme
SSH	Secure Shell
SSL	Secure Sockets Layer
SVP	shortest vector problem
TLS	Transport Layer Security
TPM	trusted platform module
ZVP	zero-value point

1 Introduction

1.1 Problem statement

The central question of this thesis is: *What are the undiscovered weaknesses of cryptographic systems related to elliptic curves, and how can we defend against them?* We tackle this question from various viewpoints, ranging from implementation choices through protocols to parameters of standardized curves. Another underlying leitmotiv is: *When is it possible to hide a weakness or even a backdoor in asymmetric cryptography primitives?*

1.2 Thesis structure

The rest of this chapter summarizes the contributions, while Chapter 2 provides some basic background on elliptic curves. The following six chapters each correspond to different projects:

- Chapter 3 describes the author's initial work on a very efficient factorization algorithm (using elliptic curves) for integers of a special form, possibly leading to RSA backdoors;
- Chapter 4 shows how the known methods for fooling primality tests can be adapted and used in the context of ECC and smartcards;
- Chapter 5 presents a key-recovery attack on real-world ECDSA implementations, using side channels and lattice reduction;
- Chapter 6 systematically studies many ECC formulas with respect to side channel attacks and introduces a unified attack framework;
- Chapter 7 proposes a new efficient and interoperable scheme for ECC Schnorr multisignatures;
- Chapter 8 aspires to provide an analytical framework to survey all standard curves and the standards according to which they were generated, in order to find potential hidden weaknesses.

1.3 Contributions

The most important contributions of this thesis follow, in order corresponding to the thesis structure.

On $4p-1$ factorization. We address Cheng's $4p-1$ factorization method [Che02] based on complex multiplication on elliptic curves and propose an improved, simplified and asymptotically deterministic version. We further study the viability of the $4p-1$ factorization method as the means of a potential backdoor for the RSA primes generated on black-box devices like cryptographic smartcards. We devise three detection methods for such a backdoor and audit 44 millions of RSA keypairs generated by 18 different types of cryptographic devices.

The results were published at Secrypt 2019 [Sed+19].

Fooling primality tests of smartcards. We analyze whether the smartcards of the JavaCard platform correctly validate primality of domain parameters. The work is inspired by Albrecht et al. [Alb+18], where the authors studied many open-source libraries and fooled them with specially crafted pseudoprimes. However, in the case of smartcards, often there is no way to invoke the primality test directly, so we trigger it by replacing (EC)DSA and (EC)DH prime domain parameters by adversarial composites. Such a replacement results in vulnerability to Pohlig-Hellman [PH78] style attacks, leading to private key recovery.

Out of nine smartcards (produced by five major manufacturers) we tested, all but one have no primality test in parameter validation. As the JavaCard platform provides no public primality testing API, the problem cannot be fixed by an extra parameter check, making it difficult to mitigate in already deployed smartcards.

The results were published at European Symposium on Research in Computer Security (ESORICS) 2020 [SJS20].

Minerva: The curse of ECDSA nonces. We present our discovery of a group of side-channel vulnerabilities in implementations of the ECDSA signature algorithm in a widely used Atmel AT90SC FIPS 140-2 certified smartcard chip and five cryptographic libraries (libgcrypt, wolfSSL, MatrixSSL, Crypto++, SunEC/OpenJDK/Oracle JDK).

Vulnerable implementations leak the bit-length of the scalar used in scalar multiplication via timing. Using leaked bit-length, we mount a lattice attack on a 256-bit curve, after observing enough signing operations. We propose two new methods to recover the full private key requiring just 500 signatures for simulated leakage data, 1200 for real cryptographic library data, and 2100 for smartcard data.

The number of signatures needed for a successful attack depends on the chosen method and its parameters as well as on the noise profile, influenced by the type of leakage and used computation platform. We use the set of vulnerabilities reported in this work, together with the recently published TPM-FAIL vulnerability [Mog+20], as a basis for real-world benchmark datasets to systematically compare our newly proposed methods and all previously published applicable lattice-based key recovery methods. The resulting exhaustive comparison highlights the methods' sensitivity to its proper parametrization and demonstrates that our methods are more efficient in most cases. For the TPM-FAIL dataset, we decreased the number of required signatures from approximately 40 000 to a mere 900.

Because of the impact, the work received a lot of media coverage [ZDN19; Ber19; Fei19; Tra19]. The results were published in [Jan+20], winning the Best Paper Award at Cryptographic Hardware and Embedded Systems (CHES) 2020.

A unified approach to special-point-based curve attacks. The Refined Power Analysis, Zero-Value Point, and Exceptional Procedure attacks [Gou03; AT03; IT03] introduced side-channel attack techniques against specific cases of elliptic curve cryptography. The three attacks recover bits of a static ECDH key adaptively, collecting information if a certain multiple of the input point was computed. We unify and generalize these attacks in a common framework, and solve the corresponding problem for a new class of inputs. We also introduce a version of the attack against windowed scalar multiplication methods, recovering the full scalar instead of just a part of it. Finally, we systematically analyze elliptic curve point addition formulas from the Explicit-Formulas Database [BL07a], classify all non-trivial exceptional points, and find them in new formulas. These results indicate the usefulness of our tooling for unrolling formulas and finding special points, potentially of independent research interest.

The results were published at Asiacrypt 2021 as "A formula for disaster: a unified approach to elliptic curve special-point-based attacks" with coauthors J.-J. Chi-Dominguez, J. Jancar and B. B. Brumley.

Interoperable Schnorr multisignatures.

We study the possibility of interoperability of different multiparty Schnorr signature schemes and classify them based on their approach to the nonce agreement. We identify issues that could hinder in-class interoperability and propose a construction of a trustless mediator that can facilitate interoperability among different classes in certain cases. Besides the risk mitigation, interoperability provides usability and performance benefits, as protocols better suited for special devices can be used in conjunction with more general protocols. Finally, we propose a new multiparty signature scheme SHINE, which is efficiently computable on resource-limited devices like cryptographic smartcards. SHINE is compatible with most existing unmodified Schnorr protocols like MSDL, MuSig2, or FROST thanks to the proposed interoperability construction.

The results were submitted to Applied Cryptography and Network Security 2022 as "Resilience via Practical Interoperability of Multiparty Schnorr Signature Schemes"with coauthors A. Dufka and P. Svenda.

DiSSECT: Distinguisher of Standard & Simulated Elliptic Curves via Traits. It is hard to trust elliptic curves standardized in a non-transparent way. To rectify this, we present a systematic methodology for analyzing properties of standard curves and statistically comparing them to the expected results with respect to their generation process. Based on this idea, we develop a large-scale computational framework, which thoroughly analyzes the curves and visualizes the results.

We put together the largest publicly available database of standard curves. Using our framework, we simulate over 200 000 curves to mimic the generation process of three major standards. Using three distinct distinguishing strategies for our 22 custom functions, we try to find any deviations pointing to possible weaknesses of standard curves. In this way, we discover properties of GOST curves inconsistent with claims about their origin, as well as an undocumented behavior of the BLS12-381 curve.

The results were submitted to Public Key Cryptography 2022 as "DiSSECT: Distinguisher of Standard & Simulated Elliptic Curves via Traits" with coauthors V. Suchanek and A. Dufka.

2 Elliptic curve cryptography in practice

Elliptic curves have been studied by mathematicians for a very long time and provide a very rich interplay between many areas, such as algebra, geometry, number theory and analysis. With the advent of computers, they became an important tool feasible for tasks such as integer factorization and primality proving. Crucially, they also play a fundamental role in modern cryptography because of the presumed hardness of the elliptic curve discrete logarithm problem (ECDLP).

First proposed by Koblitz [Kob87] and Miller [Mil85], widespread usage of elliptic curve cryptography (ECC) started in 2006 when the NIST standardized EC Digital Signature Algorithm (ECDSA). Besides ECDSA and its mature version EdDSA, elliptic curve Diffie-Hellman (ECDH) is also widely used – nowadays, all of these are deployed in protocols like TLS [Bla+06b] or SSH [SG09] that form the backbone of a secure Internet. Practical ECC benefits include smaller key sizes and more efficient implementations at the same security level compared to other public-key schemes like RSA or ElGamal [KMV00]. Hence it is no wonder that ECC's popularity grows, especially for security applications where computational power and integrated circuit space are limited, such as Internet of Things devices or smartcards.

ECC is also deployed in many modern systems [IANa; IANb] and even government-issued documents (such as eIDs) of many countries like Austria [Hol+08], Germany [Loc+15], Estonia, or United Arab Emirates [Qui+20], where ECDSA is used as an equivalent to handwritten signatures. Elliptic curves are also at the core of cryptocurrencies like Bitcoin [Nak08], Ethereum [Woo+14], Monero [Noe] or even Zcash [Sas+14]; the last one utilizes pairing-based cryptography.

Even though ECDLP falls prey to Shor's quantum factorization algorithm [Roe+17], the proposed post-quantum schemes include ones based on isogenies of elliptic curves [Aza+17; Cas+18], so they will probably still play an important role in the future.

2.1 Theoretical background

We define an elliptic curve E in the short Weierstrass model[1] over a prime[2] field $\mathbb{F}_p$, $p \geq 3$ by the following equation:

$$E/\mathbb{F}_p\colon y^2 = x^3 + ax + b,\ \ a,b \in \mathbb{F}_p,\ \ 4a^3 + 27b^2 \neq 0. \tag{2.1}$$

The group $E(\mathbb{F}_p)$ consists of affine points $(x,y) \in \mathbb{F}_p \times \mathbb{F}_p$ satisfying (2.1), together with the neutral element $\mathcal{O}$ – the point at infinity. For any positive integer k, we define the scalar multiplication $[k]P$ as the sum of k copies of P and also define $[-k]P := -[k]P$. The ECDLP is the task of finding k when given G and $[k]G$. The *order of a point* $P \in E(\mathbb{F}_p)$ is defined as the smallest positive integer k such that $[k]P = \mathcal{O}$. By k-torsion points, we mean points of order that divides k. We denote the cardinality of $E(\mathbb{F}_p)$ by $n := \#E(\mathbb{F}_p)$ and call $t := p + 1 - n$ the *trace of Frobenius* of E. For typical cryptographic applications, $n = h \cdot q$, where q is prime and $h \in \{1,2,4,8\}$; h is called the *cofactor*.

The scalar multiplication map $P \mapsto [k]P$ can also be computed using the *division polynomial* ψ_k [Was08]: that is,

$$[k](x,y) = \left(\frac{\phi_k(x)}{\psi_k^2(x)}, \frac{\omega_k(x,y)}{\psi_k^3(x,y)}\right),$$

where

$$\begin{aligned}
\psi_0 &= 0,\\
\psi_1 &= x,\\
\psi_2 &= 2y,\\
\psi_3 &= 3x^4 + 6ax^2 + 12bx - a^2,\\
\psi_4 &= 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3),\\
\psi_{2k+1} &= \psi_{k+2}\psi_k^3 - \psi_{k-1}\psi_{k+1}^3 && \text{for } k \geq 2,\\
\psi_{2k} &= (2y)^{-1}\psi_k(\psi_{k+2}\psi_{k-1}^2 - \psi_{k-2}\psi_{k+1}^2) && \text{for } k \geq 3,\\
\phi_k &= x\psi_k^2 - \psi_{k+1}\psi_{k-1},\\
\omega_k &= (4y)^{-1}(\psi_{k+2}\psi_{k-1}^2 - \psi_{k-2}\psi_{k+1}^2).
\end{aligned}$$

1. We will introduce other forms in Section 6.1.
2. Prime power fields are also possible, but very rare in cryptography; the only relevant case $\mathbb{F}_{2^m}$ would require special care and will not be important for us.

These polynomials are considered modulo the curve equation (2.1); in particular, ϕ_k and ψ_k^2 do not depend on y for any k. We will use these polynomials in Chapter 3 and Chapter 6.

Curves over more general rings. In Chapter 3 and Chapter 4, we will work with elliptic curves over $E(\mathbb{Z}_m)$, where m is composite and odd[3]. We will follow the exposition by Washington [Was08].

The projective plane over $\mathbb{Z}_m$ is defined as

$$\mathbb{P}_2(\mathbb{Z}_m) := \{(x,y,z) \in (\mathbb{Z}_m)^3 \mid \gcd(m,x,y,z) = 1\} / \sim,$$

where $(x,y,z) \sim (x',y',z')$ iff $x = ux', y = uy', z = uz'$ for some $u \in \mathbb{Z}_m^*$.

An elliptic curve over $\mathbb{Z}_m$ is then given by

$$E(\mathbb{Z}_m) = \{(x : y : z) \in \mathbb{P}_2(\mathbb{Z}_m) \mid y^2 z = x^3 + Axz^2 + Bz^3\},$$

where $A, B \in \mathbb{Z}_m$ and $4A^3 + 27B^2 \in \mathbb{Z}_m^*$. This generalizes the classical notion of an elliptic curve over a field, and indeed, with some caveats, the standard group addition law applies. Moreover, for odd coprime integers m_1, m_2, the natural projections $\mathbb{Z}_{m_1 m_2} \to \mathbb{Z}_{m_i}$ $(i \in \{1,2\})$ give rise to an isomorphism

$$E(\mathbb{Z}_{m_1 m_2}) \cong E(\mathbb{Z}_{m_1}) \oplus E(\mathbb{Z}_{m_2}), \tag{2.2}$$

which we will use heavily.

2.2 ECC vulnerabilities: state of the art

The complexity of ECC opens up many possible attack vectors. We will first describe generic attacks on the discrete logarithm problem (DLP), which serve as a basic guideline for selecting secure group sizes. Next, we will discuss implementation vulnerabilities, which are the most common source of problems in practice, and end with possible problems of selected standard curves.

3. More generally, we can define elliptic curves over any ring R such that $2 \in R^*$ and any projective R-module of rank 1 is free [Was08]; in particular, this includes all finite and local rings.

Generic attacks. While the DLP can be defined in any finite cyclic group, its difficulty might drastically differ for varying groups. In the multiplicative group $\mathbb{Z}_p^*$ (or its subgroup), where p is a prime, there is the subexponential index calculus algorithm [Pol93], which results in long keys for the DH and DSA schemes that operate in these groups. In contrast, despite several attempts, there does not exist an efficient analog of this algorithm for elliptic curves over prime fields, which is the reason ECC keys might be much shorter while still providing the same level of security. Thus it seems so far that besides certain weak classes of curves, the only efficient attacks against the ECDLP are those that could be applied to any finite cyclic group.

One such important attack is the one introduced by Pohlig and Hellman [PH78], which makes use of the Chinese remainder theorem (CRT) group decomposition: when $p_1, \ldots, p_r$ are distinct primes, we have $\mathbb{Z}_{p_1^{e_1} \ldots p_r^{e_r}} \cong \bigoplus_{i=1}^r \mathbb{Z}_{p_i^{e_i}}$. Since the DLP in $\mathbb{Z}_{p_i^{e_i}}$ can be solved by computing the corresponding DLP in $\mathbb{Z}_{p_i}$ and lifting it to $\mathbb{Z}_{p_i^{e_i}}$, the algorithm essentially reduces the DLP in a group to DLPs in its prime order subgroups.

Given a DLP in a prime order group, the best general attack in practice is Pollard's ρ method [Pol78], asymptotically running in time polynomial in the square root of the group order. Furthermore, the attack can be slightly optimized by using efficient arithmetic and the negation map [BL].

Implementation vulnerabilities. There are many different kinds of implementation-specific vulnerabilities; the following list is by no means exhaustive:

- Twist attacks [Fou+08] – these apply when the attacker is allowed to compute on the quadratic twist instead of the original one; can be mitigated by point validation at the implementation level or by using twist-secure curves at the curve design level.
- Invalid curve attacks [BMM00; JSS15] – these apply when an attacker is allowed to compute on another curve (usually sharing the linear coefficient in Weierstrass form) instead of the original curve. It can be mitigated by point validation at the implementation level or by specifying point compression at the protocol design level.

- Small subgroup attacks [LL97] – these apply when the attacker is allowed to make queries for multiplication by the secret scalar with points of small order, which can become powerful when combined with the above attacks. The basic form can be mitigated by a simple point check at the implementation level or by choosing curves with trivial (or small) cofactor (i.e., the ratio of the order of the full group and the subgroup generated by the public generator).

- Problems with random number generation (RNG), leading for example to trivial breaks of ECDSA if the nonce is repeated, as happened in 2010 with many Sony PlayStation 3 devices [Sch15; Ber+12].

- Side-channel attacks [Fan+10; ACL] – these consist of extracting and utilizing sensitive data from the implementation, usually from the scalar multiplication algorithms or sometimes even the field arithmetic itself; can be mitigated by using constant-time algorithms (which is not as easy to achieve as it might seem). This is a vast category of attacks [Gou03; AT03; IT03; BH09; BT11; Mur+12; Ben+14; PSY15; FWC16; Bel+16; Gen+16; GVY17; Dal+18; Ald+19; Rya19a; Rya19b; Gar+20; Ara+20; Mog+20; Wei+20]; we discuss the cases relevant to nonce leakage in Chapter 5 and curve formulas in Chapter 6 in more detail.

Even if the first three types of attacks are quite easy to defend against, Valenta, Sullivan, Sanso, and Heninger [Val+18] analyzed the security of ECDH key exchange implementations in common protocols TLS, SSH, and IPSec and estimated that a small proportion of hosts (<1% of HTTPS and SSH, and 4% of IKEv2) that support ECC do not perform required curve validity checks. Faulty RNG and side channels are generally much harder to avoid. Also, new implementation-specific attacks appear quite regularly [Age20].

Curve vulnerabilities. When using ECC in the real world, both sides of the scheme must agree on the choice of a particular curve, avoiding curves where the ECDLP is easier than it should be. The SafeCurves website [BL] presents a good overview of known curve vulnerabilities (and discusses implementation-specific vulnerabilities as well).

In short, to avoid the known mathematical attacks, the curve E defined over a prime fields $\mathbb{F}_p$ with a public generator G of order l should satisfy the following criteria:

- p should be large enough, e.g., over 256 bits if we aim for 128-bit security;
- l should have a large prime factor, as this determines the complexity of the Pohlig-Hellman attack; [PH78])
- the curve should not be anomalous (i.e., $l \neq p$), otherwise the Semaev-Satoh-Araki-Smart attack based on the additive transfer applies [Sma99; Sem98; SA+98];
- the embedding degree of E (i.e., the order of p in $\mathbb{F}_l^*$ if l is prime) should be large enough (e.g., at least 20 for the current parameter sizes), otherwise the MOV attack based on multiplicative transfer using the Weil and Tate pairings applies [Sem96; FR94; MOV93] – in particular, this rules out all supersingular curves;
- the absolute value of the CM field discriminant (i.e., the discriminant of the field $\mathbb{Q}(\sqrt{t^2 - 4p})$) should not be too low ([BL] suggests at least 2^{100}), otherwise Pollard's ρ attack can be speeded up due to the presence of efficiently computable endomorphisms of the curve [GLV01]; this does not pose a serious threat for the moment though, as the limits of the speedup are reasonably well understood.

Curve standardization. Curves suitable for wide use have been standardized by many organizations [ANSI98; Cer10; ANS14; PLK06; SSL14] and are not susceptible to any of the public mathematical attacks, but some of them fail to satisfy some SafeCurves criteria (such as twist security). Perhaps more worrisome is the fact that the choice of the constants is often not satisfiably explained, and thus we have no guarantee that these curves do not have a hidden weakness, deliberate or not. We explore this topic in much more depth in Chapter 8.

While mathematical weaknesses have a very large impact when they occur, buggy or vulnerable implementations are much more common in practice. The developers are often confronted with choices they might not be qualified to decide and the standards are usually not explicit enough to help them. There are many potential pitfalls, and history and statistics show us that problems will inevitably occur under these circumstances. Instead of blaming the developers, it seems to be more constructive to try to improve the standards, which are often at fault.

Indeed, widely used standards are often unnecessarily complicated, hard to implement correctly, do not explicitly protect against certain attacks (e.g., the invalid-curve attack on TLS-ECDH implementations [JSS15]), create tensions between simplicity and security (so that producing correct and secure implementations becomes very hard) and are suboptimal in terms of performance [BL16; Ber14; BL]. As a result, many ECC-based systems could be potentially vulnerable.

Bernstein suggested many improvements, e.g., the use of twisted Edwards curves (which also have a Montgomery form). He also designed Curve25519 [Ber06], specifically created to address many of the issues described above (for example, it has a very transparent generation, lifts the burden of responsible choices from the implementors and enables a very fast and constant-time scalar multiplication). Note that the curve is used under the name X25519 in the Montgomery form for ECDH, while the birationally equivalent curve Ed25519 in the twisted Edwards form used for EdDSA, a digital signature scheme improving on ECDSA [Ber+12; Ber+]. These curves are rising in popularity [IANa; IANb] and have also inspired the creation of several curves of the same type, notably Ed448-Goldilocks [Hamb]. The new TLS v1.3 standard from 2018 [Res18] requires support for Ed25519 and Ed448.

This does not mean, however, that these curves will automatically solve all problems. Certain libraries still only support the Weierstrass form, so when using these curves, they must perform certain transformations, thus opening themselves to potential problems again.

3 On $4p - 1$ factorization[1]

Factorization of composite integers is an old and important problem and cryptographic schemes such as RSA are based on its intractability. RSA is one of the most frequently deployed public key cryptosystems, and a possible factorization of RSA moduli could have a serious impact on the security of real-world applications. This was demonstrated in past incidents such as finding weak RSA keys used for TLS [Hen+12], LogJam [Adr+15] or factorable RSA keys from cryptographic smartcards known as the ROCA attack [Nem+17] with at least hundreds of millions affected devices. The performance of already known factorization methods, together with the required security margin, determine the necessary security parameters, such as the length of the prime factors p, q of the RSA modulus $N = pq$. Relevant standards (e.g., NIST FIPS 140-2 [NIST07], BSI TR-02102-1 [Bun18], *keylength.com* [Gir19]) then define the minimal required parameters.

We divide factorization algorithms into two categories:

1. General-purpose – applicable to all integers N (and thus influencing the minimal secure length of the RSA moduli); these include Pollard ρ [Pol75], Quadratic Sieve [Pom85] and asymptotically fastest Number Field Sieve [Pol93].

2. Special-purpose – very efficient when a factor $p|N$ or N itself is of a special form:

   (a) A number related to the prime factor p is smooth (has only small prime divisors) – Pollard's $p - 1$ [Pol74], Williams's $p + 1$ [Wil82], Bach-Shallit [BS85] and Lenstra's Elliptic Curve (ECM) [Len87] methods assume smoothness of the integers $p - 1$, $p + 1$, $\phi_k(p)$ (k-th cyclotomic polynomial) and $\#E(p)$, respectively.

   (b) Assumptions on p or N: there are fast methods for N of the form $N = p^r q$ [BDH99] or $N = p^r q^s$ [Cor+16]. Cheng's $4p - 1$ [Che] method is effective whenever the square-free part of $4p - 1$ is small.

1. The results in this chapter were published in [Sed+19]. See `https://crocs.fi.muni.cz/public/papers/Secrypt2019` for additional materials.

All of the mentioned methods look for a multiple kp of some prime divisor $p|N$. In the last step, the methods compute $gcd(N,kp) = d$. If $1 < d < N$, then a factor is found and the factorization can continue recursively. The methods are probabilistic since the factorization fails when $d = N$.

We focus on the relatively new and unexplored (there are only three related publications) $4p-1$ method in this chapter. Our contributions are the following:

- we simplify the method and analyze it in greater detail, showing that the number of expected iterations is 2-4 times lower than stated in [Che02] and the algorithm is asymptotically deterministic;
- we offer a compact public implementation of the method in Sage, together with an extensive run-time analysis;
- we discuss the viability of the method as a potential backdoor from different perspectives;
- we perform an audit of a large collection of RSA keypairs generated by 18 different types of cryptographic devices with respect to the potential backdoor;
- we discover and explain a discrepancy (that governs the possibility of modulus factorization) between random primes and primes generated by certain smartcards.

This chapter is organized as follows: In Section 3.1, we give a brief overview of the method, together with the related work. We simplify the method in Section 3.2 and analyze it in more depth in Section 3.3. Section 3.4 discusses the practical limits of the method and the time analysis of the Sage implementation. Section 3.5 is concerned with the real-world impact of the algorithm and covers both the backdoor discussion and our audit. We draw conclusions in Section 3.6.

3.1 Previous work

Chengs's $4p-1$ method is similar to Lenstra's ECM [Len87]. To find a prime factor p of an integer N, both methods use an elliptic curve

$E(\mathbb{Z}_N) \cong E(\mathbb{F}_p) \oplus E(\mathbb{F}_q)$ (see Section 2.1). If $m \in \mathbb{Z}$ is a multiple of $\#E(\mathbb{F}_p)$ and $P \in E(\mathbb{Z}_N)$, then the computation of $[m]P$ often fails (as it requires an inversion of a multiple of p modulo N), which reveals p. The methods differ in the choice of E and P as well as in the computation of $[m]P$. In ECM, we choose random curves E in the hope that $\#E(\mathbb{F}_p)$ is smooth, so we take m as product of small primes (e.g., $m = B!$ for some small B). It is hard to find a point on the curve $E(\mathbb{Z}_N)$ for composite N in general, so we choose P first and pick the curve coefficients accordingly.

In Cheng's $4p-1$ method, we hope that a given D is a square-free part of $4p-1$. The method constructs $E(\mathbb{Z}_N)$ so that the corresponding $E(\mathbb{F}_p)$ is anomalous ($\#E(\mathbb{F}_p) = p$), so we take $m = N$. Since we cannot choose P before E here, it is important that Cheng found a way how to avoid working with points explicitly. Instead of a direct computation of the scalar multiple $[N]P$, he used the N-th division polynomial ψ_N to compute the required $\psi_N(P) = \psi_N(x)$ for a randomly chosen $x \in \mathbb{Z}$, which he hopes to be an x-coordinate of some point on $E(\mathbb{F}_p)$. Cheng's method uses the complex multiplication (CM) method [Bro06; BS07] to construct an anomalous curve. CM computes the j-invariant of the curve as a root of the Hilbert polynomial (HP) $H_{-D}(x)$ in $\mathbb{F}_p$ corresponding to D. There are two different yet related curves (twists) E and E_c with the given j-invariant having exactly $p-2$ and p points, respectively. The curve E is defined by the constants a, b, which can be computed as rational functions of the j-invariant, i.e., $a = a(j), b = b(j)$ defines E. The curve E_c is defined by the j-invariant and some quadratic non-residue c in $\mathbb{F}_p$, i.e., $a = a(j,c), b = b(j,c)$. Since the method cannot distinguish between a curve with p points and a curve with $p+2$ points over $\mathbb{F}_p$, Cheng's method computes $[N]P$ (more precisely ψ_N) for both curves. The method iterates through various values of x (to guess the x-coordinate of some point) and various values of c (to guess the quadratic non-residue), hence two for-loops are used in the method. Cheng stated that the probability of a successful guess of x or a correct twist is $\frac{1}{2}$. Later research [RS07] showed that it is possible to choose the correct twist (having p elements) with some small additional effort (at least with the knowledge of p). However, we will show later that Cheng's method works for both twists without influencing the probability of success.

Cheng introduced his method in 2002 in [Che]. The original method computes the j-invariant as the root of the HP $H_{-D}(X)$ of degree one. Thus in this case we have a concrete value of the j-invariant and are able to construct a concrete curve E over $\mathbb{Z}_N$ (up to a twist). There are only six HPs of degree one that can occur (we are not counting the cases $-D \in \{-4, -7, -8\}$ which are excluded by a congruence condition on D) so the method can be used for a prime divisor p of N of six different forms. In the same year, Cheng generalized his method in [Che02] for HPs of an arbitrary degree. In the generalized version, a concrete j-invariant is not computed (as finding roots of polynomials modulo N is very hard in general), but the method works with j symbolically. In 2017, Shirase published the paper [Shi], where he followed up on Cheng's older publication [Che] (he clearly missed the newer one). Although Shirase "reinvented" Cheng's method only for HPs of degree at most two, the contribution of his work is not negligible. Shirase improved Cheng's unclear description of his method (especially the equation $g(X) = P_N(x) \in Z/(N)[X]$ on page 6 of [Che02] is not clear enough). On the other hand, his description is quite complex and can be simplified.

3.2 A simpler version of Cheng's $4p-1$ method

In our simplified method, we assume that N, the number to be factored, has a prime divisor p satisfying

$$4p - 1 = Ds^2,$$

where D is square-free (note that this immediately implies $D \equiv 3 \pmod 8$). We will also assume $D \neq 3$ (the case $D = 3$ is much easier and is handled separately in [Shi]). For simplicity, we will only deal with the case $N = pq$, where q is also a prime, although this is not a necessary condition. The most important ideas involved in the algorithm are the following:

1. we can control the number of points on a curve $E(\mathbb{F}_p)$ through the CM method – in our case, finding a root j of the HP modulo p and constructing a curve E with j as its j-invariant ensures that $E(\mathbb{F}_p)$ is either p or $p+2$;

2. instead of working with unknown roots $j \in \mathbb{F}_p$ of the HP H_{-D}, we compute symbolically in the ring $Q := \mathbb{Z}_N[X]/(H_{-D}(X))$;

3. division polynomials ψ_N can be used to compute the desired non-invertible denominators ($\psi_N(P) \equiv 0 \pmod{p}$):

$$\mathcal{O} = [N]P = \left(\frac{\varphi_N(P)}{\psi_N(P)^2}, \frac{\omega_N(P)}{\psi_N^3(P)} \right). \tag{3.1}$$

Algorithm 1: Cheng's $4p-1$ factorization [Che], simplified.

```
Input  : $N$ (the integer to be factored);
         $D$ (the square-free part of $4p-1$ for $p|N$)
Output: $p$ (or failure)
compute $H_{-D,N}(X)$ (the $-D$-th HP modulo $N$);
$Q \leftarrow \mathbb{Z}_N[X]/(H_{-D,N}(X))$;
$j \leftarrow [X] \in Q$;
$k \leftarrow j \cdot (1728-j)^{-1} \in Q$ (*);
$a,b \leftarrow 3k,2k \in Q$ ;
choose bound $B$ appropriately (for example $B=10$);
forall $i \in \{1,2,\cdots,B\}$ do
    generate random $x_i \in \mathbb{Z}_N \subseteq Q$;
    $z \leftarrow \psi_N(a,b,x_i) \in Q$ ;
    $d = gcd(\bar{z}(X), H_{-D,N}(X)) \in \mathbb{Z}_N[X]$ (*);
    $r \leftarrow \gcd(d,n)$;
    if $1 < r < N$ then
        return $r$;
return failure ;
```

In Algorithm 1, two operations are marked by (*), since these operations may fail. Both of these operations (the computation of d or the inverse $(1728-j)^{-1}$ in Q) can be performed using the extended version of Euclid's algorithm with polynomials over $\mathbb{Z}_N[X]$ as an input. The problematic step in the Euclid's algorithm is to compute q_k, r_{k-1} such that $r_{k-2} = q_k r_{k-1} + r_k$, when the leading coefficient lc of r_{k-1} polynomial is not coprime to N. However, this means that we can directly return $gcd(lc, N) > 1$.

3.3 Analysis of the method

This section focuses on a clear description and explanation of the method (Section 3.3.1) and its analysis (subsections 3.3.2, 3.3.3). The original Cheng's method computes within two curves – curve defined by the a, b and its twist defined by same a, b and some quadratic non-residue $c \pmod{p}$. Since p is unknown, Cheng's algorithm iterates through various c, though in Section 3.3.2, we show that this c-loop can be omitted, yielding Algorithm 1. Moreover, in Section 3.3.3, we show that the average number of iterations (the x-loop) of the method depends on the class number $h(-D)$ (the degree of the HP $H_{-D}(X)$) and is close to 1 for a large D.

3.3.1 Correctness of the algorithm

Many computations in the algorithm are performed over the quotient ring $Q = \mathbb{Z}_N[X]/(H_{-D,N}(X))$. It's easy to see that the substitution $X \mapsto j$ induces a ring homomorphism $h_j : Q \to \mathbb{F}_p$. In other words, any computation in Q corresponds to a symbolic computation with a root $j \in \mathbb{Z}_N$ of the HP (i.e., $H_{-D,N}(j) \equiv 0 \pmod{p}$). Hence $X \mapsto j$ induces a homomorphism $Q \mapsto \mathbb{Z}_N$, which can be composed with the natural projection $\mathbb{Z}_N \mapsto \mathbb{F}_p$ to obtain the homomorphism h_j. Figure 3.1 depicts the relation of computation in $\mathbb{F}_p$ and Q through the h_j. It should be noted that in Q, we are working symbolically with all roots of the HP modulo p at once.

The key and most time consuming part of the algorithm is the evaluation of the division polynomial ψ_N at $P = (x, y) \in E(\mathbb{Z}_N)$. When N is odd, ψ_N contains y only in even powers [Was08]. Thus we can eliminate y using the defining Weierstrass equation and write $\psi_N(P) = \psi_N(a, b, x)$. The homomorphism h_j maps $\psi_N(a, b, x) \in Q$ computed in the method to $0 \in \mathbb{F}_p$, as Figure 3.1 illustrates.

In Algorithm 1, we compute $\gcd(\bar{z}(X), H_{-D,N}(X)) = d$ for the lift $\bar{z}$ of $z \in Q$ to $\mathbb{Z}_N[x]$. Lemma 4 in [Che02] says that d is a constant from $\mathbb{Z}_N$. Since $h_j(H_{-D,N}(X)) = 0$ and $h_j(\psi_N(a, b, x)) = 0$, we must have $d \equiv 0 \pmod{p}$.

For a further analysis, we will need to understand the structure of Q. Since the $-D$-th HP splits completely modulo p [BS07], we have $H_{-D}(X) \equiv \prod_{i=1}^{h(-D)}(X - j_i) \pmod{p}$ for some pairwise distinct

$$\begin{tikzcd}
\mathbb{F}_p : H_{-D}(j) = 0 \arrow[r] & (a,b) = \left(\frac{3j}{1728-j}, \frac{2j}{1728-j}\right) \arrow[r] & \psi_N(a,b,x_i) = 0 \\
Q : H_{-D,N}(X) = 0 \arrow[r] \arrow[u, "h_j : X \mapsto j"] & (a,b) = \left(\frac{3X}{1728-X}, \frac{2X}{1728-X}\right) \arrow[r] \arrow[u, "h_j : X \mapsto j"] & \psi_N(a,b,x_i). \arrow[u, "h_j : X \mapsto j"]
\end{tikzcd}$$

Figure 3.1: A diagrammatic overview of arithmetic in $\mathbb{F}_p$ and Q.

$j_1, \dots, j_{h(-D)} \in \mathbb{Z}$ (hence the ideals $(X - j_i) \subseteq \mathbb{F}_p[X]$ are pairwise co-maximal). Now let $H_{-D,p}(X), H_{-D,q}(X)$ be the projections of $H_{-D}(X)$ to $\mathbb{F}_p$ and $\mathbb{Z}_q$, respectively. Applying the generalized CRT several times, we obtain the isomorphisms:

$$\begin{aligned}
Q &= \mathbb{Z}_N[X]/(H_{-D,N}(X)) \\
&\cong \big(\mathbb{Z}_q[X]/(H_{-D,q}(X))\big) \times \mathbb{F}_p[X]/(H_{-D,p}(X)) \\
&\cong \big(\mathbb{Z}_q[X]/(H_{-D,q}(X))\big) \times \prod_{i=1}^{h(-D)} \mathbb{F}_p[X]/(X - j_i) \\
&\cong \big(\mathbb{Z}_q[X]/(H_{-D,q}(X))\big) \times \prod_{i=1}^{h(-D)} \mathbb{F}_p.
\end{aligned}$$

In particular, we have $h(-D)$ different projections from Q to $\mathbb{F}_p$, and these are essentially given by lifting an element from Q to $\mathbb{Z}_N[X]$, substituting some j_i into the obtained polynomial and reducing the result modulo p.

3.3.2 Both twists work

If the constructed curve $E : y^2 = f(x)$ (where $f(x) = x^3 + 3kx + 2k$) has p points over $\mathbb{F}_p$, it is clear that for x such that $(\frac{f(x)}{p}) = 1$, the value $\psi_N(x)$ will be zero modulo p (since this x then represents a coordinate of a point on $E(\mathbb{F}_p)$). However, if E has $p+2$ points over $\mathbb{F}_p$, it must be a quadratic twist of some curve $E' : y^2 = x^3 + 3kc^2x + 2kc^3$ for some $c \in \mathbb{F}_p, (\frac{c}{p}) = -1$, such that E' has p points over $\mathbb{F}_p$. Then there is an isomorphism $E \to E'$ over $\mathbb{F}_p(\sqrt{c})$ given by $(x,y) \mapsto (cx, c^{3/2}y)$. Since

c is invertible, this implies that the division polynomials of the curves must also be related by an invertible transformation. More specifically, if we let $\psi_{n,E}(x), \psi'_{n,E'}(x)$ be the division polynomials associated to E and E', respectively, then we have $\psi_{n,E'}(x) = \psi_{n,E}(cx)$. Thus if $\left(\frac{f(c^{-1}x)}{p}\right) = 1$, the value $\psi_N(x)$ will be zero modulo p as well. Since for fixed c the values $c^{-1}x$ have the same distribution as x, we do not have to iterate over the twists and can fix any of them instead.

Moreover, the probability that value $\psi_N(x)$ will be zero modulo p for a fixed curve and a randomly chosen $x \in \mathbb{F}_p$ (more precisely, the projection of a randomly chosen $x \in \mathbb{Z}_N$) is $p_t p_x + (1-p_t)(1-p_x)$, where p_t is the probability of choosing the right twist and p_x is the probability of the event $\left(\frac{f(x)}{p}\right) = 1$.

Thus under the classical heuristical assumption that $p_t = \frac{1}{2}$ (or alternatively, after calculating that p_x is very close to $\frac{1}{2}$), the above probability is $\frac{1}{2}$.

3.3.3 Expected number of iterations

Now we can estimate the probability that the core part of the algorithm will work. During scalar multiplication on a curve over a product of rings, the rational functions in Equation (3.1) can be computed coordinate-wise. This might be problematic when the result is a "point in semi-infinity" (i.e., infinite in only some coordinates)[2], but that is exactly what we want, as one of the denominators will then reveal a factor of N.

Thus when we have an elliptic curve over

$$Q \cong \left(\mathbb{Z}_q[X]/(H_{-D,q}(X))\right) \times \prod_{i=1}^{h(-D)} \mathbb{F}_p,$$

the algorithm will succeed for a fixed $x \in \mathbb{Z}_N$ whenever there is at least one copy of $\mathbb{F}_p$ over which the x corresponds to the right twist (unless this happens over all of the copies at the same time and simultaneously over $\mathbb{Z}_q[X]/(H_{-D,q}(X))$, which is extremely unlikely, as q

2. This issue can be fixed by a more general definition of a projective space over satisfying the assumptions in Section 2.1.

has no relation to $H_{-D}(X)$). Heuristically, these copies of $\mathbb{F}_p$ behave independently, so by the argumentation in Section 3.3.2, the estimated probability that one iteration of the loop over x_i's in Algorithm 1 reveal p is $1-2^{-h(-D)}$. Therefore the expected number of the times the loop will have to be executed is close to

$$\frac{1}{1-2^{-h(-D)}} = \frac{2^{h(-D)}}{2^{h(-D)}-1}.$$

Thus when $h(-D)=1$, one iteration of the loop will work with probability around $\frac{1}{2}$, but for a large $h(-D)$, the probability is almost 1 and the algorithm becomes almost deterministic. These claims are also supported by an empirical evidence in Section 3.4.2.

Note that this is a better result than in both [Che] and [Shi], where both twists are non-deterministically tested and the expected number of execution times of the innermost loop is claimed to be around 4.

3.4 Time analysis and practical limits of the method

When we do not know D in advance, we could try to loop through all possible values of D up to some bound. This yields the complexity $(D \log n)^{O(1)}$ [Che02], as the computation of the $-D$-th HP is exponential in D, while all other parts of Algorithm 1 can be performed in a time polynomial in $\log N$ and D. Compare this to Pollard's $p-1$ method with complexity $(B \log n)^{O(1)}$, where B is the largest prime factor of $p-1$). When D is small (or known), this is polynomial in $\log N$, which is asymptotically much better than for any general classical non-quantum algorithm.

This quickly becomes inefficient for larger values of D, for several reasons. The degree of the HPs grows quite fast, which complicates both the computations in the ring Q and the computation of the HPs themselves, and their coefficients grow even faster, which might eventually become a memory problem.

It is possible to compute the $H_{-D,N}$ (H_{-D} modulo N) directly [Sut11] without computing H_{-D}, which significantly decreases the memory cost. For instance, H_{-D} takes about 93 GB to store for $D = 2\,093\,236\,031$ while $H_{-D,N}$ takes only 24 MB for 4096-bit N as the degree of the H_{-D} is 100000.

The main practical limit is still the fact that the method is only applicable to numbers of a special form. For expected density results about these numbers, see Section 3.4.1.

3.4.1 The expected occurrence of factorable numbers

We will limit ourselves to the RSA case here, because it is probably the most important application of integer factorization in the real-world. Let us take a look at the expected frequency of factorable numbers. First, let us assume that D is fixed and that p is a random $2b$-bit integer, so that $2^{2b-1} < p < 2^{2b}$. The condition $4p-1 = Ds^2$ is equivalent to $\frac{4p-1}{D}$ being a square of an odd integer. Since

$$\frac{2^{2b+1}}{D} < \frac{4p-1}{D} < \frac{2^{2b+2}}{D}$$

and the number of odd integer squares in the interval $\left[\frac{2^{2b+1}}{D}, \frac{2^{2b+2}}{D}\right]$ is roughly

$$\frac{1}{2}\left(\sqrt{\frac{2^{2b+2}}{D}} - \sqrt{\frac{2^{2b+1}}{D}}\right) \approx \frac{2^{b-2}}{\sqrt{D}}, \tag{3.2}$$

the number of possible $2b$-bit primes such that the square-free part of $4p-1$ equals D can be roughly estimated as $\frac{2^{b-2}}{\sqrt{D}}$. Since the total number of $2b$-bit primes is around

$$\frac{2^{2b}}{\ln(2^{2b})} - \frac{2^{2b-1}}{\ln(2^{2b-1})} \leq \frac{2^{2b}}{b} \tag{3.3}$$

by the Prime number theorem [Gol73], we can roughly estimate that the probability that a random $2b$-bit prime is vulnerable to factorization with respect to a given D is around $\frac{b}{\sqrt{D}\cdot 2^{b+2}}$ (for $D = 11$ and $2b = 1024$, this is around 2^{-507}).

Not let us consider all D's up to some bound B instead of a fixed D. Summing up the easy inequalities $\frac{1}{\sqrt{k}} < 2\sqrt{k} - 2\sqrt{k-1}$ for $k = 1, \dots, B$ and adding $\frac{1}{\sqrt{1}} + \frac{1}{\sqrt{2}}$ to both sides yields

$$\sum_{k=1}^{B} \frac{1}{\sqrt{k}} < 1 + \frac{1}{\sqrt{2}} - 2\sqrt{2} + 2\sqrt{B} < 2\sqrt{B}.$$

Now (3.2) implies that the number of possible $2b$-bit primes such that the square-free part of $4p-1$ equals $D < B$ can be bounded by

$$\sum_{\substack{D=3 \\ D\equiv 3 \pmod 8 \\ D \text{ is square-free}}}^{B} \frac{2^{b-2}}{\sqrt{D}} \leq 2^{b-2} \sum_{D=1}^{B} \frac{1}{\sqrt{D}} < 2^{b-1} \cdot \sqrt{B},$$

which together with (3.3) gives an estimate that the probability that a random $2b$-bit prime is vulnerable to factorization with respect to some $D < B$ is at most $\frac{\sqrt{B}b}{2^{b+1}}$ (for $B = 2^{54}$ and $2b = 1024$, this is 2^{-477}).

3.4.2 Run-time statistics

Implementation details. We implemented[3] the algorithm in Sage, an open-source computer algebra system. We note that to the best of our knowledge there is no other implementation available for the vulnerable primes based on the same principle at the time of writing the article.

Since most of the mathematical utilities needed are already implemented in Sage, the code is compact and easy to use (although it could probably be optimized even more). The only subtlety was the need to set the internal recursion limit to 20 000 in order to compute the N-th division polynomial (for N much larger than 2^{2048}, this should probably be increased even more).

Experiment. The factorization algorithm complexity is mainly determined by the class number $h(-D)$ – degree of the HP H_{-D}. We sampled the function $h(-D)$ over the square-free discriminants $-D$ ($D \equiv 3 \pmod 8$), so that we could measure the running time of the algorithm with the smallest discriminant per given class number. To practically measure the running time of the factorization algorithm, we performed the following experiment. For each $h(-D) \in [1,1000]$, we took the smallest absolute value of the discriminant $-D$ found, obtained by sampling as described above. For each discriminant, we randomly generated three composites pq with both the vulnerable prime p and a random prime q of bit-size $b \in \{256, 512, 1024, 2048\}$.

3. Our implementation is available at `https://crocs.fi.muni.cz/public/papers/Secrypt2019`.

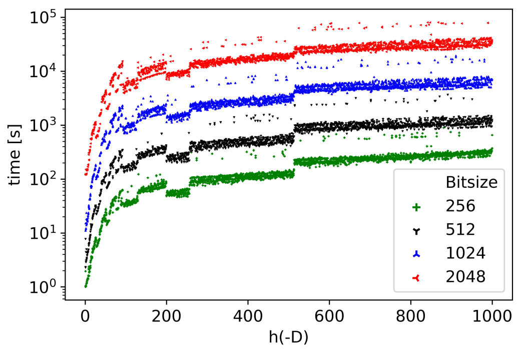

Figure 3.2: Observed running times of the factorization algorithm for composite bit-sizes $b \in \{256, 512, 1024, 2048\}$ bits for the smallest discriminant found per class number. Three composites with the vulnerable prime of the given bit-size were randomly generated per discriminant.

Figure 3.2 depicts the results of the experiment, i.e., the overall running time of the factorization algorithm for composite N with respect to the given class number. Also, the relation between D's and their corresponding class numbers is depicted in Figure 3.3, where we can see that the degree $h(-D)$ of the HP oscillates even for close values D.

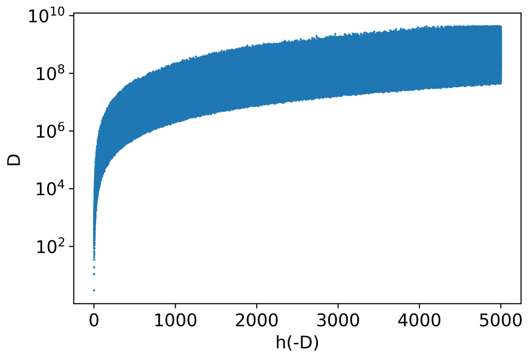

Figure 3.3: Log-scale of D sampled from the interval $[0, 2^{32}+3]$ and corresponding $h(-D)$.

In comparison, Boudot et al. [Bou+20] hold the current factorization record – they factored a 795-bit RSA modulus in about 1000 core years.

Run-time independence on D. The parameter D affects the coefficient sizes and computation time of the HP H_{-D}. Besides that, the D does not affect the rest of the algorithm. The computation of H_{-D} is also easily parallelizable. As we compute H_{-D} modulo N, from a certain class number, e.g., class number 110 for 4096-bit modulus, the coefficients of the H_{-D} become larger than N, thus the complexity depends only on $h(-D)$.

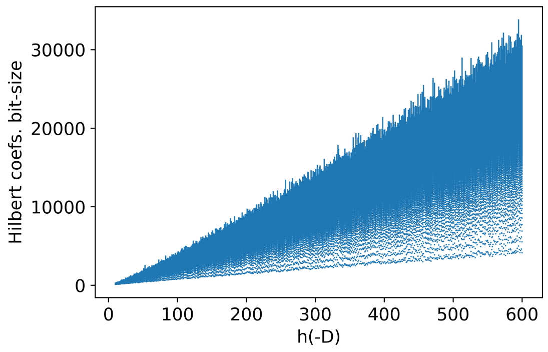

Figure 3.4: Bit-sizes of all HP coefficients for the smallest D corresponding to the given class number. The figure illustrates run-time independence on D as coefficients quickly grow over N.

Figure 3.4 demonstrates the growth of the coefficients of $H_{-D}(X)$. In comparison, Figure 3.5 shows how the computation time is affected by D although the class number is the same (in the case where reduction modulo N is only done afterwards).

Modulus bit-size complexity. As seen from the experiment, the modulus bit-size contributes to the overall complexity of the factorization algorithm by a linear factor $O(log(b))$ with respect to the class number as the modulus size mainly affects the division polynomial. This enables us to empirically study the factorization algorithm mainly with respect to the class number with the lowest such D and with the lowest bit-size to reduce computation time without affecting the results validity. Figure 3.6 depicts the linear model curve fitting over

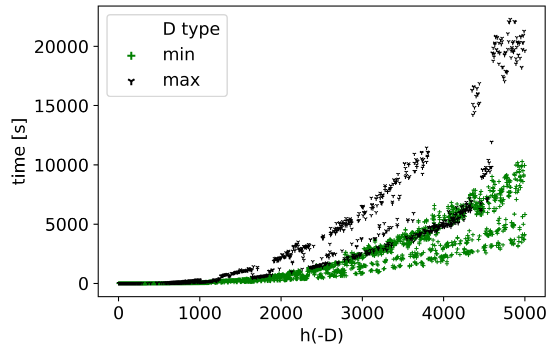

Figure 3.5: The time computation of the HP and minimal/maximal sampled D values for class numbers $h(-D)$ in $[1,5000]$.

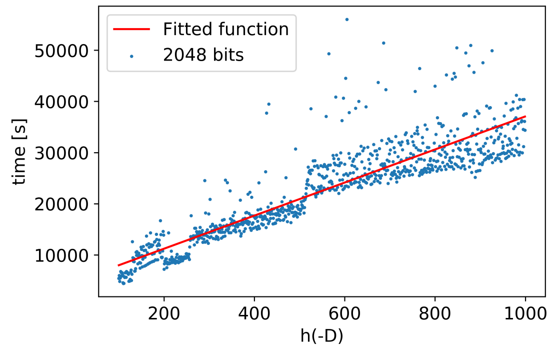

Figure 3.6: Running time for the factorization algorithm w.r.t. $h(-D)$ and fitted linear function for 2048 bit prime size.

2048 prime based moduli and Table 3.1 shows the linear models fitted for all tested bit-sizes.

Component timing. The computation of the division polynomial is by far the most expensive operation for class numbers under 1000 (and even for higher ones if the HP is computed modulo N directly). As class numbers grow over 1000, the $H_{-D}(X)$ computation becomes more significant. Figure 3.7 illustrates the factorization algorithm timing by two components, the evaluation of the division polynomial and HP computation for $b = 256$. Around the class number 2000, the component timing becomes equal. For higher class numbers, the $H_{-D}(X)$ computation asymptotically dominates the overall computation time.

p bit-size	Fitted model		
256	$0.33887x$	$-$	1.17973
512	$1.23834x$	$+$	81.44157
1024	$6.57677x$	$+$	519.07422
2048	$32.7223x$	$+$	4614.71032

Table 3.1: Runtime linear model fit with respect to the class number.

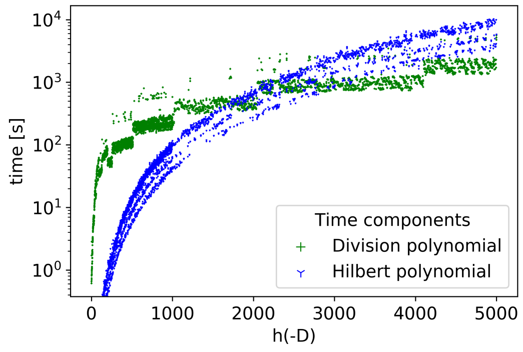

Figure 3.7: Algorithm log run-time breakdown to two major components: the evaluation of the division polynomial and the computation of $H_{-D}(X)$ for $b = 256$.

Inner loop iterations. Observe the number of inner loop iterations in the depicted dataset. From the total number of experiments 12 000 $(1000 \cdot 4 \cdot 3)$, only 12 experiments needed more than one iteration. In total, the average number of iterations is 1.001834. The class number for all experiments requiring more than one iteration was in the interval $[1,4]$, which supports our claim that the number of expected iterations quickly converges to 1 with higher class numbers.

Computation resources. Due to the heterogeneous nature of the cluster and the job scheduling system, the jobs were allocated different processors types, namely Intel Xeon Gold 5120 2.20GHz, Gold 6130 2.10GHz, E5-2630 v3 2.40GHz, E5-2650 v2 2.60GHz. The worker nodes are shared among other users, which affects caches of the processor and thus the overall system performance. Due to the mentioned irregularities, the timing measurements are approximate. However, the jobs were allocated across all CPU types randomly.

Running time step-changes. There are noticeable changes in the running time of the factorization algorithm for some class number ranges. Even though the experiment jobs ran on a cluster with varying load and processor types, we conclude these regions are not a result of a systematic error as for each discriminant there were three random composites generated, this was performed for all four bit-sizes, thus it gives 12 different experiment jobs per single D. The effect is observable in all bit-sizes in all experiments. The regions are present even after the re-computation of the region in further validation experiments. As the division polynomial computation is the main running time component, we conclude the regions are a result of the particular Sage implementation, depending on the class number. Currently, we have no detailed explanation of the phenomena, and it remains an open problem.

3.5 The $4p-1$ method as a backdoor

The analysis from the previous section shows that if the RSA primes are sufficiently long and generated randomly, it is almost impossible for the resulting public key to be $4p-1$ factorable in practice. Taking the contrapositive, if a public RSA key is $4p-1$ factorable, there is an overwhelming probability that at least one of the primes was generated in this way on purpose, instead of being vulnerable by chance.

This could be interesting from the viewpoint of kleptography [YY97]. It would be possible to backdoor the prime number generation methods in black-box devices (such as smartcards or Hardware Security Modules (HSMs) to generate prime(s) p such that the square-free part of $4p-1$ is relatively small (as generating such primes is very easy). We first describe the backdoor construction process and later elaborate on the prospective detection methods, showing that the existence of the backdoor cannot be ruled out for the longer key lengths like 2048 bits, if only keys (including private primes) are available for the analysis.

In contrast, the RSA prime number generation in a wide range of open-source cryptographic libraries was already analyzed with no such backdoor found [Sve+16].

3.5.1 The backdoor construction

In this section, we investigate the properties of Cheng's $4p-1$ method when used as a cryptographic backdoor intentionally producing moduli that are factorable. Namely, we analyze the possibility that the backdoor with a particular choice of D will be both reasonably efficient to exploit for an attacker with the knowledge of chosen D (so he can compute the factorization), yet very hard to detect by an *Inquirer*. We define the *Inquirer* according to [YY97] as a person examining the (large number of) generated keys from a potentially backdoored implementation for the statistical presence of any characteristics hinting at the existence of the backdoor. The *Inquirer* wins if the backdoor is detected with non-negligible probability. The attacker wins if the presence of the backdoor is not detected, yet the attacker can still factorize the resulting keys in a reasonable time frame.

The use of the method as a backdoor has three phases: 1) selection of suitable backdoor parameters, 2) generation of backdoored prime(s), and 3) factorization of a given (backdoored) public key:

1. An attacker selects a value D with a suitably small class number $h(-D)$. An attacker can use either a single fixed D (or a small number of them) for all backdoored primes or generate a separate D for every backdoored key.
2. During the RSA keypair generation, the first prime is generated at random (non-backdoored), while the second one is constructed as follows:
   (a) Generate randomly an odd number s with the length corresponding to the required length of prime.
   (b) Compute candidate prime p as $p = \frac{Ds^2+1}{4}$.
   (c) Check if candidate p is probable prime using, e.g., the Miller-Rabin primality test.
   (d) Output p if probable prime, or repeat the construction with a different value of s if not.
3. The given public key is factorized using Algorithm 1 as described in Section 3.2.

Method advantages for use as a backdoor.

- All standard RSA key lengths now assumed secure can be backdoored (including 2048, 4096 and 8192-bit lengths).
- No observable bias present in the public keys (if the second prime is chosen at random and the proper distribution of s is chosen).
- A favorable ratio between the factorization time with the knowledge of D (an attacker) and the time required by *Inquirer* to detect the existence of such a D (see Figure 3.8).
- The adjustable factorization difficulty using value D with suitable class number $h(-D)$.
- The good parallelizability for the HP computation part of the factorization [Sut11] which dominates for the sufficiently large class number – see Figure 3.7.
- The expected number of invocations of the Miller-Rabin primality test during the keypair generation is heuristically same as for the situation with truly random (non-backdoored) primes.

Method disadvantages for use as a backdoor.

- Easy detection of the backdoor presence if private keys are available for inspection and the D is reused (two methods are discussed in Section 3.5.2).
- Since D needs to be unique for every keypair, it has also to be established quickly.
- The backdooring of keys with short lengths (1280 bits and below) is detectable even when unique D is used (Method 1).
- If D is leaked, the backdoored keys with this specific D become exploitable by anyone (not "only us").

3.5.2 Inquirer detection strategies

We propose three principally different methods to detect the presence of backdoor for the different scenarios concerning the availability of private keys for inspection and the length of the inspected keys.

Method 1: Inquirer with access to the public keys only. An *Inquirer* picks a candidate D_i value, assumes the key being backdoored with this D_i attempts to perform the factorization using $4p-1$ method. If successful, both the presence as well as the actual parameter D_i used is revealed. The naïve method would be to examine all possible values D_i, starting from 11 until the allowed examination period is exhausted (e.g., at least 1000 virtual CPU years worth of computation). Note that an attacker aims to use such a D that has the corresponding class number $h(-D)$ as small as possible to achieve as fast factorization as possible. Figure 3.3 shows the relation between the value D and its $h(-D)$.

Even if unsuccessful, this examination establishes a lower limit on the computational time that an actual attacker needs for the factorization of a key as seen from Figure 3.3.

Figure 3.2 shows the running time to factor a composite N with a particular choice of D, which is only known to the attacker who generated N in this way, i.e., using it as a potential backdoor. The experiment illustrates the growth of the factorization complexity for an attacker knowing the D. On the other hand, an *Inquirer* trying to detect such a backdoor and without the knowledge of particular D has to try all possible D's up to the D_{max}. The detection complexity is thus the sum of all factorization times up to the D_{max} (or surface under the curve up to the D_{max}). For an illustration of such case, see Figure 3.8.

Method 2: Inquirer with access to the private key(s) with shorter primes (up to $\sim$ 768 bits). An *Inquirer* performs the direct factorization of $4p-1$ value by generic-purpose factorization method. The resulting factors are then checked for the existence of unexpectedly small D (or its multiplies), which would implicate the possibility to use $4p-1$ method for factorization and thus a presence of the backdoor. The remaining part must be also eligible for square root computation. The expected size of D for a truly random (non-backdoored)

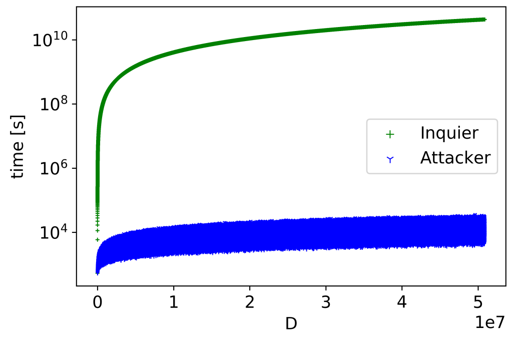

Figure 3.8: Estimated factorization times for an Inquirer (without knowing D) and an attacker (knowing D) up to the lowest D for class number 5000 and bit-size 1024. The inquirer tries all D's up to the actual D.

prime is large (around the bit-length of the tested prime, see Figure 3.9 for the experimental results from 10000 random primes), so a small D is unexpected from non-backdoored keys.

Method 3: Inquirer with access to a large number of private keys. An *Inquirer* collects large number of private keys generated by inspected black-box implementations and computes the batch-GCD algorithm [Hen+12] over all $4p-1$ values constructed from the corresponding primes. Would the same D be used for any two primes, batch-GCD will succeed in factorization, revealing the presence of the backdoor as well as D used. This method is usable also for larger key lengths than would be Method 2, efficiently analyzing 2048-bits keys and longer.

Here we describe the batch-GCD method. For all i, let

$$g_i = \gcd\left(4p_i - 1, \prod_{i \neq j}(4p_j - 1)\right).$$

If $4p_i - 1 = D_i s_i^2$ and $4p_j - 1 = D_j s_j^2$, we can see that $D_i = D_j$ implies $D_i | g_i$. Thus we factor each $g_i = \prod q_k^{e_k}$, compute a candidate $D_i' = \prod q_k^{h_k}$, $0 \leq h_k \leq e_k$, i.e., a divisor of g_i, such that $D_i' \equiv 3 \pmod 8)$ and D_i' is square-free. If $\frac{4p_i - 1}{D_i'}$ is a perfect square for some D_i', we found D_i, a square-free part of the $4p_i - 1$.

As an *Inquirer* can collect and investigate a large number of private keys during batch-GCD, the probability of not investigating at least one pair of two primes with the same D quickly decreases due to the Birthday paradox. This motivates any sensible backdooring attacker to use different D for every new prime generated. Having a unique D generated in turn creates the need for efficient reconstruction of the D's value on an attacker's side, e.g., leaking it in additional information like padding or maintaining the large database of all the Ds used.

3.5.3 Audit of real-world keys

We collected a large dataset of 512, 1024 and 2048-bit RSA keypairs generated by fifteen different cryptographic smartcards and three HSMs with both public and private keys stored (44.7 million keypairs in total). As we knew the keypair primes, we direcly use *Inquirer* methods 2 and 3 to search for a D and attempt to detect a potential backdoor.

Application of Method 2: Factorization of $4p - 1$. We used a randomly selected subset from all keys collected with 5 000 512-bit RSA keypairs and 100 public 1024-bit RSA keys for every inspected device. Each prime is analyzed for vulnerability to the $4p - 1$ factorization method, using Algorithm 1 implemented by the Sage computer algebra system for the actual computation.

We factored $4p - 1$ (and $4q - 1$) and computed their square-free parts. In the majority of cases, the square-free parts were the numbers themselves, and the smallest square-free part found having 490 bits in the 1024-bit case and 229 bits in the 512-bit case. Thus these public keys are far from being $4p - 1$ factorable, and it would be impractical to use the $4p - 1$ factorization method on these keys. In fact, if these keys could be factored with the method, then so would be any randomly generated keys of the same bit-size. Section 3.5.3 further discusses the observed results. Note, that we were not able to completely factor a small portion of these numbers in the given time frame (2 hours for one number), but since the Sage factorization algorithm contains a square test and revealed prime factors as large as 110 bits in other cases, we can be reasonably sure that the square-free parts of these unfactored numbers are much larger than 2^{54} as well.

Application of Method 3: Batch-GCD. We used all 44.7M collected private keys, including the 2048-bit keys (which are unsuitable for Method 2 due to their length) to look for the shared value of D with the batch-GCD algorithm [Hen+12]. We also added $\#D = \prod_{D \leq 50868011} D$, i.e., the product of all square-free D's congruent to 3 modulo 8 up to the minimal D with $h(-D) = 5000$ to a batch-GCD dataset.

We found that no two primes share a common square-free part D in $4p - 1$ and due to $\#D$ all Ds used have to be greater than 50868011. Therefore, we can conclude that if the the backdoor is present, each prime has to have its own unique D (as reusing any D is very unlikely to be missed as it would have to be drawn from a set of $(44.7M)^2$ possible Ds due to the Birthday paradox to evade detection on our dataset). Note that a unique D also means, that an attacker must be able to 1) infer the D used for the given public key and 2) compute the HP for this specific D, slowing down the subsequent factorization.

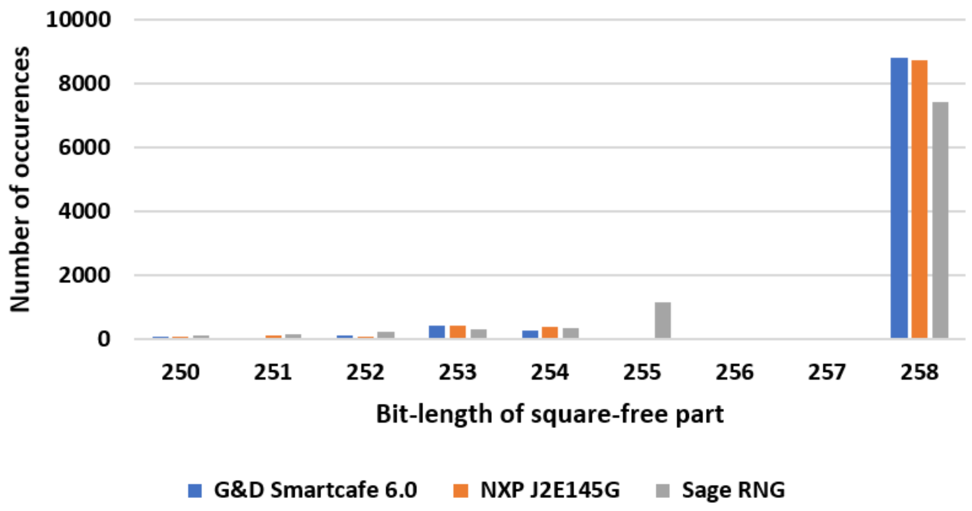

Figure 3.9: Histogram of bit-lengths of square-free parts obtained from the factorization of $4p - 1$ values constructed from 10000 primes found in 512-bit RSA keys. All other devices than explicitly listed produced a distribution undistinguishable from the one of the random primes generated by Sage (Sage RNG). The reason for the observed differences are explained in Section 3.5.3.

Distribution of square-free parts.

We took the distribution of the square-free parts of $4p-1$ and $4q-1$ obtained by applying Method 2 to every analyzed device and compared it to the reference distribution for p and q generated randomly by Sage. No significant differences were found, with two exceptions – G&D SmartCafe 6.0 and NXP J2E145G smartcards, as shown on Figure 3.9. Here, we explain the reason for the observed differences.

The expected probability that $4p-1$ is square-free for large p is

$$1-\sum_{r\text{ an odd prime}}\frac{1}{r(r-1)}\approx 0.748$$

(established experimentally from 10^6 random primes generated by Sage), as for any odd prime r, $4p\equiv 1 \pmod{r^2}$ iff $p\equiv\frac{1}{4} \pmod{r^2}$ and there are exactly $r(r-1)$ residue classes modulo r^2 that can contain p. This is consistent with the experimental results obtained from both Sage and most cards. However, we observed from [Sve+16] that G&D SmartCafe 6.0 avoids primes p such that $p-1$ is divisible by 3 or 5, while NXP J2E145G avoids primes p such that $p-1$ is divisible by any number between 3 and 251 inclusive. If $p\not\equiv 1 \pmod 3$, then

$$4p-1\equiv p-1\not\equiv 0 \pmod 3$$

(so that $4p-1$ cannot be divisible by 9, which would otherwise happen with probability $\frac{1}{2\cdot 3}$). However, we did not account for the effect of this condition on other primes r, so the probability that $4p-1$ is square-free will not increase by $\frac{1}{6}$ in this case, but only by 0.148 (for convenience again found experimentally). Yet still, forbidding the case $p\equiv 1 \pmod 3$ increases the resistance to the factorization (even if only very slightly). This case is special because

$$4p-1-(p-1)=3p\equiv 0 \pmod 3.$$

Conversely, forbidding the case $p\equiv 1 \pmod r$ for $r\neq 3$ decreases this resistance (although even more marginally), as this leads to forbidding r "good" possible residue classes of $4p-1$ modulo r^2 (note that $1\not\equiv\frac{1}{4} \pmod{r^2}$), so that the probability that $4p-1$ will be divisible by r^2 will be $\frac{1}{r(r-1)-r}=\frac{1}{r(r-2)}$ instead of $\frac{1}{r(r-1)}$ in the case that the condition $p\not\equiv 1 \pmod r$ would not be imposed.

For sufficiently large primes, we experimentally found that if $p-1$ is not divisible by 3 nor 5, the probability that $4p-1$ is square-free is rougly 0.88. If $p-1$ has no factor between 3 and 251, the probability is roughly 0.875, which closely matches the results obtained from G&D SmartCafe 6.0 and NXP J2E145G smartcards, respectively.

3.6 Conclusions

We proposed an improved version of Cheng's $4p-1$ method and thoroughly analyzed it, both theoretically and empirically. We conclude that even though the $4p-1$ factorization method is powerful in theory, it does not seem to have any impact on real-world applications due to a very limited set of numbers on which it can be applied, occurring extremely rarely if the primes are randomly generated.

However, an attacker may *intentionally* generate the primes to result in the factorable keys to form so-called *kleptographic* attack, especially in the black-box devices like cryptographic smartcards. We therefore analyzed more than 44 millions of keypairs generated by 15 smartcards and 3 HSMs and found no indication of the backdoor. We were able to rule out the existence of this backdoor for the key lengths of 512 and 1024 bits, where the detection method based on the full factorization (Method 2) is applicable as no small D was found.

Unfortunately, we cannot rule out the presence of the backdoor in keys with longer lengths, like 2048 bits, despite of the availability and inspection of the private keys. An attacker may use a unique D for every prime generated, thus evading the detection by batch-GCD based method (Method 3). The complete backdoor detection (or its exclusion) is still an open question.

As already mentioned in [Che02], there are several other possibilities for future work on the topic of $4p-1$ factorization, including the exploration of the possibility of using Weber polynomials instead of Hilbert polynomials (whose coefficients do not grow as quickly), using curves of a higher genus or studying the discrete logarithm problem for primes of the same structure. Moreover, the inherent asymmetry of the factorization with and without the knowledge of D could prove useful in the construction of some cryptosystems.

4 Fooling primality tests on smartcards[1]

Many public key cryptosystems crucially rely on prime numbers for their security. Yet for performance reasons (especially on constrained devices such as smartcards), most widely used primality tests, such as the Miller-Rabin (MR) test [Mil75; Rab80], are only probabilistic [Ble05; Alb+18]. Thus there exist *pseudoprimes*, i.e., composite numbers passing these tests. When implemented correctly, probabilistic tests still provide a sufficient assurance of primality. However, carefully crafted pseudoprimes [Arn95b] can fool an implementation that is not utilizing enough randomness [Alb+18]. In (EC)DH and (EC)DSA, this can lead to private key recovery, using Pohlig-Hellman [PH78] style attacks.

JavaCard [Ora19] is a popular platform for building systems based on programmable smart cards. It offers a Java-like environment on which multiple applications, *applets*, can be installed. Thanks to Javacard's rich cryptographic API (supporting (EC)DSA, (EC)DH and much more [Sve19]), these applets include electronic passports and IDs, EMV applets for credit-cards, key managers, cryptocurrency wallets or applets for two-factor authentication. While the API is defined by an open standard, the implementation of the platform itself is almost always proprietary, with manufacturers releasing very little information about the code used in a particular family of cards. This black-box nature makes the public assessment of implementation security more difficult, but nevertheless, security problems have been discovered in the past [Nem+17].

In this work, we test the robustness of present primality tests in JavaCards by replacing (EC)DSA and (EC)DH prime parameters with MR pseudoprimes and other composites. In contrast to [Alb+18], we do not have access to the code inside the smartcard and are not able to call a primality testing function on its own. Instead, we resort to performing standard operations (such as signature generation) using modified parameters (which still need to have specific properties), and observe any deviations from the expected behaviour. This is further

1. The results in this chapter were published at European Symposium on Research in Computer Security (ESORICS) 2020 [SJS20]. See `https://crocs.fi.muni.cz/papers/primality_esorics20` for additional materials.

complicated by the fact that the smartcards do not act deterministically, do not have debugging functionality, and are prone to many errors.

The main contributions of this research are:

- We open the topic of fooling primality tests on black-box devices and propose a method for a systematic review of primality tests (and the relevant domain parameter validation) in black-box devices that use (EC)DSA/(EC)DH.
- We develop new ways in which parameters can be replaced with pseudoprimes in (EC)DH and (EC)DSA, along with practical attacks against these parameters. In particular, the attack against composite p in ECDSA is new to the best of our knowledge.
- We examine the implementation security of ECDH and ECDSA in nine smartcards from five major manufacturers, showing that all cards but one are vulnerable due to insufficient primality testing of domain parameters. Issues found were responsibly disclosed to affected vendors.
- We systematically survey the relevant attacker scenarios and types of attacks with possible real-world impact and propose defence mechanisms.

We review the previous work on attacking primality tests in Section 4.1. Section 4.2 analyzes the attack scenarios and briefly presents possible attacks. The methodology for testing the cards is given in Section 4.3, along with a basic explanation of the used domain parameters. Readers interested only in practical security should feel free to skip this section, while still grasping most of the contents of Section 4.4 that analyzes the testing results, and Section 4.6 that follows up with a discussion of proposed defences. Section 4.5 provides technical details about the full parameter generation and possible attacks and Section 4.7 concludes this chapter. Finally, the appendices contain an overview of the MR test (Appendix B.1), the pseudoprime construction (Appendix B.2), datasets of generated domain parameters (Appendix B.3) and example implementations of concrete attacks (Appendix B.4).

4.1 Previous work

The idea of breaking a cryptographic protocol by fooling primality tests was first mentioned in [Ble05]. Albrecht et al. [Alb+18] analyzed primality tests in open-source libraries, and fooled many of them with carefully crafted pseudoprimes. Their construction (extending the one in [Arn95b] and briefly described in Appendix B.2) relies on the assumption that the implementation of the MR test uses only a small number of bases that are either fixed or chosen from a relatively small set. This was indeed the case for many libraries.

Note that all the libraries inspected in [Alb+18] had a dedicated function for primality testing whose source code was accessible. In contrast, the situation for black-box devices where the code is not known, and the primality test (if present) cannot be separated from the rest of the program, has not been studied before to the best of our knowledge.

Furthermore, somewhat practical examples of attacks against various (EC)DH implementations with insufficient primality tests, including the case when pseudoprimes are included in elliptic curve domain parameters, were described in [GMP19].

4.2 Attack scenarios

As in [Alb+18], we assume a setting where the attacker can control or affect the cryptosystem domain parameters used by the applet – so that primes can be replaced by composites – and wants to break the confidentiality of (EC)DH or unforgeability in (EC)DSA. We also assume that the attacker knows the factorization of the injected parameters, as he most likely crafted them himself.

However, with the exception of primality testing, we still expect that all of parameter validation is implemented properly (with the exception of a cofactor check of an elliptic curve, as the cards lack the performance to do it).

In our scenario an *applet* developed by an *applet developer* uses the functions of the JavaCard API on a card supplied by a *manufacturer* to perform some cryptographic operations while allowing untrusted parameters provided by the attacker to be used.

4.2.1 Rationale for the attack scenarios

To explain the rationale behind such a scenario, we consider the specifics of the JavaCard environment as well as existing cryptographic protocols and standards. Note that physical access (as is commonly relevant for the smartcard usage domain) is often not required.

A JavaCard applet developer might use untrusted domain parameters, because:

- The provided API functions that set parameter values, such as `ECPrivateKey.setFieldFP()`, place no limitation (except bit-sizes) on the parameters, which are provided as sequences of bytes and are interpreted as unsigned integers.
- The API documentation contains no security notice that the set parameters should be trusted or a warning of what are the consequences of setting domain parameters that are untrusted or otherwise invalid [Ora19].
- The API contains no functionality for direct primality testing or domain parameter validation for (EC)DSA or (EC)DH and no way to implement it efficiently. Thus the developer might (understandably) assume that the validation is performed implicitly.

Multiple protocols allow to transmit the domain parameters and thus force a party to either authenticate or validate them:

- TLS, up to version 1.2 [Bla+06a] and prior to RFC8422 [NJP18], allowed explicit (EC)DH parameters to be sent from the server to the client, although authenticated by the server public key.
- The certificate format specified in the X.509 standard allows public keys to hold full domain parameters for (EC)DH or (EC)DSA [PHB02]. Using this format in a JavaCard applet (e.g., for interoperability reasons) might lead to untrusted parameters being used.
- The ICAO document 9303 [ICAO15] specifying the security requirements for machine-readable travel documents allows

transmitting the (EC)DH domain parameters in the Chip Authentication and PACE protocols. The specification warns that insecure domain parameters will cause leaks of secret data and that parameters should not be used unless explicitly known to be secure (without further elaboration). As the card transmits the parameters to the reader, it is the one responsible for the validation.

All relevant (EC)DH and (EC)DSA standards specify procedures for validating the domain parameters and allow the use of untrusted domain parameters provided the validation succeeds. For (EC)DSA, two standards specify the validation requirements:

- FIPS 186-4 [NIST13] refers to the NIST Special publication 800-89 [NIST06] that in turn requires the primes used in the domain parameters to be accompanied by a seed and verifies they were generated using the specified verifiably random method.
- ANSI X9.62 [ANSI98] requires a primality test of the prime domain parameters, using the MR test with the number of rounds equal or larger than 50, using random bases. The IEEE P1363 [IEEE00] standard for (EC)DH has exactly the same requirement.

The strong requirements for primality testing and domain parameter validation in the above standards might lead the applet developer to believe that an appropriate validation is performed by the card and that the use of given parameters is secure. As the detailed implementation guidance is not provided by JavaCard specifications and recommendations from standards like IEEE P1363 and X9.62 are not explicitly mentioned, the platform vendor is left with decision what level of checks to implement.

We also consider another scenario where primality testing and domain parameter validation make a significant difference in security. TLS is an open system where communicating parties are likely to be realised by different software vendors. In the case of closed systems like dedicated network line encryption boxes, the same entity configures both communicating endpoints, which may be based on the commodity cards. A *platform integrator* (not the same as the card *manufacturer*) supplies the software responsible for setting the domain parameters on both ends. These two endpoints are designed to communicate with each other and to establish a secure channel using (EC)DH (and potentially (EC)DSA for authentication). Without robust primality testing and domain parameter validation on the card, the domain parameters supplied to cards at both ends can contain pseudoprimes or composites and be weak to a passive eavesdropping attacker. These parameters can even be authenticated by the *platform integrator*, yet without proper validation and primality testing, the card will accept them. The *platform integrator* could then also claim some plausible deniability, by blaming the weak parameters on a bug in the customised curve generation codebase or arguing the pseudoprime in the parameters passed their primality tests. A similar case happened in the Juniper Dual EC incident [Che+16], where the exploitable weakness was a result of a series of small coding errors, seemingly unintentional.

One example of a vulnerability, where attacker-controlled domain parameters were used, was the Microsoft CryptoAPI ECDSA verification vulnerability (CVE-2020-0601) [Age20]. It was due to a faulty certificate verification mechanism, which matched certificates provided to the trusted ones by comparing the public key. This allowed an attacker to supply a certificate with modified domain parameters, which would be trusted.

Even when not directly using untrusted parameters, the adversarial setting makes sense when we account for the physical nature of cards and, thus, for fault injection attacks. These could be mounted to manipulate any trusted parameters [BMM00; TT19] that the applet will use (e.g., in (EC)DH).

4.2.2 Attacks overview

We focus on attacks theoretically applicable to all implementations accepting composite parameters, instead of those stemming from specific behaviour of any one implementation. We present four different attacks, based on the cryptosystem and the injected parameter. In all four cases, it is possible to efficiently recover the private key for suitable injected parameters. The details will be given in Section 4.5.

For a composite group order n in ECDSA/ECDH or DSA/DH, it is well known that the discrete logarithm problem (DLP) in the group can be decomposed into DLPs in its quotient groups of prime-power order, which are much easier [PH78]. Thus for sufficiently smooth injected group orders, the discrete logarithm can be computed.

A similar decomposition and DLP difficulty reduction occurs when injecting a composite in place of the prime defining the full multiplicative group in DH/DSA [DCE].

We use yet another decomposition when injecting a composite in place of the prime defining the finite field for ECDSA/ECDH. As far as we know, this is a new result.

4.3 Methodology for assessing primality tests

In this section, we describe the method we used to analyze primality testing in cards of the JavaCard platform. Throughout the remaining text, the term *pseudoprime* will always mean a composite number that passes the MR test with respect to several small bases (the first t primes in our case).

In [Alb+18], the library functions for testing primality are ready to be called directly, and the source code can be analyzed to see for what purpose and with what parameters they are invoked. In contrast, we cannot even be sure if such functions exist in the closed-source implementation of the JavaCard platform. Hence we need to guess where they could be likely present and invoked (e.g., during domain parameter validation or key generation) and what parts of the algorithm could behave problematically if a prime input was replaced with a composite one. Also, unlike in [Alb+18], we only have a very limited amount of pseudoprime bit lengths to choose from.

JavaCard specifies five main cryptographic algorithms involving prime numbers or domain parameters: RSA, DSA, ECDSA, DH and ECDH (though not all cards support all of them). We analyzed all the relevant functions from the JavaCard specification and found no way to invoke primality testing in the RSA API with user-provided inputs. Also, the primes used there constitute the private key, and a scenario with them being replaced with pseudoprimes does not trigger a primality test. As a result, only the methods of the (EC)DH and (EC)DSA algorithms are applicable. Additionally, we restricted the testing focus on the ECDSA and ECDH algorithms only, as none of the tested cards support DH and only one supports DSA. However, we still study the theoretical aspects of using DSA/DH parameters.

The practical analysis of primality testing consists of three steps:

1. Constructing pseudoprimes and other composites (Section 4.3.2 and Section 4.3.3).
2. Generating (EC)DSA and (EC)DH parameters with primes replaced with the numbers crafted in the previous step (Section 4.5).
3. Triggering the card's primality test with the modified parameters as input, e.g., key generation, signing, verification in case of (EC)DSA or key agreement in case of (EC)DH (the rest of this section.)

In the last step, for any operation we perform on the card, the card only returns a response (output or error value) and the duration of the computation, which is often insufficient to understand exactly what happened due to implementations being closed-source. By the behaviour of the card under test, we mean such a response to our calls of API functions. To gain more information, we could also observe the card's power consumption or EM emissions during computation, but we do not consider these here. We use three types of basic operations in sequence to observe the behaviour:

3a) *Parameter setting.* Individual (EC)DSA or (EC)DH parameters are set on a `Key` object as byte arrays, interpreted as unsigned integers.

3b) *Key generation.* After setting all parameters, a `Keypair` can be generated. Note that the JavaCard does not differentiate between an ECDSA and ECDH keypair. In our tests, we skip this operation if it fails and continue with a manually generated private key, to also test the scenario where a keypair to be used is imported to the card.

3c) *Signing and verification* or *Key agreement.* After a `Keypair` object is successfully generated, it can be used to initialise a `Signature` or a `KeyAgreement` object and perform the operation. We supplied random data for signing and performed the key agreement between two keypairs generated on the card if possible. If the key generation failed, we instead substituted the private key and performed key agreement between it and the generator point on the curve.

To perform these operations, we developed and released our tool `ECTester` [JS19], which accesses the public JavaCard API and is generic to all cards.

4.3.1 Domain parameters

In this section, we examine the requirements on domain parameters used in (EC)DSA and (EC)DH, specifically primality requirements and show what requirements need to be fulfilled while replacing a prime with a composite. Since the parameters and the corresponding implementation checks for the finite field case and for the elliptic curve case differ significantly, we study them separately.

The DSA/DH case.
In DSA/DH, there are three domain parameters [NIST13]:

- p is the prime defining the multiplicative group $\mathbb{Z}_p^*$ in which we compute;
- g is an element of $\mathbb{Z}_p^*$;
- q is the order of g in $\mathbb{Z}_p^*$.

Note that the above already implies $g^q \equiv 1 \pmod{p}$, $q \mid p-1$ and $g \neq 1$ (unless $q=1$) and we can expect that these conditions could be checked by the implementation.

The supported sizes include $\{(1024,160), (2048,224), (2048,256)\}$ bits for p,q respectively. Classically, q is required to be prime, as the running time of the Pohlig-Hellman algorithm [PH78] depends on the size of the largest factor of q. Also, the random nonce k, which is generated during signing, needs to be invertible mod q. Thus for testing, we could replace either p or q with a pseudoprime. However, this replacement is non-trivial, as the conditions above are quite easy to satisfy when computing p and g from q, but somewhat hard if given

p, as one needs to factor $p-1$ and hope it has a prime factor q of the correct size. We discuss this in Section 4.3.4.

In the DH protocol on the JavaCard platform, the domain parameters are the same as in DSA, but the q parameter is optional [Ora19]. This means that either no checks related to q are performed, or that p is assumed to be a safe-prime, i.e. $p = 2q + 1$. We do not consider the case when the safe prime condition is assumed in the remainder of this chapter and instead refer the reader to [GMP19]. Similarly, we do not consider the case where there are no checks related to q present, as it is straightforward to subvert the parameters in such a system (for example, q can be very small).

Note that we did not test actual DSA/DH parameter sets, as mentioned earlier in Section 4.3, due to lack of support in the tested cards.

The ECDSA/ECDH case. This case is a little more complicated. The JavaCard API supports curves in the short Weierstrass form either over prime fields $\mathbb{F}_p$ or binary fields $\mathbb{F}_{2^m}$. We do not work with the binary field case, as most cards at our disposal do not support it. The prime field case then requires the inputs p, a, b, G_x, G_y, q, h, where:

- p is the prime defining the field $\mathbb{F}_p$ over which we will work;
- a, b are the coefficients of the elliptic curve E in short Weierstrass form over $\mathbb{F}_p$;
- G_x, G_y are the affine coordinates of the generator $G \in E(\mathbb{F}_p)$;
- q is the order of G;
- $h = n/q$ is the cofactor, where $n = \#E(\mathbb{F}_p)$.

As for supported sizes, p should have either 160, 192, 224, 256, 384, 512 or 521 bits. Computing q or n is prohibitively expensive for the card, so it is reasonable to assume that only the condition $[q]G = \mathcal{O}$ will be checked, possibly together with the size of q (by Hasse's theorem, $n = qh$ should be roughly the same size as p). In ECDSA/ECDH, q should be prime for the same reasons as in DSA/DH. Thus for testing, we can replace either p or q by a pseudoprime (for q, this is discussed in [GMP19]). To do that, we need to either construct an elliptic curve with a prescribed number of points (we used our tool `ecgen` [Jan19] that supports the complex multiplication method [Bro06; BS07]) when q is replaced, or to construct an elliptic curve over $\mathbb{Z}_p$ (with composite p) and correctly compute its order.

For each card and each bit-size in $\{160, 192, 224, 256, 384, 512, 521\}$, we test the card's behaviour for ECDSA and ECDH with parameter sets described in Table 4.2. The rest of this section shows how we generated p and n, while Section 4.5 explains how we constructed the malicious parameters from them. The full parameters used for testing in this chapter are included in Appendix B.3.

4.3.2 Generating pseudoprimes

As we are considering only the MR primality test, we use a slightly tweaked version of Arnault's method with three pseudoprime factors, described in Appendix B.2. We construct numbers that are pseudoprime to t smallest primes taken as bases, assuming the resource constrained smartcard will choose its bases from a set of small primes. The only limitation is that the bit-size of the pseudoprime must be one of the supported ones, as discussed in Section 4.3. To achieve this, we must try many combinations of t, k_2, k_3 to arrive precisely at the supported bit-sizes, while also trying to maximise t (Table 4.1). For each bit-size, the pseudoprime generation process took at most a few minutes on an ordinary laptop (using the precomputed values of t, k_2 and k_3).

bit-size	t	k_2	k_3
160	11	73	101
192	13	61	101
224	14	197	257
256	16	233	101
384	23	137	157
512	30	137	157
521	30	137	157
1024	52	241	281

Table 4.1: Parameters to construct pseudoprimes by tweaked Arnault's method [Arn95a; Alb+18].

4.3.3 Generating special composites

To systematically compare the card behaviour, we also used random composites with controlled numbers of factors or varying levels of smoothness, to get finer granularity. In this way, we can detect if the primality test is present at all (though possibly faulty).

Composites with a given number of factors. To generate a composite number of a given bit-size with a given number of factors, we use a greedy approach. In each step, we generate a random prime number of size b/r, where b is the number of remaining bits, and r is the number of remaining factors to be generated. We chose 3-factor and 10-factor numbers for comparison, as more factors would already lead to too smooth numbers.

Composites with a given smoothness level. For the smooth case, we employ a similar greedy algorithm that randomly chooses prime factors up to the smoothness bound and retries until a number with the right bit-size is constructed.

4.3.4 Generating complete domain parameters

In this section, we explain how to generate complete parameters for ECDSA/ECDH and DSA/DH, based on the pseudoprime and other composite inputs generated in Section 4.3.2 and Section 4.3.3. We use these parameters to test ECDSA/ECDH on the cards.

The challenge in embedding composites into the domain parameters lies in the fact that the card might check many properties of the parameters, while the only thing we are currently interested in is the compositeness of some of them. Thus the parameters should be as close to the correct ones as possible. Section 4.3.1 lists the properties that the card might verify in the domain parameter validation algorithms [ANSI98; IEEE00]. For each scenario, we also described the corresponding attack.

ECDSA/ECDH: prime p, composite q. The approach, in this case, is almost the same as the one described in [GMP19]. We use the complex multiplication method [Bro06; BS07], realised by our tool `ecgen` [Jan19], which is able to construct a curve over a prime field in short Weierstrass form with a given number of points. We need to take into account that the structure of $E(\mathbb{F}_p)$ is either cyclic or a product of two cyclic groups. This poses an issue because the JavaCard platform limits the size of the cofactor to an unsigned short integer, so just 16 bits. In the curves generated by two points, often the cofactor does not fit into 16 bits, even if we pick a large subgroup. Thankfully, the cards do not perform validation of the cofactor, as it is an optional input, so we just pick the generator with the largest order and set the cofactor to 1. Given a composite q, generating a suitable 256-bit curve took just a few minutes on an ordinary laptop.

One of the forms of composite q we tried to generate was that of an appropriately sized primorial (i.e., the product of all the primes up to some bound). However, the complex multiplication method, as implemented in the `ecgen` tool, was unable to generate them, even after a significant time spent on the task (e.g., a week on a single curve). The method searches for the curves by enumerating values of their complex multiplication discriminant, starting from 1, until a suitable curve and prime field is found. This points to an absence of prime field curves with primorial order and a small complex multiplication discriminant, which is an interesting observation.

ECDSA/ECDH: composite p, arbitrary q. Here we assume for simplicity that p is square-free and has no small factors (up to some bound, we chose 50). We want to find a curve whose order has no small divisors; otherwise, the card might reject the curve for a wrong reason, as we have observed in practice.

For each prime factor p_i of p, we iterate over all possible curves over $\mathbb{Z}_{p_i}$ until we find one whose order is prime (this will minimise the number of prime factors of the resulting curve over $\mathbb{Z}_p$). We also prefer if the order of the curve is never repeated for different p_i's, but this is easily satisfied in practice. When such a curve is found for each p_i, we create the desired curve modulo p just by using the CRT on the Weierstrass coefficients a, b of the individual curves (see Section 2.1. Since p has no small prime divisors, we can expect the same to be true for the order of the final curve as well, thanks to the construction, as the resulting order is the product of the individual orders.

To obtain a generator point of the resulting curve, we simply pick a generator point of each curve, and we use the CRT again on their coordinates. Since each curve over $\mathbb{Z}_{p_i}$ was cyclic and their orders were distinct, the final curve is cyclic as well, so we can set the cofactor to be 1. This whole process takes just seconds for the 3- and 10-factor 256-bit composites used in this chapter.

DSA/DH: prime p, composite q. This is the easiest scenario, as it almost completely follows the way ordinary DSA parameters are generated. We first pick a composite or pseudoprime q, then choose random properly sized integers k until $p = kq + 1$ becomes a prime. Then we repeatedly pick a random $r \in \mathbb{Z}_p^*$ until we get a generator of $\mathbb{Z}_p^*$ and compute $g = r^{(p-1)/q}$. In this way, we ensure that g has order q modulo p. This generation process is very fast, and takes just seconds to generate 1024-bit parameters.

DSA/DH: composite p, prime q. This case is more problematic to construct than the above one. First, let us assume that p is a Carmichael number (as is the case for the pseudoprimes we are constructing (Appendix B.2). We assume that either of the conditions

$$q \mid p-1, \quad g^q \equiv 1 \pmod{p}, \quad \text{and } g \neq 1$$

could be checked, so we will want to satisfy all of them.

These conditions imply that $g^q \equiv 1 \pmod{p_i}$ for all prime factors p_i of p, hence $g^{\gcd(q,p_i-1)} \equiv 1 \pmod{p_i}$. As q is a prime and $g \neq 1 \pmod{p_i}$ for some i (otherwise $g = 1$), this implies $q \mid p_i - 1$ for some i.

Thus we need $p-1$ to have a prime factor q of a size corresponding to the size of p (e.g., if p has 1024 bits, then we need q to have 160 bits). Given a specially constructed p, this means factoring $p-1$ and hoping for a factor of the correct size. This is exactly what we did for generating the DSA parameters, even though it was only practical for the 1024-bit parameters, given that factoring larger than 1024-bit random integers and hoping for a factor of a correct bit-size is computationally hard for our computation cluster. Finding an appropriate 1024-bit pseudoprime p such that $p-1$ has a 160-bit factor took a few days on an equivalent of an ordinary laptop.

Once we have p and q, we can again loop through random r from $\mathbb{Z}_p^*$ and compute g as $g = r^{(p-1)/q}$ until $g \neq 1$. This will imply that $g \not\equiv 1 \pmod{p_i}$ for at least one i, so that the primality of q together with the congruence $g^q \equiv r^{p-1} \equiv 1 \pmod{p}$ (as p is a Carmichael number) will imply that the order of g modulo p_i is q, hence $q \mid p_i - 1$.

Note that it is possible that no such g exists, even if p is a pseudoprime - for example for the Carmichael number $p = 7 \cdot 19 \cdot 67$ and $q = 5$, we have that $q \mid p-1$, but $q \nmid p_i - 1$ for any i, so there is no element of order q modulo p. However, it can be empirically seen that is unlikely to happen when p and q are large enough.

It seems hard to adapt this strategy of generating parameters for a fixed composite non-Carmichael p (which instead has a given number of factors or is smooth). One would have to simultaneously force $q \mid p-1$ and $q \mid p_i - 1$ for some prime factor p_i of p, which is equivalent to $q \mid \gcd(p-1, p_i-1)$. But unlike in the Carmichael case (where $\gcd(p-1, p_i-1) = p_i - 1$), heuristics show that we cannot expect $\gcd(p-1, p_i-1)$ to have a large prime factor for most composite p, let alone a factor of an exactly given size. Thus we do not consider this case further, but we stress that its significance is mostly limited to testing of black-box devices. A motivated attacker would use pseudoprime (or just Carmichael) p, as it has a much better chance to bypass potential primality tests, while making the generation of the other parameters easier.

4.4 Practical results

The analysis was performed on cards with ECC support that we were able to obtain in small quantities and covers most major vendors (except for Gemalto and Idemia). The cards were fabricated in the period between 2012 and 2018. Note that due to lengthily and costly certification processes, the pace of software changes in the smartcard environment is significantly slower than for standard software development. As a result, the products by the same vendor tend to reuse the same existing codebase (as visible from results for the NXP cards), and our findings are likely valid for the newer product versions as well. The results are summarized in Tables 4.2a and 4.2b.

The main result of our testing is that most manufacturers, apart from *Athena* and *Infineon*, seem to lack primality tests of the p and n parameters for ECDSA and ECDH. This follows from the same observed card behaviour for the tests with pseudoprime parameters (Section 4.3.2) as for the tests with general composite parameters (Section 4.3.3). Missing primality testing invites Pohlig-Hellman style attacks mentioned in Section 4.2. Due to the non-deterministic nature of smartcard computations, we had to run the tests many times to get representative results. The different bit-sizes of the curves used, ranging from 160 bits to 521, do not impact the results in an unexpected way.

Card		*p*		*q*				
	prime	pseudo	3f	pseudo	3f	10f	11s odd	11s even
Athena IDProtect	OK	IL	IL	IL	IL	IL	CYC	EXC
G&D SmartCafe 6.0	OK	OK	OK	OK	OK	OK	CYC	EXC
G&D SmartCafe 7.0	OK	OK/MUT	OK/MUT	OK	OK	OK	MUT	EXC
Infineon CJTOP 80k	OK	IL	IL	IL/OK	IL	IL	EXC	EXC
NXP JCOP v2.4.1	OK	OK/VRF	OK/VRF	OK	OK	OK	IL	IL
NXP JCOP CJ2A081	OK	OK	OK	OK	OK	OK	IL	IL
NXP JCOP v2.4.2 J2E145G	OK	OK/VRF	OK/VRF	OK	OK	OK	IL	IL
NXP JCOP J3H145	OK	OK/MUT	OK/VRF/MUT	OK	OK	OK	EXC	EXC
TaiSYS SIMoME VAULT	OK	OK/MUT	IL/MUT*	OK	OK	OK	EXC	EXC

(a) ECDSA results.

Card		*p*		*q*				
	prime	pseudo	3f	pseudo	3f	10f	11s odd	11s even
Athena IDProtect	OK	IL	IL	IL	IL	IL	CYC	EXC
G&D SmartCafe 6.0	OK	MUT	MUT	MUT	MUT	MUT	CYC	EXC
G&D SmartCafe 7.0	OK	OK	OK	OK	OK	OK	MUT	EXC
Infineon CJTOP 80k	OK	IL	IL	IL	IL	IL	EXC	EXC
NXP JCOP v2.4.1	OK	OK	OK	OK	OK	OK	IL	IL
NXP JCOP CJ2A081	OK	OK	OK	OK	OK	OK	IL	IL
NXP JCOP v2.4.2 J2E145G	OK	OK	OK	OK	OK	OK	IL	IL
NXP JCOP J3H145	OK	OK/MUT	OK/MUT	OK	OK	OK	EXC	EXC
TaiSYS SIMoME VAULT	OK	OK	OK	OK	OK	OK	EXC	EXC

(b) ECDH results.

Table 4.2: Results of domain parameters validation using on-card primality testing by nine different cards from five major manufacturers. Multiple values separated with a slash indicate that multiple results are present with decreasing occurrence from left to right. *IL (see below) happens on verification, key generation and signing works.

Result types

OK	Operation without error
IL	`ILLEGAL_VALUE` exception
VRF	Failed to verify signature
EXC	Unexpected exception
CYC	Card cycles indefinitely
MUT	Card does not respond

Parameter names

prime	standard parameters
pseudo *p*	pseudoprime *p*
3f *p*	3-factor composite *p*
pseudo *q*	pseudoprime *q*
3f *q*	3-factor composite *q*
10f *q*	10-factor composite *q*
11s odd *q*	11-smooth odd *q*
11s even *q*	11-smooth even *q*

- 🟩 Green background signifies tests with the expected result, i.e. the card correctly computed with the parameters or the card correctly rejected them.
- 🟨 Yellow background marks tests where the card exhibits unexpected behaviour, but are not vulnerabilities, and are not exploitable by the attacks from Section 4.2.
- 🟥 Red background marks tests where the card accepted parameters it should have rejected, and is thus vulnerable to attacks from Section 4.2.

We may have passed the primality test using a pseudoprime curve order in the case of the *Infineon CJTOP 80k* card, as the key generation and ECDSA signing and verification worked in a few rare cases, even though the card rejected the parameters most of the time. We observed this in roughly 3 out of 1000 tries on a 192-bit pseudoprime order curve. Our hypothesis is that the implementation is choosing small MR bases, which occasionally lie in the set of liars for our provided pseudoprime.

We were not able to pass the primality test present on the *Athena IDProtect* card, perhaps because it uses random MR bases or some other primality test.

We also observed that cards occasionally went mute and did not respond to the command, often upon invoking key generation. This behaviour is outside of the PC/SC specification[2] and results in a PC/SC error being raised by the reader's driver. It could also mean that the cards perform some kind of a self-test during the operation and stop responding as a security measure if the test fails. The presence of such self-tests is well documented in cards. In ECDSA, this error might stem from the card generating a nonce k that is non-invertible modulo q, which the system might not expect.

In the ECDSA case, several cards occasionally produced invalid signatures. This is possibly due to the modular inversion algorithm assuming a prime modulus. We did not investigate this matter further, but these invalid signatures might leak information about the private key or the used nonce, which might be abused by a lattice attack.

The behaviour of the cards also differs for smooth q and for 10-factor q. We think this is due to some unknown checks failing when such a smooth order is given, not due to a primality test. Furthermore, two cards (*Athena IDProtect*, *G&D SmartCafe 6.0*) cycle indefinitely on key generation on a curve with smooth odd order, we do not have any explanation for this behaviour.

Algorithms used during the operations, such as the modular multiplicative inverse or the modular square root, may be implemented to rely on the modulus being prime. Thus we were surprised to see the cards mostly working for composite p.

2. The PC/SC specification specifies the general communication protocol between the card and the reader device.

4.5 The attacks in detail

In this section, we discuss the attack details in each of the four scenarios we consider.

Attack on ECDSA/ECDH with prime p and composite q. Using the classical Pohlig-Hellman algorithm [PH78], the DLP asymptotically becomes only as hard as the DLP in a subgroup of order l, where l is the largest prime factor of the group order q. There it can be solved by the Pollard ρ algorithm, which costs roughly $\sqrt{\frac{\pi}{4}l} \approx 0.886\sqrt{l}$ point additions [BL]. Thus for example, when using a 256-bit curve and n has three factors of roughly the same size, the total computation cost of the DLP is approximately $3 \times 0.886 \times \sqrt{2^{86}} \approx 2^{44}$, which is already practical (and can be much cheaper for a larger number of factors). Compare this with a case of using the Pollard ρ algorithm to solve DLP on a standard 256-bit curve, where one gets the cost of $0.886 \times \sqrt{2^{256}} \approx 2^{128}$. An example of this attack is given in Appendix B.4.

Attack on ECDSA/ECDH with composite p and arbitrary q. When a composite p is a product of distinct primes $p_1, \ldots, p_e$ in ECDSA or ECDH, we have $E(\mathbb{Z}_p) \cong \bigoplus_{i=1}^{e} E(\mathbb{Z}_{p_i})$ (see Section 2.1), with the isomorphism essentially realised by the CRT applied to point coordinates. Thus the DLP on $E(\mathbb{Z}_p)$ again asymptotically becomes only as hard as the hardest DLP on some $E(\mathbb{Z}_{p_i})$ (since after solving the DLP in all individual groups, we can use the CRT to obtain the desired discrete logarithm). Since the order of $E(\mathbb{Z}_{p_i})$ is roughly p_i, the situation is very similar to the one for composite q in ECDSA/ECDH. An example of this attack is given in Appendix B.4.

Attack on DSA/DH with prime p and composite q. The Pohlig-Hellman algorithm is applicable in an exact analogy to the composite n case in ECDSA/ECDH. Note that the sub-exponential index calculus algorithm could also be used to solve the individual DLPs, but we expect it to perform worse than Pollard ρ (whose cost is asymptotically the same as for ECDSA/ECDH), as it cannot efficiently use the extra information about the factorisation of q.

Attack on DSA/DH with composite p and prime q. In this case, we know g^x modulo p, where $0 < x < q$ and $q \mid p_i - 1$ for some prime factor p_i of p (this follows from the construction described in Section 4.3.4).

Thus we also know the value g^x modulo p_i and finding x modulo p_i gives us x directly, since $x < q \leq p_i - 1$. Therefore it is sufficient to solve the DLP modulo p_i. Note that on one hand, Pollard ρ does not have an advantage compared with the case with a real prime p, as the group order is still q. On the other hand, the complexity of an index calculus algorithm only depends on p_i, which can be much lower than p. Hence the security level will be lower than it should be and might lead to a private key recovery for small enough p_i. We demonstrated the practicality of this approach in Appendix B.4.

4.6 Proposed defences

Without a robust primality test, a card cannot properly validate domain parameters. As the public JavaCard API lacks primality testing functionality, we cannot expect the developers to perform the validation either. Thus applications that allow the setting of custom domain parameters may result in a vulnerable applet.

Furthermore, the absence of primality testing functionality hinders the development of more complex cryptographic applications. For example, the vulnerability in the RSA key generation presented in the ROCA attack [Nem+17] could have been mitigated by applets generating the primes for their RSA keypairs themselves, thus avoiding full firmware fixes of the affected devices (which are often impossible in the case of cards). The lack of solid number-theoretic functionality in the JavaCard API prevented this though.

Fortunately, most of the protocols and implementations use standard named curves such as NIST P-256 or Curve25519. This seems to limit the current real-world impact of the aforementioned absence of primality testing in domain parameter validation.

We analyzed an extensive list of open-source implementations of JavaCard applets [Eni19] and found none that would use unauthenticated domain parameters in (EC)DSA or (EC)DH. Most used a fixed standard curve, with a few using domain parameters supplied in a command, but those were either authenticated or it was apparent from the context that they were provided by a trusted party, for example during the setup of the applet. However, one should keep in mind the possibilities of an untrusted setup described in Section 4.2, as well as the possibility of fault injection attacks. We also note that open-source JavaCard development comprises only a very small part of deployed JavaCards and that most applets are closed-source.

The recent trends in cryptography head towards misuse-resistance, the property of protocols and APIs that makes it hard for the developers to use and implement them incorrectly. Protocols and cryptosystems should allow simple and secure implementations. Examples of this include the nonce-misuse resistant authenticated encryption modes such as the SIV [Har08] or libraries with a very simple API such as libsodium or NaCl [BLS12]. With this direction in mind, the missing domain parameter validation steers the developers to misuse the API and undermine the security of their applets.

We thus propose several changes to the JavaCard specification:

- Require full domain parameter validation, for example as specified in ANSI X9.62 [ANSI98] and IEEE P1363 [IEEE00], which includes primality tests of prime parameters.
- Add an API that supports a set of named curves; allow manufacturers to only support this API. Consider perhaps deprecating or discouraging explicit domain parameter setting.
- Add a primality test to the public API.

Validating elliptic curve domain parameters consists of more than primality testing and general sanity checks on the parameters. Luckily, the necessary checks are specified in the aforementioned standards.

The modification of JavaCard API to accept only named curves instead of the full specification of curve parameters limits flexibility for the future inclusion of new curves as it might not be possible to update the list after card deployment. On the other hand, strict usage of only named curves prevents attacks similar to the recent attack on the Microsoft CryptoAPI library (CVE-2020-0601) [Age20], which cannot be prevented only by domain parameter validation.

The Miller-Rabin with random bases or Baillie-PSW primality tests should allow a robust and reasonably efficient (even on limited smartcard chips) implementation of primality testing. For an example of a performant and misuse-resistant primality test, see Massimo and Paterson [MP20].

4.7 Conclusions

We have explored the robustness of primality testing in domain parameter validation by smartcards of the JavaCard platform. Due to unavailability of primality testing functionality in the public JavaCard API, we tried to trigger the tests indirectly by using specially crafted composite domain parameters for ECDSA and ECDH operations.

We analyzed nine different smartcards from five major manufacturers and found that all but one failed to properly verify the primality of the provided ECDSA and ECDH domain parameters, not even requiring pseudoprimes to fool them, just composites. This results in a vulnerability to Pohlig-Hellman [PH78] style attacks, allowing the extraction of the private key. Our approach is generic to all black-box devices performing ECDSA and ECDH and the tooling can be reused.

Furthermore, the vulnerability is not easily mitigated for already deployed smartcards. The code responsible for the domain parameter validation is often stored in a read-only memory, which cannot be updated. In addition, the on-card verification of the provided domain parameters by the developer cannot be efficiently performed due to a lack of a primality testing functionality in the public JavaCard API.

5 Minerva: The curse of ECDSA nonces[1]

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a very popular digital signature algorithm, used among others in the TLS protocol [Res18], document signing and in blockchain applications. One of its main pitfalls is that it requires a unique nonce per signed message. Its fragility with regards to these nonces is well known. Any reuse of the nonce for a different message trivially leads to key recovery, as was the case in the PlayStation 3 game console [FAIL10], which utilized a fixed value for signing its binaries.

While nonce reuse problems are to a large extent mitigated by using deterministic generation of nonces as specified in RFC 6979 [Por13] or used in EdDSA [Ber+12], other potential issues can prove fatal. Knowing the nonce used for a single known message – or being able to brute-force it, as was the case for the Chromebook H1 chip [Chro19] used for U2F authentication – leads to private key recovery as well. Even worse, ECDSA is one of the Schnorr-like [Sch89] signature schemes, featuring a linear relation between the nonce and the private key. Thus even partial information about the random nonces (such as knowledge of certain bits) can be sufficient to recover the private key, often via the use of lattice reduction techniques. The partial information can originate from implementation issues (such as a faulty random number generator or a biased sampling method), or it might be observed via side-channel attacks or even forced via fault injection.

The already large, yet still incomplete list of past ECDSA-related vulnerabilities indicates a systemic difficulty in the implementation of secure, leakage-resistant ECDSA code. We developed a new leakage-testing tool, analyzed a variety of cryptographic libraries and devices, and found several with timing leakage during the ECDSA signing. In contrast to the more commonly used proof-based approach intrinsic to lattices, we focused on extensive heuristic comparison of existing and newly proposed methods on the benchmark data as collected from several real-world vulnerabilities and having different noise profiles.

1. The results in this chapter were published in [Jan+20], winning the Best Paper Award at Cryptographic Hardware and Embedded Systems (CHES) 2020. See `https://minerva.crocs.fi.muni.cz/` for additional materials.

This work brings the following contributions in the area of lattice-based attack methods against real implementations of ECDSA:

- Discovery and responsible disclosure of a set of vulnerabilities in the implementation of ECDSA in a certified cryptographic chip and five cryptographic libraries that allow to extract the private key used (Section 5.2 and Section 5.3).
- Propositions of two new distinct methods for private key recovery, which can tolerate a certain number of errors in the estimation of nonce's known bits (Section 5.4).
- A systematic and extensive comparison of the newly proposed and all previously published methods and their parameterization on the benchmark created using noise profiles observed in the real-world vulnerable implementations (Section 5.5). Notably, we show that newly proposed methods require an order of magnitude fewer signatures for a successful attack on the TPM-FAIL dataset [Mog+20].
- Release of an open-source tool[2] for extensive analysis of ECC implementations for software cryptographic libraries and JavaCard smartcards.

5.1 Related work

The task of recovering the ECDSA private key given enough information about signature nonces can be seen as an instance of the Hidden number problem (HNP). The HNP was originally introduced by Boneh and Venkatesan [BV96] to prove the hardness of computing Diffie-Hellman secrets. They showed a way to solve it by transforming it into a Closest Vector Problem (CVP) instance, solvable via lattice reduction and Babai's nearest plane algorithm [Bab86]. Howgrave-Graham and Smart [HS01] and Nguyen and Shparlinski [NS02; NS03] used the HNP to show that the (EC)DSA schemes are insecure if the attacker can obtain information about the most significant bits of the nonces used. Since then, the HNP was used in many attacks against biased or leaked nonces in (EC)DSA, often utilizing side channels such

2. ECTester: `https://github.com/crocs-muni/ECTester`.

as timing [Ald+19; BT11; Mog+20], cache attacks [BH09; Ben+14; PSY15; FWC16; Rya19a; Dal+18; Rya19b; Wei+20], electromagnetic measurements and other side channels [Gar+20; Gen+16; Bel+16].

Other attacks utilizing the HNP include using information about nonce distribution [BH19] or fault injection in the case of the SM2 signature algorithm [LCL13]. There have also been some theoretical extensions [FGR12; GRV16; HR06]. Finally, a very different approach was taken by Mulder et al. [Mul+13], where lattice reduction is used just as a processing step, and the core of the method lies in the Fast Fourier Transform. The method, also known as Bleichenbacher's approach, allows more input errors in exchange for much more signatures than the lattice-based approach.

The real device leakage and measurement noise profiles largely influence the input information and can cause errors, to which lattice-based methods are susceptible. Furthermore, the parameter space for these methods is extensive. Thus so far, there is only a limited systematic comparison of the efficiency of existing methods, evaluated based on the number of required input signatures (the fewer, the better). We addressed the situation and created such a comparison based on real-world noise profiles.

5.2 The vulnerability

The vulnerable devices and libraries (see Table 5.1 for a list) leak the effective bit-length of the scalar used in scalar multiplication on an elliptic curve through timing. The leakage is insignificant for many applications of scalar multiplication such as ECDH or key generation as only the bit-length of the private key is leaked, which represents a small amount of information about the private key, which is always the same.

However, in ECDSA [NIST13], the bit-length of the random nonce is leaked. Such leakage is much more significant as each signature represents new and usable information about the private key. When enough leaked information is accumulated, the private key can be completely recovered using lattice reduction techniques.

Table 5.1: Libraries and devices we analyzed with respect to their leakage.

Type	Name	Version/Model	Scalar multiplier	Leakage
Library	OpenSSL	`1.1.1d`	Montgomery ladder[1]	no
	BouncyCasle	`1.58`	Comb method[2]	no
	SunEC	`JDK 7 - JDK 12`	Window-NAF	no
			Lopez-Dahab ladder	yes
	WolfSSL	`4.0.0`	Sliding window	yes[3]
	BoringSSL	`974f4dddf`	Window method	no
	libtomcrypt	`v1.18.2`	Sliding window	no
	libgcrypt	`1.8.4`	Double-and-add	yes
	Botan	`2.11.0`	Window method[4]	no
	Microsoft CNG	`10.0.17134.0`	Window method	no
	mbedTLS	`2.16.0`	Comb method	no
	MatrixSSL	`4.2.1`	Sliding window	yes
	Intel PP Crypto	`2020`	Window-NAF	no
	Crypto++	`8.2`	unknown	yes
Card	Athena IDProtect	`010b.0352.0005`	unknown	yes
	NXP JCOP3	`J2A081, J2D081, J3H145`	unknown	no
	Infineon JTOP	`52GLA080AL, SLE78`	unknown	no
	G+D SmartCafe	`v6, v7`	unknown	no

[1] Applies the fixed bit-length mitigation.
[2] Uses many scalar multiplication algorithms.
[3] Likely not exploitable, due to a small amount of leakage.
[4] Uses additive scalar blinding.

5.2.1 Elliptic Curve Digital Signature Algorithm (ECDSA)

The public domain parameters of the ECDSA consist of a standardized elliptic curve E over a finite field and a generator point G on E of prime order n. To sign a hashed message $H(m)$ with a private key $x \in \mathbb{Z}_n^*$, the signer:

1. generates a uniformly random value ("nonce") $k \in \mathbb{Z}_n^*$,
2. computes $r = ([k]G)_x \pmod{n}$, the x-coordinate of $[k]G$ [3],
3. computes $s = k^{-1}(H(m) + xr) \pmod{n}$ [3],
4. outputs (r,s) as the signature of $H(m)$.

5.2.2 Leakage

For three out of five affected libraries in Table 5.1 (libgcrypt, MatrixSSL, SunEC) and the affected smartcard (Athena), the signing runtime directly depends on the bit-length of the nonce linearly: each additional

3. And returns to step 1. if the value equals 0.

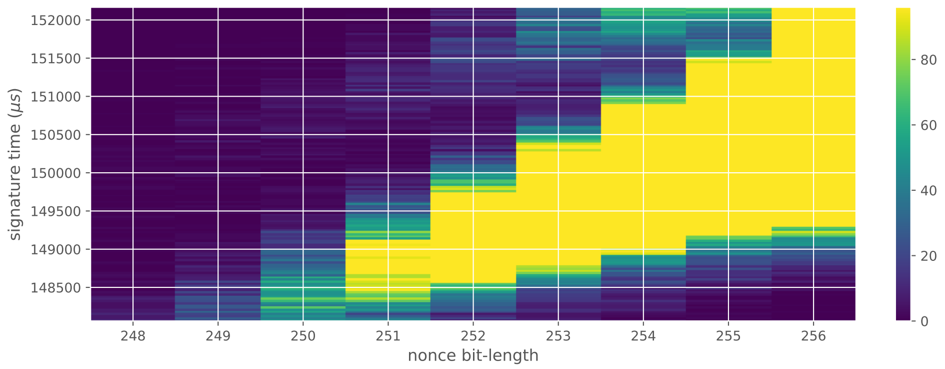

Figure 5.1: Heatmap of the signing duration and the bit-length of ECDSA nonces for 500 000 signatures using `secp256r1` curve on the Athena IDProtect card.

bit represents one more iteration of a loop in scalar multiplication, which increases the runtime. The leakage can be seen in the heatmap in Figure 5.1 and more clearly in powertraces of the card performing ECDSA signing in Figure 5.2.

For implementations leaking just the noisy bit-length (libgcrypt, SunEC, Athena) the leakage can be modeled using three parameters: duration of constant time processing in signing (e.g., hashing) ($base$), duration of one iteration of the scalar multiplication loop ($iter\_time$) and the standard deviation of the noise ($sdev$). On the `secp256r1` curve, this leakage **L** can be modelled as a random variable:

$$\begin{aligned} \mathbf{L} &= base + iter\_time \cdot \mathbf{B} + \mathbf{N} \\ \mathbf{B} &\sim \mathbf{Geom}(p = 1/2, (256, 255, \ldots, 0)) \\ \mathbf{N} &\sim \mathbf{Norm}(0, sdev^2) \end{aligned} \tag{5.1}$$

where **B** represents the bit-length with a truncated geometric distribution and **N** the noise. Only two of the above parameters, $iter\_time$ and $sdev$, affect how much the implementation leaks; we will use them to assess how easy it is to mount an attack.

In the case of wolfSSL and its sliding window scalar multiplication implementation, the dependency is more complex, and the leakage much smaller. Thus we consider it to be leaking, yet not likely exploitable. The MatrixSSL implementation also leaks the Hamming weight of a scalar: each non-zero bit increases the computation runtime. The Crypto++ library leaks the bit-length and other unidentified information.

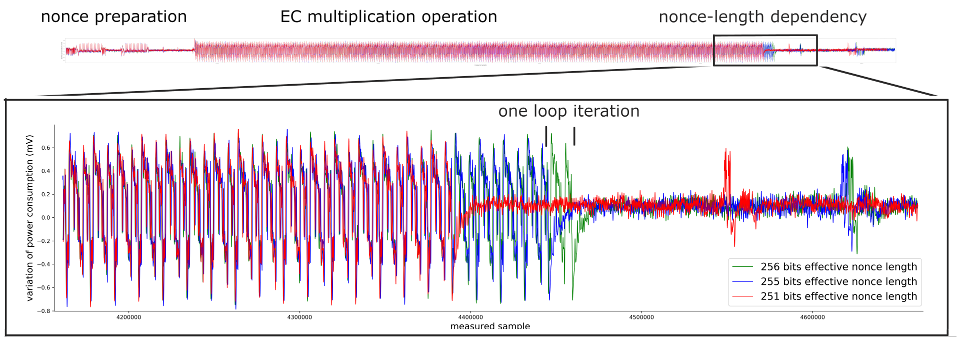

Figure 5.2: The visible leakage of nonce's bit-length on a power consumption trace of the Athena IDProtect smartcard as captured by an ordinary oscilloscope at 40 MHz sampling frequency for three signatures (aligned) with nonces having 0, 1 and 5 leading zero bits. The zoomed region of the whole ECDSA operation displays the difference at the end of the multiplication operation. The pattern corresponding to a single iteration of the scalar multiplication algorithm is clearly discernible, allowing an attacker with physical access to a card to establish the bit-length of the nonces precisely and with no error. Note that captured powertrace is presented only to highlight the root cause of vulnerability – the oscilloscope is not required for the successful attack.

5.2.3 Causes

We identified two main categories of root causes for the reported group of vulnerabilities. While the first one stems from an intricate relation between the ECDSA nonce, the private key, and the resulting signature, the second one relates to the general difficulty of implementing non-leaking scalar multiplication.

Nonce issues. The susceptibility of (EC)DSA nonces to lattice attacks does not seem to be widely known amongst developers of cryptographic software. There are four main issues regarding nonce use in (EC)DSA: nonce reuse, the bias in nonce randomness, nonce bit-length leaks, and other leaks of partial information about nonces. Due to the aforementioned lattice attacks and their variants, all of these issues might lead to a private key recovery attack.

Deterministic generation of nonces, as done in EdDSA[3] [Ber+12] or RFC6979 [Por13] mitigates the issues of nonce reuse and nonce bias. However, it does not address the latter two in any significant way. Deterministic generation of nonces might help the attacker in case the attacker has a noisy side channel leaking information about the nonce. If the attacker can observe the signing of the same message multiple times, they might use the fact that the same nonce was used to reduce the noise in the side channel significantly.

Leaky scalar multiplication. Not leaking the bit-length of the scalar used in scalar multiplication is surprisingly hard. Take almost any algorithm that processes the scalar in a left to right fashion – e.g., the Montgomery ladder [Mon87] – and instantiate it with incomplete addition formulas (that cannot correctly compute $\mathcal{O} + Q$ or $2\mathcal{O}$ in a side-channel indistinguishable way from $P + Q$ and $[2]P$). The result introduces a side channel leaking the bit-length. At the start of the ladder that computes a multiple of a point G, the two ladder variables are initialised either as $R_0 = \mathcal{O}$, $R_1 = G$ or as $R_0 = G$, $R_1 = [2]G$, depending on the used algorithm.

In the first case, the computation might start at a position with an unset bit in the scalar, i.e., the loop bound can be fixed in advance, as Algorithm 2 shows. However, until the first set bit is encountered, all of the additions and doublings will involve the point at infinity, and because of our assumption that the used formulas are incomplete, they will leak this information through some side channel. The leak might have the form of short-circuiting addition formulas, which check whether the point at infinity was input and short circuit accordingly to satisfy $\mathcal{O} + P = P$ and $2\mathcal{O} = \mathcal{O}$. Such was the case for vulnerable versions of *libgcrypt* and was the reason why merely fixing a loop bound in scalar multiplication was not enough to fix the issue. The formulas might leak the fact that the point at infinity is present through different channels than timing: power or electromagnetic side channels come to mind, as the point at infinity is often represented using only 0

3. We remark that EdDSA is not vulnerable to our attack, as it uses a mitigation described in Section 5.2.4.

or 1 values, which can often be distinguishable in multiplication and addition on a powertrace.

Algorithm 2: Montgomery ladder (complete)

```
Input  : $G$ (the base point);
         $k = (k_l, \dots, k_0)_2$ (the scalar in binary representation)
Output : $[k]G$
$R_0 \leftarrow \mathcal{O}$
$R_1 \leftarrow G$
forall $i \in [l, \dots, 0]$ do
    $R_{\neg k_i} \leftarrow R_0 + R_1$
    $R_{k_i} \leftarrow [2]R_{k_i}$
return $R_0$
```

In the second case, the most significant bit of the scalar must be explicitly found [BT11], as Algorithm 3 shows. The ladder must start at that bit, as the variables are initialized into a state such that the point at infinity will not appear (so that incomplete formulas can be used).

Such an implementation clearly leaks the bit-length through timing alone, because of the loop bound on the bit-length of the scalar.

Algorithm 3: Montgomery ladder (incomplete)

```
Input  : $G$ (the base point);
         $k = (k_l, \dots, k_0)_2$ (the scalar in binary representation)
Output : $[k]G$
$R_0 \leftarrow G$
$R_1 \leftarrow [2]G$
forall $i \in [|k| - 1, \dots, 0]$ do
    $R_{\neg k_i} \leftarrow R_0 + R_1$
    $R_{k_i} \leftarrow [2]R_{k_i}$
return $R_0$
```

The use of incomplete formulas or the direct computation of the bit-length of the scalar for use as a loop bound in scalar multiplication was the source of leakage in all of the vulnerable software cryptographic libraries. As the implementation on the vulnerable card is closed source, we were not able to analyze it directly; however, a similar cause is likely, given the nature of the leakage on the powertraces in Figure 5.2.

Both the vulnerable Athena IDProtect card and the used Atmel AT90SC chip with its cryptographic library, respectively, are FIPS 140-2 [Ath12a] and Common Criteria (CC) [Ath12b; Atm09] certified devices, respectively. The presence of such a vulnerability in certified devices can be explained by noting that the CC security target of the AT90SC chip [Atm09] contains mention of two versions of ECC functionality, fast and secure variants. It further goes to mention that the fast variants are not protected against side-channel attacks and should not be used on secure data. We hypothesize that these fast and insecure functions were erroneously used by the Athena IDProtect card and resulted in the vulnerability.

5.2.4 Mitigations

Below, we discuss several mitigations that can be applied to a leaking implementation to fix the vulnerability. The mitigations either stop the leak or mask it with additional randomness to prevent reconstruction of the private key.

Complete formulas. As described in Section 5.2.3, one of the root causes of the vulnerability was the usage of incomplete addition formulas, which introduced a measurable time difference between the case where one of the points being added is the point at infinity and the case where both points are affine. This ultimately led to a leak of the nonce length. The use of complete formulas [RCB16; SM16], which behave in the same way for all possible inputs, would prevent such leak. Besides being constant-time, these formulas also make the addition algorithm less complex. However, these formulas are slower than incomplete formulas, by a factor of around 1.4 as reported by Renes, Costello, and Batina [RCB16], and until late, they were not even available for all short Weierstrass curves.

Using complete formulas is likely the most systematic mitigation, so we recommend to the developers of the affected systems to switch to complete formulas whenever possible. The mitigation also requires that the scalar multiplication loop starts at a fixed bit and that the bit-length of the scalar is not explicitly computed to shorten the loop.

Fixing the bit-length. Even if incomplete addition formulas are used, the vulnerability can still be removed by making all the nonces the same bit-length. One option is to add a suitable multiple of the group order to the nonce so that the result of multiplying a point by the nonce will not be affected. For example, following the recommendation by Brumley and Tuveri [BT11], we could compute the modified nonce as

$$\hat{k} = \begin{cases} k + 2n & \text{if } \lceil \log_2(k+n) \rceil = \lceil \log_2 n \rceil \\ k + n & \text{otherwise.} \end{cases}$$

This fixes the bit-length of $\hat{k}$, ensuring that $\lceil \log_2 \hat{k} \rceil = \lceil \log_2 n \rceil + 1$. Subsequently, the scalar multiplication loop will always run the same number of times (as both points being added are already affine during the first addition), and the nonce length will not be leaked through timing. As the computational cost of using $\hat{k}$ over k is negligible, we recommend the affected systems to do so as a second line of defense.

Scalar randomisation. Alternatively, there are various side-channel countermeasures utilizing scalar randomization (i.e., performing the scalar multiplication without directly multiplying by the scalar). A basic overview of these methods can be found in Danger et al. [Dan+13a]. While introducing some overhead, they also effectively protect against some power analysis attacks.

Long nonces. Finally, multiplication by much longer nonces (without prior reduction modulo the group order n) should thwart these types of attacks, as learning the most significant bits of the scalar seems to provide almost no information about the most significant bits of the scalar modulo n. Such an approach is taken by EdDSA [Ber+12]: for example, when using Ed25519, the nonce is deterministically created as a 512-bit hash, whereas n has only 255 bits. In principle, the countermeasure could be adapted for ECDSA as well.

5.2.5 Responsible disclosure

We disclosed the vulnerabilities to the affected vendors upon discovery; we also provided assistance and patches fixing the vulnerability to several of them. Currently, all of the vulnerabilities in the software products are fixed in their newer versions. The state of the vulnerable chip AT90SC is unknown, as it is currently offered by the WiseKey company [Wis], which did not confirm or deny our findings regarding the chip. The Athena IDProtect card is no longer in production, as Athena was acquired by NXP Semiconductors, which confirmed to us that no new products are based on the vulnerable code. However, existing cards remain vulnerable as updates to JavaCards are often almost impossible to deploy.

We then disclosed the vulnerability publicly, together with a proof of concept and a testing tool that can be used to verify that other implementations are not vulnerable.

5.3 The attack

In the case of (EC)DSA and even attestation systems such as EPID [Dal+18] or ECDAA [FIDO18; Mog+20], the knowledge of the most significant bits of nonces (with the goal of computing the private key) can be turned into a HNP instance [BV96], which can be turned into an instance of the Closest Vector Problem and solved using the methods of lattice reduction. We will introduce the HNP and show how the knowledge of the most significant bits of nonces translates into a HNP instance for the aforementioned systems.

Notation. We use $\lfloor y \rfloor_q$ to denote the reduction of $y \in \mathbb{Z}$ modulo q (so the result lies in $[0, q-1]$) and $|y|_q := \min_{a \in \mathbb{Z}} |y - aq|$ to denote the distance of $y \in \mathbb{R}$ to the closest integer multiple of q.

It is easy to see the following (in)equalities:

$$\begin{aligned} |y|_q &= |y| && \text{for all } y \in [-q/2, q/2], \\ |y|_q &= |y - aq|_q && \text{for all } a \in \mathbb{Z}, \\ |y - z/2| &< z/2 && \text{for all } z \in \mathbb{R}, 0 < y < z \end{aligned} \tag{5.2}$$

Definition 5.3.1 (Approximations)**.** By $\mathrm{APP}_{l,q}(y)$, we will denote any $u \in \mathbb{Q}$ satisfying

$$|y - u|_q \leq q/2^l.$$

The most significant modular bits [BV96; NS03] are defined in the following way:

Definition 5.3.2 (Most significant modular bits)**.** The $l > 0$ most significant modular bits of an element $y \in \mathbb{Z}_q$ (regarded as an integer in $[0, q-1]$) are the unique integer $\mathrm{MSMB}_{l,q}(y)$ such that

$$0 \leq y - \mathrm{MSMB}_{l,q}(y) \cdot q/2^l < q/2^l.$$

(Thus for $l = 1$, the most significant bit of y is 0 or 1 depending on whether $y < q/2$.)

As noted in [NS03], this definition is in contrast with the usual definition of the most significant bits of an integer. In the lattice context, the modular bits are more convenient to work with, but see Remark 1 in Section 5.3.1.

It is worth observing that the most significant modular bits give rise to a specific approximation of y: the inequality

$$|y - \mathrm{MSMB}_{l,q}(y) \cdot q/2^l|_q < q/2^l$$

is equivalent to the in Definition 5.3.2, since $y - \mathrm{MSMB}_{l,q}(y) \cdot q/2^l \in [-q/2, q/2]$.

Thus we can take$\mathrm{MSMB}_{l,q}(y) \cdot q/2^l$ as $\mathrm{APP}_{l,q}(y)$ and after recentering the bits, we can take $\mathrm{MSMB}_{l,q}(y) \cdot q/2^l - q/2^{l+1}$ as $\mathrm{APP}_{l+1,q}(y)$ giving the exact approximation used in Nguyen and Shparlinski [NS03].

The Hidden number problem. Following [NS03], we state the HNP in the following way:

Definition 5.3.3 (Hidden number problem)**.** Given approximations $a_i = \mathrm{APP}_{l_i,q}(k_i)$, where $k_i = \lfloor \alpha t_i - u_i \rfloor_q$, for many known t_i that are uniformly and independently distributed in $\mathbb{Z}_q^*$, known u_i and a fixed secret $\alpha \in \mathbb{Z}_q$, find α.

In the text, we will use the more common definition where the problem is, given many HNP inequalities $|k_i - a_i|_q < q/2^{l_i}$, find α.

For (EC)DSA, α is the fixed secret key, t_i and u_i are values constructed from the signatures such that $k_i = \lfloor \alpha t_i - u_i \rfloor_q$, and $a_i = \mathrm{APP}_{l,q}(k_i)$ is the information leaked. In our case, $a_i = 0$, as the l most-significant modular bits are zero[4]. This simplifies the above inequality to:

$$|\alpha t_i - u_i|_q < q/2^{l_i}. \tag{5.3}$$

5.3.1 Constructing the HNP

Assume that we are given d (EC)DSA signatures (r_i, s_i) of message hashes $H(m_i)$ such that for each respective nonce $k_i > 0$, we know that the most significant $l_i > 0$ bits of k_i are zero. Denoting the curve order by n and the private key by x, we have

$$\begin{aligned} \lfloor k_i \rfloor_n &< n/2^{l_i} \\ |s_i^{-1}(xr_i + H(m_i))|_n &< n/2^{l_i} \\ |x\lfloor s_i^{-1} r_i \rfloor_n + \lfloor s_i^{-1} H(m_i) \rfloor_n|_n &< n/2^{l_i} \end{aligned} \tag{5.4}$$

By setting $q = n, \alpha = x$, $t_i = \lfloor s_i^{-1} r_i \rfloor_n$, $u_i = \lfloor -s_i^{-1} H(m_i) \rfloor_n$, we have a HNP instance.

Remark 1. The fact that the most significant l_i bits (in the classical sense) of k_i are zero translates into the inequality

$$\lfloor k_i \rfloor_n < 2^{\lceil \log_2(n) \rceil}/2^{l_i},$$

which is weaker than the used $\lfloor k_i \rfloor_n < n/2^{l_i}$. This means that the inequalities above might not be true for all k_i. However, as the ratio $n/2^{\lceil \log_2(n) \rceil}$ is around 10^{-9} for the curve `secp256r1` that we are working with, we will use the more convenient stronger inequality which will still be true with overwhelming probability. It is worth having this distinction in mind though, as it would cause problems for example for the curve `sect571k1`, where the ratio $n/2^{\lceil \log_2(n) \rceil}$ is almost 2.

In our setup, the values l_i are estimated from timing only, so it may well happen that the actual number of most significant zero bits is

4. We do not use the knowledge of the most-significant set bit, as it allows us to tolerate one-sided errors.

either higher or lower than l_i. The former case is not a significant issue, as the attack will still work (although we are using less information than available, so we need to balance this by a higher number of signatures). However, the latter case is quite problematic, as we are utilizing false information in the attack, and the lattice approach will very likely fail to compute the correct result. We will refer to the second case as to "input errors" in the remainder of the text and discuss some options of dealing with them in Section 5.4. For now, just note that there are two basic ways of removing the errors: either lowering the l_i (which is a standard technique) or subtracting a certain multiple of $n/2^{l_i}$ from k_i, so that the first inequality in (5.4) will hold. The latter option amounts to adding a certain multiple of $n/2^{l_i}$ to u_i, which clears the high bits of k_i.

5.3.2 Solving the HNP

Given d HNP inequalities of the form

$$|\alpha t_i - u_i|_n < n/2^{l_i},$$

we can construct a lattice spanned by the rows of the $(d+1) \times (d+1)$ matrix B [NS02; Ben+14]:

$$B = \begin{pmatrix} 2^{l_1}n & 0 & 0 & \dots & 0 & 0 \\ 0 & 2^{l_2}n & 0 & \dots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & 2^{l_d}n & 0 \\ 2^{l_1}t_1 & 2^{l_2}t_2 & 2^{l_3}t_3 & \dots & 2^{l_d}t_d & 1 \end{pmatrix}$$

Then, by the HNP inequalities above, the vector

$$\mathbf{u} = (2^{l_1}u_1, \dots, 2^{l_d}u_d, 0)$$

is a vector unusually close to a lattice point. The closest lattice point often has a form

$$\mathbf{v} = (2^{l_1}t_1\alpha, \dots, 2^{l_d}t_d\alpha, \alpha),$$

in which case finding such lattice point reveals the private key α. To do so, one needs to solve the Closest Vector Problem (CVP). There are several algorithms for solving the CVP, the original paper [BV96] used Babai's nearest plane algorithm [Bab86] with LLL [LLL+82]. One could also use BKZ for lattice reduction or solve the CVP by enumeration. There is also a technique of transforming an instance of CVP to a Shortest Vector Problem (SVP) by embedding the target vector into a larger lattice:

$$C = \begin{pmatrix} B & 0 \\ \mathbf{u} & n \end{pmatrix}$$

Then, one can solve the SVP by lattice reduction and either looking directly at basis vectors or by further enumeration to find the shortest vector. As we used several heuristic arguments, the solution is not guaranteed to be found.

Generally, each inequality adds l_i bits of information and the problem starts to be solvable (theoretically) as soon as the lattice contains more information than the unknown information in the private key. The expected amount of information in N signatures can be computed as $N \cdot \sum_{i=2}^{\lceil \log_2(n) \rceil} 2^{-l_i-1} \cdot l_i \approx \frac{3}{4}N$ assuming only signatures with $l_i \geq 2$ are used. Adding inequalities with $l_i < 2$ generally does not help, as those will not lead to the desired vector being unusually close to a lattice point [Ara+14].

Using the above formula, we obtain the expected minimum of around $N = 342$ signatures for a 256-bit private key. Since the amount of information is linear in N it can be computed as $N \approx \frac{4}{3} \cdot |K|$ for size $|K|$ of the private key. Adding dimensions is also not for free, as the runtime of lattice algorithms grows significantly with an increase in the number of dimensions. However, adding some overhead of information, such that the lattice contains around 1.3 times the information of the private key, was shown to improve the success rate [Rya19b].

5.3.3 Baseline attack

The baseline version of the attack, against which we will compare the variants, is an application of the attack from Brumley and Tuveri [BT11]. We collect N signatures from the library or card while measuring the duration of signing precisely. Then, we sort the signatures by their duration and take d of the fastest. For the basic attack, we then assume these all have at least $l_i = 3$ most-significant bits of the nonce zero, we construct the HNP and transform it into a CVP and then an SVP matrix. Given no noise (perfect dependency of the signing duration on the bit-length), the assumption of $l_i = 3$ makes sense if the collected number of signatures N is larger than $d \cdot 2^{l_i} = 8d$, as then the d fastest signatures will likely have at least the required number of most-significant zero bits.

We use fplll [FPL16], an open-source implementation of the LLL and BKZ lattice reduction algorithms, to perform LLL and progressive BKZ [Aon+16] reduction of the SVP matrix with block sizes $\beta \in \{15,20,30,40,45,48,51,53,55\}$. After each reduction step, we check if the second to last column of the matrix contains the private key by multiplying the base-point by the column entries and testing equality to the public key.

5.4 Attack variants and new improvements

The main limitation of the attack described in Section 5.3.3 is the required number of signatures and its sensitivity to input errors (as the time measurements are noisy). This section offers a remedy: we present and discuss in detail the following methods, the last two of which are new to the best of our knowledge:

- Measurement improvements (*known*);
- Random subsets (*known*) [BT11];
- Recentering (*known*) [BH19; Mog+20];
- The CVP/SVP methods (*known*);
- Nonce differences (*partially discussed in* [FGR12] *and* [BH19]);
- **Geometric bounds** (*new in this work*);
- **CVP + changes in** u (*new in this work*).

In Section 5.5, we will systematically compare these to each other as well as to the baseline attack on both simulated and real data.

As the listed methods are mostly independent of each other[5], their combinations could possibly provide improvement in the decreased

5. There are two notable exceptions: CVP + changes in u requires the use of the CVP method and the recentering approach cannot be used in nonce differences.

number of necessary signatures. However, the high number of computational experiments for all possible combinations makes an exhaustive analysis prohibitively expensive.

Improving the input data. One straightforward way of improving a noise-sensitive method success rate is to lower the noise via better measurements. In our context, this means either mounting a microarchitectural side-channel attack on a vulnerable library or performing power/electromagnetic side-channel attack on a vulnerable card.

In the former case, the attacker would use, for example, a cache side channel as done by Ryan [Rya19b], to count the number of calls to the point addition and doubling functions. This would give him a much more precise measurement of bit-length than simply timing the full ECDSA signing operation, which contains hashing, data processing, and other library functions.

In the latter case, the attacker with physical access to a vulnerable card would obtain power or electromagnetic traces as those shown in Figure 5.2 and use pattern matching to count the number of addition and doubling operations in the trace. This will again give the attacker a much more precise measurement of bit-length.

We expect both of the above approaches to lead to noise-free information about the bit-length of the nonce and thus a much easier application of the HNP to obtain the private key, leading to a similar success rate as in Figure C.13.

Random subsets. The errors in the input can be mitigated by randomly selecting subsets of signatures from the set of all signatures with relatively short duration as proposed by Brumley and Tuveri [BT11]. If the number of errors is not too large, we can expect to find a subset without any error after a reasonable number of tries. Note that this strategy is also trivially parallelizable, unlike most of the other methods.

In practice, we can take a medium-sized subset of the fastest signatures (for example, $\frac{3}{2}d$, where d is the dimension of the matrix we are aiming for) and repeatedly run the attack with randomly selected d of these signatures. However, since the strategy does not always use the d fastest signatures, it does not use the information in an optimal way. On average less information is present in the randomly selected signatures, with the hope that the increased number of tries will lead to both enough information and few errors being present in at least one of them.

Recentering.

To increase the the amount of information in each HNP inequality, one can use the fact that the nonce is non-negative and that to gain a HNP inequality we only need to upper-bound the absolute value, as only it affects the norm of the vector. Instead of using the inequality (5.4), one can apply (5.2) to get:

$$|x\lfloor s_i^{-1}r_i\rfloor_n + \lfloor s_i^{-1}H(m_i)\rfloor_n - n/2^{l_i+1}|_n < n/2^{l_i+1}.$$

CVP/SVP approach. Even though the HNP naturally translates into a CVP instance, such an approach is usually believed to not be very suitable [Ben+14; BH19; PSY15]. Instead, the standard approach is to further convert the CVP instance into an SVP instance by including the rows of u_i's in the matrix and checking for a short vector after lattice reduction (as described in Section 5.3.2). However, for reasons that we explain later, we also evaluate the CVP approach (using Babai's nearest plane algorithm) and measure how it compares to the SVP approach, even though we expect it to perform worse.

Differences of nonces. Instead of using information about the nonces k_i themselves, we could instead use information about their differences $k_i - k_j$, hoping that the most significant bits of k_i and k_j might cancel out.

Assuming that $l_i \leq l_j$, we can make a similar computation as in (5.4). From the inequality:

$$||\lfloor k_i\rfloor_n - \lfloor k_j\rfloor_n|_n < n/2^{\min(l_i,l_j)} \tag{5.5}$$

we get:

$$||\lfloor s_i^{-1}(H(m_i) + r_i x)\rfloor_n - \lfloor s_j^{-1}(H(m_j) + r_j x)\rfloor_n|_n < n/2^{\min(l_i,l_j)}$$
$$|x(\lfloor s_i^{-1}r_i - s_j^{-1}r_j\rfloor_n) + \lfloor s_i^{-1}H(m_i) - s_j^{-1}H(m_j)\rfloor_n|_n < n/2^{\min(l_i,l_j)}.$$

Thus by setting $t_i = \lfloor s_i^{-1}r_i - s_j^{-1}r_j\rfloor_n$, $u_i = -\lfloor s_i^{-1}H(m_i) - s_j^{-1}H(m_j)\rfloor_n$ we again obtain a HNP instance.

When utilizing nonce differences in practice, we aim to create the pairs (i, j) in a way that $l_i = l_j$ whenever possible to minimize information loss due to the bound being the minimum. For the geometric bounds, it corresponds to taking differences of neighbouring signatures. Note that if the number of bits $l_i = l_j$ was chosen erroneously too high in both cases, the error is nullified in the difference.

We note that when using differences of nonces, recentering is not possible, as the differences might be negative. This fact is, to some extent, mitigated by the aforementioned effect of subtraction correcting some errors.

Bounds l_i. The bounds l_i state the minimal number of leading zero bits of nonces k_i ($k_i = |\alpha t_i - u_i|_n < n/2^{l_i}$), representing the amount of information we gain from the nonces. They play a crucial role in our attacks since the lattice-based methods used to solve the HNP are quite sensitive to errors in l_i when the amount of information in the lattice is close to the size of the private key.

We are facing the problem of how to assign the bounds l_i to a given set of unknown nonces k_i based only on the times of corresponding signatures. The goal is to assign maximal values l_i to k_i such that the number of errors (false inequalities) is small. The values l_i should reflect the distribution of $log_2(k_i)$ for a selected set of signatures (only a subset of signatures is used in the attack). The selection of the signatures is natural – only signatures with the shortest times are used in the attack, as corresponding to the shortest k_i and largest l_i.

The nonces k_i are generated uniformly at random; we can thus expect that the number of most significant zero bits follows a truncated geometric distribution when n is close to a power of two. Thus roughly for one-half of the nonces $l_i = 0$, for one-quarter of nonces $l_i = 1$, and so on. Assuming a one-to-one linear dependency between the bit-length of k_i and the duration of signing, we would obtain a clear method of assigning bounds to the signatures, sort them by duration and apply the above distribution. However, the real timing leakage is noisy and the distributions of duration for signatures with different bit-lengths overlap (see Figure 5.1 and Figure C.12).

In the experiments, when assigning bounds to the fastest d signatures out of N signatures collected, we used two strategies for the bound assignment:

- *constant bounds* – a fixed $c \in \{1,2,3,4\}$ is used for all bounds l_i;
- *geometric bounds* (new in this work) – bounds are calculated according to the above truncated geometric distribution based on N, the number of signatures collected. One half of signatures has $l_i = 0$, one quarter has $l_i = 1$, etc. Then simply the fastest d signatures are taken with their calculated bounds.

Figure 5.3 shows that on simulated data with no noise, the geometric bounds constitute the mean of the distribution of true leading zero bits, as the difference of the simulated data and the bounds has zero mean.

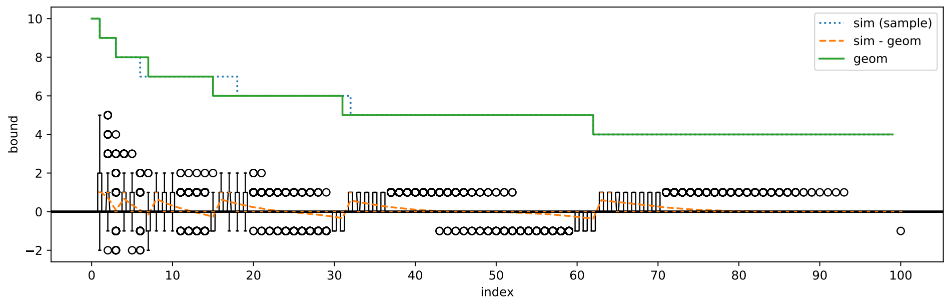

Figure 5.3: Plot of the geometric bounds (**geom**) with $N = 2000$ for $d = 100$ fastest signatures, along with a random sample of the true leading zero bits from simulated data (**sim**) and boxplots of the distribution of the difference of the simulated data and the bounds (**sim - geom**). Negative values imply an error; positive implies some available information is unused.

CVP + changes in u. When using the SVP approach, the u_i's are incorporated into the matrix, and it is not possible to change them without affecting the final result of lattice reduction. Thus any error in the u_i's propagates to the reduced lattice. In contrast, taking the CVP approach, the lattice can be reduced independently of the u_i's (which form the target). Such reduction is beneficial as we can reduce the lattice just once (and potentially with better quality) and subsequently try to solve the CVP with many possible choices of u_i's (making changes at the l_i-th positions where the errors are most likely to occur). Since the reduction is the most computationally expensive part, if we are solving CVP via Babai's nearest plane algorithm, we can efficiently try many different changes and fix the input errors.

For each e-tuple, we could try flipping one bit of u_i at the l_i-th position simultaneously, which should be feasible at least for $e \leq 3$. Since the runtime of the method should not depend on small changes in the u_i's, we expect that the total runtime for the strategy will be one reduction of the matrix and $\binom{d}{e}$ times the runtime of the CVP solving, for example via Babai's nearest plane algorithm, which is very fast. The advantage of the strategy is that it is trivially parallelizable.

5.5 Systematic comparison of attack variants

We compare existing methods for private key recovery with our newly proposed ones for the range of possible parameterizations. We run the methods while varying the number of signatures collected (N) and varying the dimension of the matrix (d) for four datasets with different noise profiles collected from the real-world leakages. For most experiments, the number of signatures N covered the range $[500, 7000]$ with a step size of 100 and three additional steps at $8000, 9000, 10000$. The dimension of the matrix and the number of signatures used in the matrix d ranged from 50 to 140 with a step size of 2. Such enumeration of the parameter space allowed us to evaluate the performance of the methods with a significantly wider range of parameters than presented in the introductory papers. Thus each variant of the attack goes through $3174 = 69 \cdot 46$ combinations of N and d with a typical runtime in seconds to minutes. Because of the size of the parameter space, we repeated each variant five times with random samples of N signatures from each of the four datasets described in Section 5.5. In total, the presented results took over 18 core-years to compute on our university cluster.

We evaluate the success rate of the attack improvements with respect to the number N of signatures available, which is frequently the biggest limitation for an attacker, influencing the attack practicality and runtime. Another interesting point of evaluation is the minimal number of signatures collected $(min(N))$, where a given attack variant succeeds for the first time. When not explicitly specified, we use the base parameters and method, as described in Section 5.3.3.

Dataset	*base* (μs)	*iter_time* (μs)	*sdev* (μs)
`sim`	0	1	0
`sw`	453.4	12.7	17.2
`tpm`	27047.3	236.1	211.3
`card`	43578.4	371.5	451.3

Table 5.2: Estimated Equation (5.1) parameters in our datasets.

Test data. The attack variants are evaluated using four separate data sets (denoted as `sim`, `sw`, `card`, `tpm`) with varying levels of noise. All the datasets used consist of at least 50 000 ECDSA signatures over the `secp256r1` curve from which we randomly sample N signatures for the evaluation of our attacks.

- The **sim dataset** contains simulated data for which there is an exact one-to-one correspondence between the signing duration and the bit-length of the random nonce with no systematic noise. However, these simulated signatures were still generated by uniformly randomly selecting the random nonce and computing the number of most-signifcant zero bits. A given sample is thus a result of a random process and varies naturally.

- The **sw dataset** contains data from a vulnerable version of the software cryptographic library *libgcrypt* collected from a simple C program on an ordinary Linux laptop.

- The **tpm dataset** contains data from the recent work of Moghimi et al. [Mog+20] collected from a vulnerable STMicroelectronics TPM (Trusted Platform Module). The data was collected via a custom Linux kernel module and contained a relatively small amount of noise.

- The **card dataset** contains data from the vulnerable Athena IDProtect smartcard, collected by a Python script running on an ordinary Linux laptop with a standard standalone smartcard reader connected. Such measurements are particularly noisy due to the complex software stack and hardware components between the script and a card.

Bounds l_i. Our first experiment compared the constant bounds with $c \in \{1,2,3,4\}$ to the geometric ones. As the heatmaps in Figures 5.4 and 5.5 and the plot in Figure 5.6 show, geometric bounds perform better than constant bounds for the baseline attack (see Section 5.3.3) on all datasets. They utilize the leaked available information much better, following the mean distribution of the actual leading zero bits.

Figure C.13 shows the limits of advances through changing the bounds assignment – the bounds were picked optimally here, using the prior knowledge of the actual nonces and the private key. Such a success rate can also be achieved by improving of the signal-to-noise ratio in input measurements as described in Section 5.4.

The limits of improvement through changing the assignment of the bounds can be seen on Figure C.13, where bounds were assigned optimally, using the prior knowledge of the actual nonces and the private key. Such a success rate can also be achieved by improvement of the signal-to-noise ratio in input measurements as described in Section 5.4.

The best results for $min(N)$ that we obtained with geometric bounds were 1000, 1800, 1500 and 2600, respectively, for the `sim`, `sw`, `tpm` and `card` datasets, respectively.

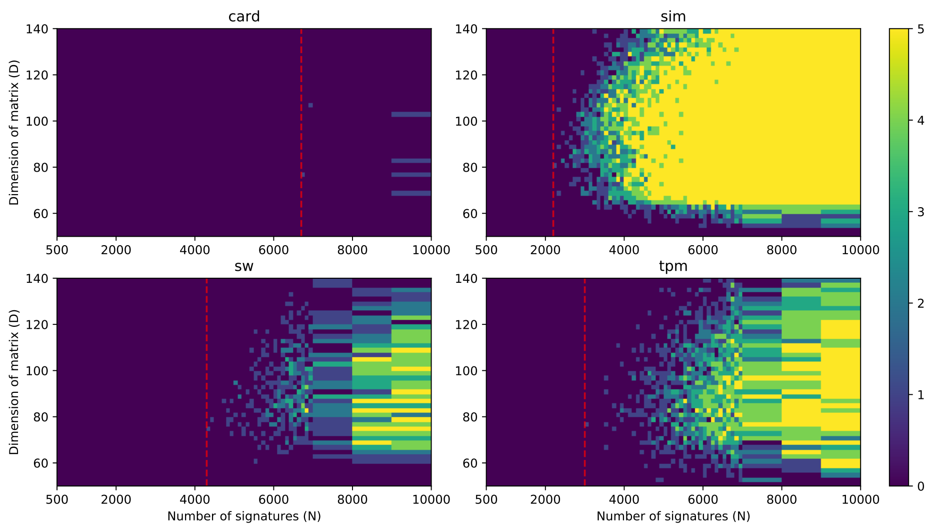

Figure 5.4: Success rate heatmap (out of 5 tries) for constant bounds ($c = 3$).

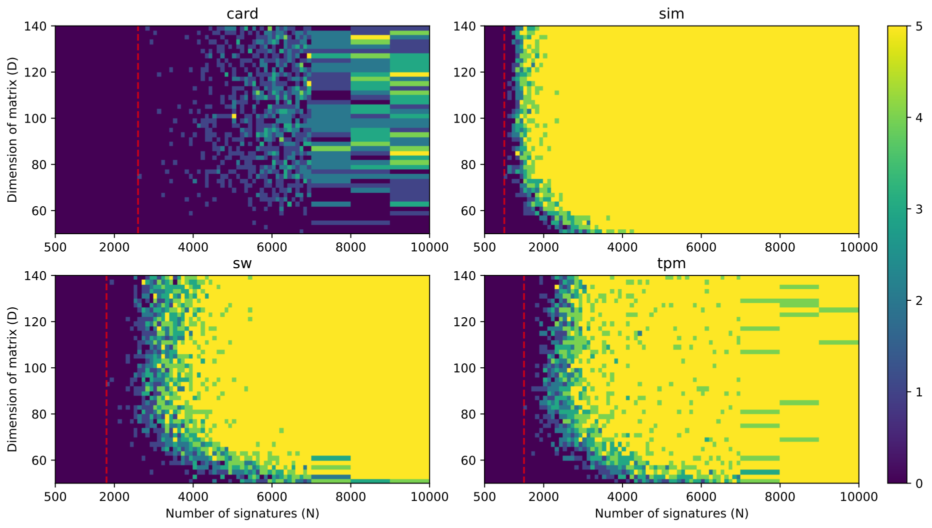

Figure 5.5: Success rate heatmap (out of 5 tries) for geometric bounds.

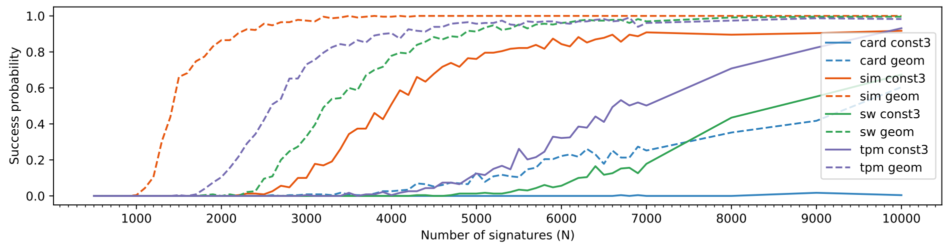

Figure 5.6: Success rate (averaged over all analyzed dimensions) of geometric bounds and constant bounds ($c = 3$) on the various datasets. Note that better results would be obtained if considering only dimensions above 70 according to Figure 5.4 and 5.5.

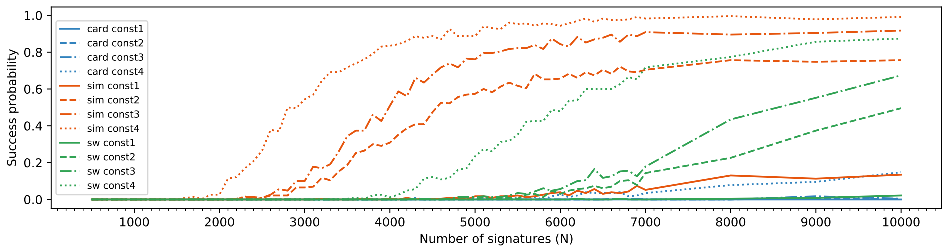

Figure 5.7: Success rate (averaged over all analyzed dimensions) of constant bounds with $c \in \{1,2,3,4\}$ on the various datasets.

CVP/SVP approach. In this experiment, we combined the geometric bounds with either solving the HNP via SVP and examining the reduced basis, or with CVP via Babai's nearest plane algorithm.

As expected (e.g., from [Ben+14]), Babai's nearest plane algorithm always performed worse than solving the HNP via SVP and direct search through the short basis vectors. The negative shift in success rate can be seen in Figure 5.8. We note that there are more methods of solving the SVP or CVP problems, such as enumeration [GNR10] or sieving [Alb+19], that we did not use in this work, mainly due to their runtime requirements. We expect these methods to provide better results than the simple Babai's nearest plane algorithm for CVP or the simple search through reduced lattice basis vectors for SVP.

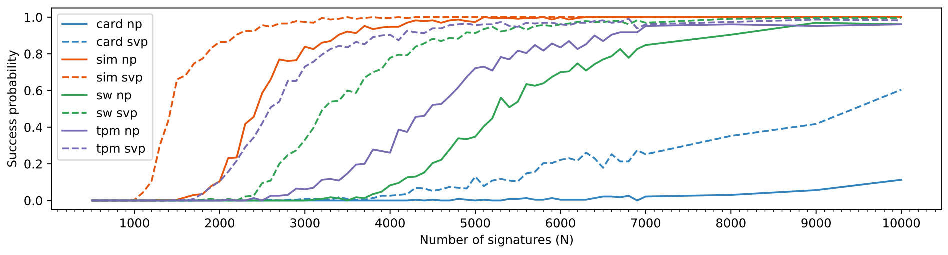

Figure 5.8: Success rate (averaged over all analyzed dimensions) of SVP (dashed line) and Babai's nearest plane (NP, solid line) algorithm, using geometric bounds.

Recentering. The application of recentering, along with geometric bounds and SVP solving, presented significant improvements to success rate for all but the `card` dataset (see Figure 5.9). The improved $min(N)$ values are 500, 1200, 900 and 2100, respectively, for the `sim`, `sw`, `tpm` and `card` datasets, respectively. Using recentering on the noisiest `card` dataset increased the success rate only slightly, if at all.

Using the true number of leading zero bits as the bounds, together with recentering, shows that with ideal input, the lattice attack achieves the theoretically expected minimum amount of signatures as computed in Section 5.3.2. The attack first succeeds at 400 signatures (see Figure C.13), while the theoretical expected minimum is 342.

In our early experiments, we noted a strange behavior of recentering with biased bounds, which we summarise in Appendix C.2.

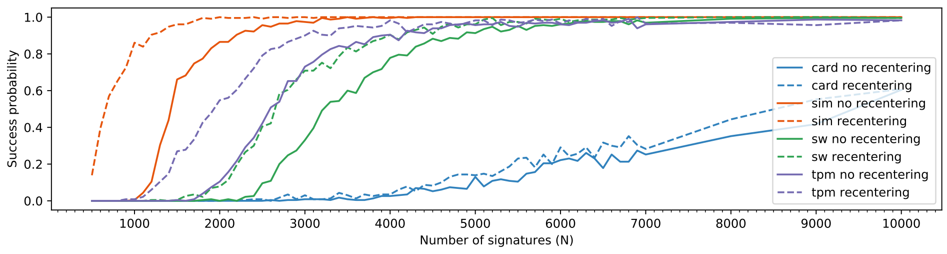

Figure 5.9: Success rate (averaged over all analyzed dimensions) of SVP with and without recentering, using geometric bounds.

Random subsets. To evaluate the random subsets method, as introduced by Brumley and Tuveri [BT11], we performed the attacks for each of the points (N, d) up to 5 times, taking the $1.5d$ fastest signatures and using 100 random subsets of size d to construct the HNP, with geometric bounds, SVP solving and recentering. Due to the high runtime requirements of this attack, we chose to evaluate it only on dimensions D between 90 and 102. The comparison is made to the same attack without taking the random subsets.

Taking random subsets decreased the success rate for all but the `card` datasets (see Figure 5.10), where it produces quite better results, achieving a success probability close to 1 for $N = 10000$. The worse behavior for less noisy datasets can be explained by noting that taking the d fastest signatures is optimal for the amount of information in them, and taking a random subset of $1.5d$ fastest signatures decreases this amount of information.

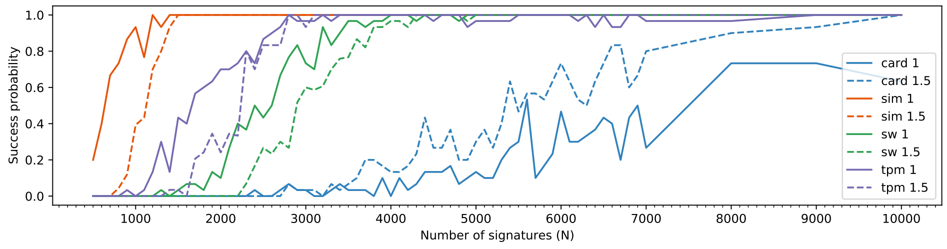

Figure 5.10: Success rate (averaged over all analyzed dimensions) of performing 100 random subsets from $c \cdot d$ fastest signatures, with $c \in \{1, 1.5\}$. The more visible noise in the plot is caused by the smaller range of dimensions analyzed.

Differences. In this experiment, we evaluate the effect of using nonce differences on the attack when using geometric bounds and SVP solving but without recentering, as it is incompatible with taking nonce differences. Performing nonce differences surprisingly resulted in performance comparable to applying recentering to the baseline attack (see Figures 5.9 and 5.11). This might be explained by the fact that taking nonce differences might correct errors if the two nonces share the sequence of erroneous bits. These kinds of errors appear to be quite likely, with most of them being just one bit past the bound, as shown on Figure C.14. However, the success rate for the noisiest `card` dataset decreased.

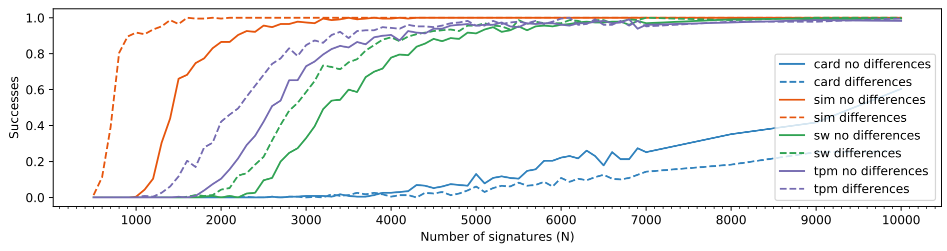

Figure 5.11: Success rate (averaged over the dimensions) of performing the attack using nonce differences on the various datasets.

CVP + changes in u. In this experiment, we extended the block sizes of the progressive BKZ reduction with 57,60, and 64 (to get more thoroughly reduced lattice than in the base approach) and also applied recentering using geometric bounds. Subsequently, we tested all possibilities of correcting errors occurring at exactly 0, 1, 2 or 3 positions (thus running Babai's nearest plane algorithm up to $1 + \binom{140}{1} + \binom{140}{2} + \binom{140}{3} = 457451$ times).

The results show that even though the average number of errors is much higher than three (see Figure C.15), successfully correcting three errors often makes the difference between the successful or failed attack (see Figure 5.12). However, the method also increased runtime significantly, with both the stronger BKZ reduction and the brute-forcing of changes taking extra time. One attack run thus got prolonged from a few minutes to a maximum of four hours. Still, due to its parallelizable nature, this method is a strong candidate for improving the attack through more computation power.

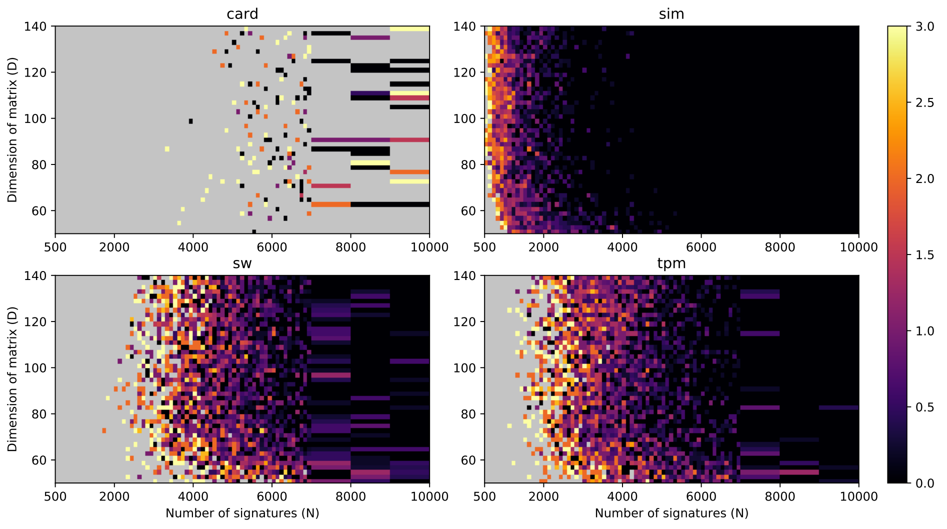

Figure 5.12: Heatmap of average minimal amount of fixed errors required for attack success. Babai's nearest plane algorithm with recentering and geometric bounds was used. Gray color marks areas where none of the attack tries succeeded.

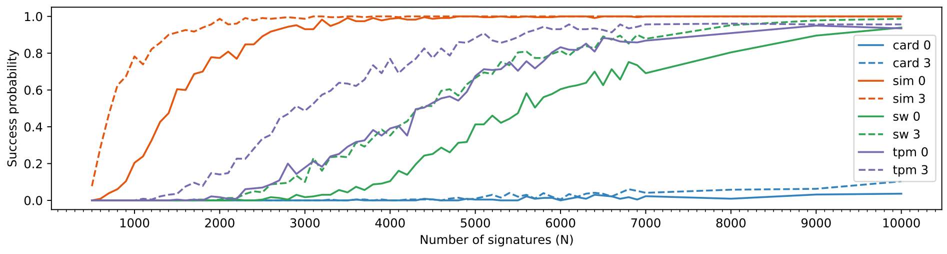

Figure 5.13: Success rate (averaged over the dimensions) of performing the attack using the Babai's nearest plane algorithm, recentering and either with up to 3 errors fixed (3), or with no error fixed (0).

5.6 Conclusions

Our work identified a set of practically exploitable vulnerabilities in ECDSA implementations in certified security chips, cryptographic libraries, and open-source security projects, allowing for the extraction of the full private key using time observation of only several thousands of signatures. We further significantly reduced the required number of signatures in situations with noisy measurements by two new HNP-solving methods – using a geometric assignment of bounds instead of a constant one and an exhaustive correction of possible errors before the application of the HNP solver.

The existence of the real-world vulnerable implementations provided the opportunity to create benchmark datasets with realistic noise profiles, used for a systematic comparison of the existing and newly proposed HNP-solving methods. Using simulated and three real-world datasets, we provide the most extensive comparison of variously parametrized methods available so far in the research literature for this specific domain. The results show that:

- Newly proposed geometric assignment of bounds better approximates the real distribution of bit-lengths in the fastest signatures and leads to a decrease in the required number of signatures for successful key recovery.
- HNP solving via SVP always outperforms CVP with Babai's nearest plane algorithm.
- Using random subsets noticeably increases the success rate on noisy datasets.
- Recentering decreases the required number of signatures for the success of the attack by 30% in some datasets.
- Using differences of nonces with the same estimated bit-length instead of original nonces value works surprisingly well, as (potential) errors likely cancel out.
- Exhaustive correction (up to some bound limited only by available computational resources) of potential errors after the CVP lattice reduction significantly decreases the required number of signatures for a successful attack.

Compared to the very recent work of Moghimi et al. [Mog+20] on a similar vulnerability found in STMicroelectronics TPM chips, the new methods achieve key extraction with a much smaller number of signatures required when compared using their original dataset.

Only 900 signatures (see Figure 5.9) are sufficient, though the success rate with such amount of signatures is low. The success rate improves significantly at 2000 signatures, which is still an order of magnitude lower than 40 000 signatures, as reported by [Mog+20].

While the actual private key extraction demands non-trivial methods, the leakage itself is relatively easy to detect, showing surprising deficiencies in the testing of security devices and cryptographic libraries – we release an open-source tester to support such assessment.

6 A unified approach to special-point-based curve attacks[1]

Since the initial proposal of elliptic curve cryptography (ECC) by Koblitz [Kob87] and Miller [Mil85], the main building block of most elliptic curve cryptosystems has been scalar point multiplication, which involves a plethora of different formulas. There are several side-channel attacks targeting the formulas, either via forcing an intermediate value to be zero or by causing the computation to fail. However, these attacks are only described in special cases, specific to a small number of formulas. In this chapter, we unify and generalize the attacks, and systematically classify exceptional points in most widely used formulas.

Related work. In 2003, Goubin [Gou03] presented a new side-channel attack on implementations of elliptic curve cryptography. Titled Refined power analysis (RPA), it uses a power side channel and the existence of points with a zero coordinate to mount an adaptive attack on static elliptic curve Diffie-Hellman (ECDH). Smart [Sma03] described effective countermeasures against RPA. Subsequently, Akishita and Takagi [AT03] proposed a slightly different method named the Zero-value point (ZVP) attack. It focuses on forcing zeros into intermediate values inside a given point addition formula, and not only in the point coordinate. Several extensions followed: Zhang, Lin, and Liu [ZLL12] modified the ZVP attack to target genus 2 curves, and Crepeau and Kazmi [CK] proposed ZVP for elliptic curves over binary extension fields. Danger, Guilley, Hoogvorst, Murdica, and Naccache [Dan+13a] gave new countermeasures against ZVP and RPA, while Martinez, Sadornil, Tena, Tomas, and Valls [Mar+13] analyzed Edwards curves with regards to ZVP attacks, showing that some addition formulas on Edwards curves are resistant to ZVP attacks. Finally, Murdica, Guilley, Danger, Hoogvorst, and Naccache [Mur+12] proposed the Same Value Analysis (SVA) attack, which tries to detect the repeated use of some finite field value via a side channel.

1. The results in this chapter will be published at Asiacrypt 2021 as "A formula for disaster: a unified approach to elliptic curve special-point-based attacks" with coauthors Jesus-Javier Chi-Dominguez, Jan Jancar and Billy Bob Brumley.

Izu and Takagi [IT03] analyzed the Brier and Joye [BJ02] addition formulas and presented an Exceptional procedure attack (EPA). It uses a similar adaptive mechanism as the aforementioned attacks, but relies on an error side channel by inducing incorrect computations. To avoid EPAs, it is best to use complete addition formulas that always compute the sum of two points correctly for all inputs. Renes, Costello, and Batina [RCB16] credit Bosma and Lenstra [BL95] for the only known complete formulas for prime order short Weierstrass curves, while Bernstein, Birkner, Joye, Lange, and Peters [Ber+08] and Hisil, Wong, Carter, and Dawson [His+08] proposed complete formulas for Twisted Edwards curves. The Explicit-Formulas Database (EFD) by Bernstein and Lange [BL07a] contains formulas for many different curve models and coordinate systems.

What could possibly go wrong? Most of the current public EC libraries do not use complete formulas for short Weierstrass curves, with the exception of ECCKiila [Bel+20]. This includes production libraries:

- Mozilla issued two security advisories for unimplemented exceptions in NSS's projective addition, leading to incorrect (degenerate) multiplication results;
- OpenSSL had unimplemented exceptions during its projective ladder step addition, leading to incorrect (degenerate) results;
- BoringSSL's check for exceptional projective inputs was not constant time, leaking critical algorithm state;
- Python's `fastecdsa` module had an unimplemented exception during affine point doubling, leading to incorrect (degenerate) results.

Section 6.4 examines these cases in more detail, in the context of our framework.

Contributions and outline. In this chapter, we present a novel formal framework to unify the ZVP, RPA, and EPA attacks as instances of a more general problem, which we solve for some cases (Section 6.2). Our approach leads to a new attack on windowed scalar multiplication algorithms (Section 6.2.5). Next, we develop a semi-automated methodology to discover non-trivial exceptional points, applying it to systematically analyze EFD formulas, completely classifying all such points (Section 6.3). We then survey widely deployed software libraries, gaining insight into the practical implications of our analysis (Section 6.4) Finally, we draw our concluding remarks in Section 6.5.

6.1 Background

Let $p \geq 3$, $E_W/\mathbb{F}_p : y^2 = x^3 + ax + b$ be a short Weierstrass curve with $n := \#E(\mathbb{F}_p)$, ψ_k be its k-th division polynomial, and $\phi_k := x\psi_k^2 - \psi_{k+1}\psi_{k-1}$. For any $k \in \mathbb{Z}$, we write $m_k(x_0) := \frac{\phi_{|k|}(x)}{\psi_{|k|}^2(x)}$ as a rational representation of x-only scalar multiplication. Then for all $k_1, k_2, i \in \mathbb{Z}$, we have $(m_{k_1} \circ m_{k_2})(x) = m_{k_1 k_2}(x)$, $m_{k_1}(x) = m_{-k_1}(x)$ and $m_{k_1}(x) = m_{\pm k_1 + in}(x)$.

6.1.1 Curve models and their zero-coordinate points

For a short Weierstrass curve E_W, the points with zero y-coordinate are exactly the points of order 2. Points with zero x-coordinate exist iff b is a square in $\mathbb{F}_p$, in which case $(0, \pm\sqrt{b}) \in E_W(\mathbb{F}_p)$ [HMV04]. Any elliptic curve can be converted to the short Weierstrass model.

Montgmomery. The Montgomery model of an elliptic curve [Mon87; CS18] is

$$E_M/\mathbb{F}_p \colon By^2 = x^3 + Ax^2 + x \quad A, B \in \mathbb{F}_p,\ B(A^2 - 4) \neq 0.$$

Similar to the short Weierstrass model, the neutral element $\mathcal{O}$ does not have an affine representation. Points of order 2 are $(0,0)$ and $(\frac{1}{2}(-A \pm \sqrt{A^2 - 4}), 0)$, though the latter two might not be defined over $\mathbb{F}_p$. All the other affine points have non-zero coordinates.

Twisted Edwards. The twisted Edwards model of an elliptic curve [Ber+08] is

$$E_T/\mathbb{F}_p \colon a_T x^2 + y^2 = 1 + d_T x^2 y^2 \quad a_T, d_T \in \mathbb{F}_p,\ a_T d_T (a_T - d_T) \neq 0.$$

Typically, we also require a_T to be a square in $\mathbb{F}_p$ and d_T a non-square in $\mathbb{F}_p$. The neutral element is the affine point $(0,1)$, the point $(0,-1)$

has order 2, and the points $\big(\pm 1/\sqrt{a}, 0\big)$ have order 4. All the other affine points have non-zero coordinates.

Edwards. The Edwards model of an elliptic curve [Edw07; BL07b] is

$$E_E/\mathbb{F}_p\colon x^2 + y^2 = c^2(1 + dx^2y^2) \quad c,d \in \mathbb{F}_p,\ cd(1 - dc^4) \neq 0.$$

When using `yz` or `yz-squared` coordinates, we also require d to be a square in $\mathbb{F}_p$, though in other cases, we may require it to be non-square. The neutral element is the affine point $(0,c)$, the point $(0,-c)$ has order 2, and the points $(\pm c, 0)$ have order 4. All the other affine points have non-zero coordinates.

For any Edwards curve $E_E/\mathbb{F}_p$, we can rescale $c \mapsto 1$ by taking $d \mapsto dc^4$, $x \mapsto cx$, $y \mapsto cy$ (thus also obtaining a twisted Edwards curve with $a_T = 1$).

6.1.2 Point coordinates and addition formulas

In practice, we mostly work with non-affine coordinates[2], as they delay the costly field inversion required in affine computations. For example, (x,y) can be represented with standard projective coordinates as $(x : y : 1)$, from the set of points $\{(\lambda x, \lambda y, \lambda) | \lambda \in \mathbb{F}_p^*\}$ (that is, projective points are lines in $\mathbb{F}_p^3$, without the zero vector). Some curve models allow performing point additions with either x-only (short Weierstrass and Montgomery models [Mon87]) or y-only (Edwards models [CGF]) coordinates, assuming the difference of the input points is known. Table 6.1 lists the non-affine coordinates used in EFD.

A point addition formula (w.r.t. a given curve model and coordinate system) is an explicit way of computing the sum of two points on an elliptic curve. It takes the coordinates of the two points as inputs and returns the coordinates of their sum, depending on the used representation. There are also formulas for doubling or tripling a point, or for computing the simultaneous doubling of a point and an addition of a different point, known as ladder formulas.

An addition formula is called *unified* if it correctly computes $P + P$ and *complete* if it correctly computes $P + Q$ for any P and Q on any

2. We use the name *non-affine* for coordinate systems other than affine coordinates and *projective* to denote the standard projective coordinates.

Model	Coordinates	(x,y) representation	$\mathcal{O}$ representation	#
E_W	projective [CMO98; BJ02; BL07a; RCB16]	$(xZ\colon yZ\colon Z)$	$(0\colon 1\colon 0)$	21
	jacobian [CC86; CMO98; HNM98; HMV04; Mel07]	$(xZ^2\colon yZ^3\colon Z)$	$(1\colon 1\colon 0)$	36
	modified [CMO98; BL07a]	$(xZ^2\colon yZ^3\colon Z\colon aZ^4)$	$(1\colon 1\colon 0\colon 0)$	4
	w12 with $b=0$ [CLN10]	$(xZ\colon yZ^2\colon Z)$	$(1\colon 0\colon 0)$	2
	xyzz	$(xZ^2\colon yZ^3\colon Z^2\colon Z^3)$	$(1\colon 1\colon 0\colon 0)$	6
	xz [BJ02; IT02]	$(xZ\colon Z)$	$(1\colon 0)$	22
E_M	xz [Mon87]	$(xZ\colon Z)$	$(1\colon 0)$	8
E_T	projective [Ber+08]	$(xZ\colon yZ\colon Z)$	$(0\colon 1\colon 1)$	3
	extended [HCD07]	$(xZ\colon yZ\colon xyZ\colon Z)$	$(0\colon 1\colon 0\colon 1)$	18
	inverted [Ber+08; His+08]	$\left(\frac{Z}{x}\colon \frac{Z}{y}\colon Z\right)$	none	3
E_E	projective [BL07b; HCD07; His+08]	$(xZ\colon yZ\colon Z)$	$(0\colon c\colon 1)$	12
	inverted [BL07c; His+08]	$\left(\frac{Z}{x}\colon \frac{Z}{y}\colon Z\right)$	none	6
	yz [Gau06]	$\left(yZ\sqrt{d}\colon Z\right)$	$\left(\sqrt{d}\colon 1\right)$	6
	yz-squared [Gau06]	$\left(y^2Z\sqrt{d}\colon Z\right)$	$\left(\sqrt{d}\colon 1\right)$	6

Table 6.1: Non-affine coordinates analyzed in this chapter, and the number of respective EFD formulas. Note that the conversion from `xz`, `yz`, and `yz-squared` coordinates to affine is not unique, and that both `yz` and `yz-squared` assume $c = 1$.

curve satisfying the assumptions of the formula. Certainly, unified formulas are important as they do not require exceptions and encourage secure constant-time implementations, where point doubling is indistinguishable from point addition. Any complete formula is also unified, but the converse is not true. For prime order short Weierstrass curves, only a single complete formula is known [RCB16].

6.1.3 Explicit-Formulas Database

The EFD by Bernstein and Lange [BL07a] is the largest publicly available database of formulas for different coordinate systems and curve models. It provides the formulas in a 3-operand notation, breaking down the computation into individual binary and unary operations on intermediate values. This machine readable format mimics the computations in real software and hardware. We exported the EFD data and provide it in a repository [3] with some cleanups and added missing information. The EFD contains addition formulas (i.e., $P + Q = \mathbf{add}(P, Q)$), doubling formulas (i.e., $[2]P = \mathbf{dbl}(P)$), tripling formulas,

3. `https://github.com/J08nY/efd`

differential addition formulas (i.e., $P+Q=\mathbf{dadd}(P-Q,P,Q)$) and ladder formulas (i.e., $([2P],P+Q)=\mathbf{ladd}(P-Q,P,Q)$).

The EFD also includes automated formula verification in Sage, though it only compares the expressions as rational functions. This means the results are correct globally, but not necessarily locally – there might be exceptions for points where the denominators equal zero and the quotient is undefined. We investigate these cases in Section 6.3.

6.1.4 Scalar multiplication algorithms

During ECDH key exchange, all scalar multiplications use a single scalar. The multiplied point is a public key of the other party, which is unknown before the computation. This excludes the use of any heavy precomputations like comb-based methods. Following Jancar [Jan20], we divide the applicable scalar multiplication algorithms into three rough categories:

- *Basic* ones (often called *double-and-add*) that scan the scalar bit by bit, and perform either doubling or addition based on the bit value [HMV04]. During the scalar multiplication, a basic multiplier executes the formulas:

  $$\begin{aligned}[2k]P &= \mathbf{dbl}([k]P) \qquad \text{or}\\ [k+1]P &= \mathbf{add}(P,[k]P),\end{aligned}$$

  depending on the iteration; k is equal to some part of the scalar.

- *Ladder* ones that resemble the basic ones, but use a ladder formula [Mon87] with two temporary variables maintaining a constant difference. This ensures the computations are uniform and take the same time, regardless of the scalar. The formula executions in this scalar multiplier are:

  $$\begin{aligned}([2(k+1)]P,[2k+1]P) &= \mathbf{ladd}(P,[k+1]P,[k]P) \qquad \text{or}\\ ([2k]P,[2k+1]P) &= \mathbf{ladd}(P,[k]P,[k+1]P),\end{aligned}$$

  depending on the iteration.

- *Window* ones that divide the scalar into blocks of digits (called windows) of a given width and precompute the corresponding multiples of the point. The precomputation is cheap enough to be possible even for variable points. If zero digits are skipped, we call the window *sliding* [HMV04]. The formula executions in this scalar multiplier are:

$$\begin{aligned} [2k]P &= \mathbf{dbl}([k]P) \qquad \text{or} \\ [k+e]P &= \mathbf{add}([e]P,[k]P), \end{aligned}$$

depending on the iteration; $[e]P$ is a precomputed point.

Scalar multiplication algorithms can also use signed digit representations of the scalar, most often the binary Non-Adjacent Form (NAF), or in the window case window NAF. In the rest of this chapter, we refer to the *accumulator point* that represents the point variable to which points are added in scalar multiplication, and which stores the current multiple of the input point through the iterations of the algorithm. Note that a ladder-based scalar multiplier has two accumulator points which have a constant difference.

6.1.5 Side-channel attack countermeasures

To mitigate side-channel attacks on ECC, including those discussed in this chapter, several countermeasures were developed over the years. Here we show those important in the context of our attacks, which are based on randomization and target the scalar multiplication with a secret scalar.

Scalar randomization. The first possibility of randomization lies in the secret scalar itself. There are several techniques which randomize the scalar and compute either one scalar multiplication (group scalar randomization) or several (additive, multiplicative, or Euclidean scalar splitting) [Dan+13a]. For us, it is important that this countermeasure leads to randomized multiples of the input point, stored in the accumulator point, as the algorithm proceeds. Thus, if the attacker learns that a particular multiple of the input point was computed during some scalar multiplication, they learn almost nothing about the secret scalar used.

Point randomization. Another possibility of randomizing values inside the scalar multiplication lies in the use of non-affine point representations and their scaling property. As one affine point corresponds to an entire class of non-affine points, one can pick a random representative out of the class when converting the affine input point for scalar multiplication. This randomizes almost all intermediate values in the scalar multiplication [Dan+13a]. It does not randomize zero values in one of the coordinates of the affine point like $(x, 0)$ or $(0, y)$, as their projective representatives are $(xZ \colon 0 \colon Z)$ or $(0 \colon yZ \colon Z)$ for some $Z \in \mathbb{F}_p^*$.

Curve randomization. Finally, it is possible to randomize the curve over which the computations are performed. This also randomizes almost all intermediate values in the scalar multiplication. One example of such randomization is using an isomorphic or an isogenous curve [Dan+13a; Sma03].

6.1.6 The Refined power analysis and Zero-value point attacks

Goubin's RPA [Gou03] is a side-channel attack against ECC implementations using a static secret, such as ECDH or X25519, together with basic or ladder scalar multiplication. It is based on the assumption that adding[4] a point P_0 with a zero x- or y-coordinate to another can be distinguished from adding a general point, at least over several measured traces. We discussed the existence of zero coordinate points in Section 6.1.1. The side channel is usually based on power or electromagnetic emanation, where one can distinguish the multiplication with a zero field element from the general case (see e.g. Fig. 1 in [Dan+13b]) due to the dependency of power consumption of a device on the data and instructions that are being executed. The attacker measures the power consumption of a device using an oscilloscope and a current probe.

In each iteration, the attacker makes a guess $k' \in \mathbb{Z}_n^*$ for the partial secret key k, and then checks the guess by querying the implementation using the public key $P_1 = [k'^{-1} \bmod n]P_0$. The guess was correct iff the implementation computes $[k']P_1 = P_0$, detectable using a side channel. Since the scalar multiplication is iterative in nature, the attacker adaptively guesses the bits of the key one by one, building upon the previous guesses. All scalar randomization countermea-

4. The attack also applies to doubling. For simplicity, we only consider addition in this paper, but our results easily extend to doubling.

sures successfully thwart the RPA attack, as well as Smart's curve randomization via isogenies [Sma03], while point randomization or curve randomization via isomorphisms do not, since the zero point coordinate does not get randomized. Unlike [FGV11], the attack does not require fault injection.

More generally, Akishita's and Takagi's ZVP attack [AT03] considers intermediate scalar values computed during point addition (as a subroutine of scalar multiplication). The intermediate values can be expressed as a polynomial expression in the input coordinates (see Algorithm 4 for an example of the intermediate values and Figure D.20 for the unrolled version). If the attacker can select a point P such that $P + [k']P$ produces a zero scalar intermediate value during the formula's execution (not necessarily at the end), the attacker can detect the zero using a side channel. Then they can deduce which multiples of the input were computed during the scalar multiplication, and thus recover the secret scalar. Unlike RPA, ZVP does not assume the existence of points with a zero coordinate; in particular, it applies to prime-order curves.

The value of the input point P depends on k', the used formulas, the particular intermediate value that is being zeroed out, as well as the curve. It seems that finding these points for even a mildly large k' is an open problem, claimed to be as difficult as computing the k'-th division polynomial. The maximal k' required for key recovery is in the same range as the secret scalar, approaching n. For some coordinate systems and formulas for (twisted) Edwards curves, the intermediate expressions can be classified [Mar+13], but the general case is not settled. The ZVP attack can be thought of as a generalization of the RPA attack, and the same countermeasures prevent it.

6.1.7 Exceptional procedure attacks

In practice, scalar multiplications use non-affine point representations (shown in Table 6.1), only mapping the non-affine result into its unique affine representation at the end. This final conversion is the only part of the computation requiring field inversions, usually of the Z-coordinate. EPA are based on finding a pair of points P and Q such that the final conversion of $P + Q = \mathbf{add}(P, Q)$ fails, because the expression being inverted is zero. The implementation then either throws an error, or produces an obviously detectable output [IT03]. Such points are called exceptional w.r.t. a given formula; see Section 6.3 for a more precise definition and classification of all non-trivial exceptional points for EFD formulas.

6.2 A unified approach to the attacks

The attacks introduced in Section 6.1.6 and Section 6.1.7 have a lot in common. In this section, we build a common framework that captures them as special cases.

6.2.1 Attack setting

Let $S : (k, P) \mapsto [k]P$ be a scalar point multiplication algorithm on a curve. Assume k is a fixed secret input, and P is an arbitrary affine point. This scalar multiplication with a fixed secret scalar and chosen input point is the target in our setting. The evaluation of $S(k, P)$ consists of a sequence of formula executions. As described in Section 6.1.4 and displayed in Figure 6.1, the formulas take as input some multiples of P, which depend on k and S.

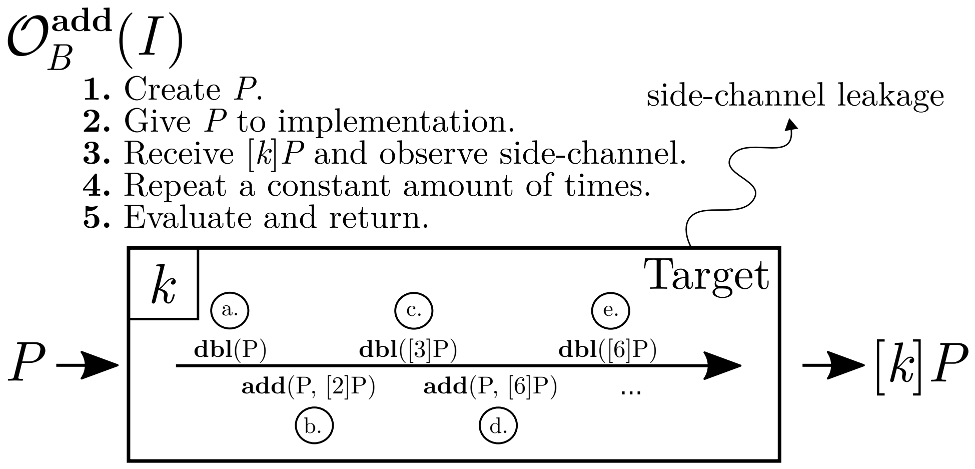
$\mathcal{O}_B^{\mathbf{add}}(I)$
1. Create P.
2. Give P to implementation.
3. Receive $[k]P$ and observe side-channel.
4. Repeat a constant amount of times.
5. Evaluate and return.
side-channel leakage
k
Target
a.
c.
e.
P
dbl(P)
dbl([3]P)
dbl([6]P)
add(P, [2]P)
add(P, [6]P)
...
b.
d.
$[k]P$

Figure 6.1: An example of the Boolean special point oracle, with a target performing the $S(k, P)$ scalar multiplication execution using a basic double-and-add-always algorithm. The scalar k has MSBs 110.

Let us define $\mathcal{O}^{\mathcal{F}}_{B,U} : \mathbb{I}^m \to \{0,1,\perp\}$, the *Boolean special point oracle for formula $\mathcal{F}$*:

$$\mathcal{O}^{\mathcal{F}}_{B,U}(I) := \begin{cases} 1 & \text{if } I \text{ was input into } \mathcal{F} \text{ during } S(k,P) \text{ computation;} \\ 0 & \text{if } I \text{ was not input into } \mathcal{F}; \\ \perp & \text{if the oracle could not determine the result,} \end{cases}$$

where $\mathcal{O}^{\mathcal{F}}_{B,U}(I) \in \{0,1\}$ for $I \in U$, and $\mathbb{I} = \{[i] | i \in \mathbb{Z}\} \cup \{\_\}$ is the set of symbolic multiples of the input point P, (with $[i]$ representing the point $[i]P$ and _ representing any multiple of the point P). When $U = \mathbb{I}^m$, we omit the subscript, and we simply write I instead of $\{I\}$. The arity m of the oracle is the same as the arity of $\mathcal{F}$, e.g., 2 for **add**.

We define the *temporal special point oracle* $\mathcal{O}^{\mathcal{F}}_{T,U} : \mathbb{I}^m \to \{0,1,\perp\} \times \mathcal{P}(\mathbb{N})$ as $\mathcal{O}^{\mathcal{F}}_{T,U}(I) = (\mathcal{O}^{\mathcal{F}}_{B,U}(I), \mathcal{T})$, where $\mathcal{T}$ is a set of iteration indices when $\mathcal{F}$ took I as an input. If the oracle cannot distinguish between a multiple $[i]$ and its negative $[-i]$, we add $\pm$ to its notation and obtain $\mathcal{O}^{\mathcal{F}}_{\pm B,U}$ and $\mathcal{O}^{\mathcal{F}}_{\pm T,U}$.

An example instance of the Boolean oracle is $\mathcal{O}^{\mathbf{add}}_{B}$, which given $I = (\_, [3])$ returns 1 iff the formula **add** ever received as its second input $[3]P$ during the $S(k,P)$ computation. A different example of an oracle, useful in the case of a windowed S, is $\mathcal{O}^{\mathbf{add}}_{T}$ with input $I = ([5], \_)$. It returns all of the iterations in which the **add** formula took $[5]P$ as its first input. We assume an instance of the oracle makes a constant amount of queries to the implementation performing the scalar multiplication, with chosen input points.

Section 6.2.4 shows how to construct instances of the Boolean and temporal special point oracles using the techniques of RPA, ZVP, and EPA attacks, as well as how to use these oracles in an attack.

6.2.2 The dependent coordinates problem

To unify the attacks, we introduce an abstract problem and analyze it.

According to the Section 6.1 definition, for the rest of this section we fix a prime $p \geq 3$ and an elliptic curve $E/\mathbb{F}_p$ given[5] by $Y^2 = f_E(X)$, where $f_E(X) = X^3 + aX + b$ and $a, b \in \mathbb{F}_p$. Let $\mathbb{G}$ be a subgroup of

5. In principle, our techniques apply to other curves models as well, but we use the short Weierstrass model for simplicity, as it represents all curves.

$E/\mathbb{F}_p$ with prime order q. Recall that m_k is the x-coordinate of the rational multiplication-by-k function on E. Furthermore, let

$$R_E := \mathbb{F}_p[X_1, X_2, Y_1, Y_2]/(Y_1^2 - f_E(X_1), Y_2^2 - f_E(X_2))$$

be the coordinate ring of E, and for a multivariate polynomial g, let $\deg g$ denote its multi-degree, given as the sum of its degrees with respect to all individual variables. Finally, note that lower case letters denote scalar values, whereas upper case letters denote either free variables or curve points.

Definition 6.2.1 (DCP: dependent coordinates problem.)**.** Given a polynomial $f \in \mathbb{F}_p[X_1, X_2, Y_1, Y_2]$ and an integer k, find a pair of points (if they exist) $P, Q \in \mathbb{G}$ such that $Q = [k]P$ and $f(X_1, X_2, Y_1, Y_2) = 0$, where $P = (X_1, Y_1), Q = (X_2, Y_2)$. If $f \in \mathbb{F}_p[X_1, X_2]$, we call the problem the *x-only dependent coordinates problem*, or xDCP.

Without loss of generality, we can also consider $k \in \mathbb{Z}_q$ instead of $k \in \mathbb{Z}$, and replace f by any of its representatives from R_E.

Solving the DCP via the xDCP. The following lemma utilizes the curve equation and successive squaring to eliminate potential occurrences of Y_1 and Y_2.

Lemma 1. Let $f \in \mathbb{F}_p[X_1, X_2, Y_1, Y_2]$, $k \in \mathbb{Z}$ and let (P, Q) be a solution to the DCP determined by f and c. Then there exists a polynomial $f' \in \mathbb{F}_p[X_1, X_2]$ such that (P, Q) is also a solution to the xDCP determined by f' and k and $\deg f' \leq 6 \cdot \deg f + 12$.

Proof. Working in R_E, we replace all even powers of Y_1 and Y_2 by powers of $f_E(X_1)$ and $f_E(X_2)$, respectively; representing f as $f_0 + f_1 Y_1 + f_2 Y_2 + f_{12} Y_1 Y_2$ for some $f_0, f_1, f_2, f_{12} \in \mathbb{F}_p[X_1, X_2]$. Next, we eliminate

Y_1 and Y_2:

$$\begin{aligned}
f_0 + f_1Y_1 + f_2Y_2 + f_{12}Y_1Y_2 &= 0\\
Y_1(f_1 + f_{12}Y_2) &= -(f_0 + f_2Y_2)\\
f_E(X_1)(f_1 + f_{12}Y_2)^2 &= (f_0 + f_2Y_2)^2\\
f_E(X_1)(f_1^2 + f_{12}^2 f_E(X_2) + 2f_1f_{12}Y_2) &= f_0^2 + f_2^2 f_E(X_2) + 2f_0f_2Y_2\\
Y_2(f_E(X_1)\cdot 2f_1f_{12} - 2f_0f_2) &= f_0^2 + f_2^2 f_E(X_2)\\
&\quad - f_E(X_1)(f_1^2 + f_{12}^2 f_E(X_2)\\
f_E(X_2)(f_E(X_1)\cdot 2f_1f_{12} - 2f_0f_2)^2 &= (f_0^2 + f_2^2 f_E(X_2)\\
&\quad - f_E(X_1)(f_1^2 + f_{12}^2 f_E(X_2))^2.
\end{aligned}$$

Thus, instead of finding the roots of f, we find the roots of f', where

$$\begin{aligned}
f' =& f_E(X_2)(f_E(X_1)\cdot 2f_1f_{12} - 2f_0f_2)^2\\
&- \left(f_0^2 + f_2^2 f_E(X_2) - f_E(X_1)(f_1^2 + f_{12}^2 f_E(X_2))\right)^2.
\end{aligned}$$

To conclude the proof, it suffices to estimate

$$\begin{aligned}
\deg f' =& \max\{2\cdot\max\{\deg f_1f_{12} + 3, \deg f_0f_2\} + 3,\\
& \quad 2\cdot\max\{2\cdot\deg f_0, 2\cdot\deg f_2 + 3,\\
& \quad \max\{2\cdot\deg f_1, 2\cdot\deg f_{12} + 3\} + 3\}\}\\
\le& 4\cdot\max\{\deg f_0, \deg f_1, \deg f_2, \deg f_{12}\} + 12\\
\le& 4\cdot\deg(f_0 + f_1Y_1 + f_2Y_2 + f_{12}Y_1Y_2) + 12\\
\le& 4\cdot\frac{3}{2}\cdot\deg f + 12.
\end{aligned}$$

□ □

Lemma 1 effectively allows us to only consider xDCP instead of DCP for the remainder of this chapter. Yet with care: we lost the information about the signs of Y_1 and Y_2 during the squaring procedure in the proof, so the resulting xDCP also has solutions with incorrect signs (note that xDCP is always sign-agnostic).

The multi-degree bound is loose and might be much lower in many instances. When solving ZVP or EPA, the multi-degree of f is typically between 1 and 8, so the reduction to xDCP is still practical (see Appendix D.3 for a few examples). Furthermore, we can often factor the expressions and take only a single factor as f.

An easy case. If $f \in \mathbb{F}_p[X_2, Y_2]$, then the DCP becomes easy whenever a solution exists. Using Lemma 1, we instead solve xDCP with $f' \in \mathbb{F}_p[X_2]$, finding the roots algorithmically. If there is a root corresponding to the x-coordinate of some point Q, we simply compute $P = [k^{-1} \bmod q]Q$ and we are done. Note that this approach relies heavily on ignoring the relationship between P and Q until the very end. In particular, the solvability of DCP does not depend on the size of k in this case. This contrasts the claims of Akishita and Takagi [AT03], who found constructing ZVP points for addition (which amounts to solving an instance of the DCP) as hard as computing the k-th division polynomial.

The number of solutions. We now estimate the number of k's such that the xDCP has a solution. If f is linear in one of its variables, say X_1, then for any $x_2 \in \mathbb{F}_p$, there is exactly one x_1 such that $f(x_1, x_2) = 0$ (except for rare cases when $F(X_1, x_2)$ is a constant polynomial). The probability that both x_1 and x_2 are the x-coordinates of $P, Q \in \mathbb{G}$ is roughly $\frac{1}{4} \cdot \frac{q}{n}$. For any such point pairs, there is exactly one $k \in \mathbb{Z}_q$ such that $Q = [k]P$, corresponding to the two possible solutions k, $q - k$. Even though such k's can overlap, we estimate the number of k's for which xDCP has a solution as $2 \cdot p \cdot \frac{1}{4} \cdot \frac{q}{n} \approx \frac{p}{2}$ when $\mathbb{G}$ is a large subgroup. The same heuristic applies when the degree D of at least one variable in f is coprime to $\varphi(p) = p - 1$, since taking the D-th power is an invertible operation in $\mathbb{F}_p$. In general, the correspondence between the roots of f is more problematic, but based on our empirical results, the above heuristic still seems to be reasonably accurate.

6.2.3 Solving the xDCP

The basic strategy to solve the xDCP described in [AT03] is setting $X_2 = m_k(X_1)$ and then finding the roots of $f(X_1, X_2) \in \mathbb{F}_p[X_1]$. If any of the roots is an x-coordinate of a point $P' \in \mathbb{G}$, we take $P = P', Q = [k]P$. The main limitation is that m_k is very hard to compute for large $k \geq B$. In practice, $B \approx 2^{20}$, mainly due to memory requirements.

Shifting the scalar. Suppose that both l and kl are small modulo q for some $l \in \mathbb{Z}$. Then we set $X_1 = m_l(X), X_2 = m_{kl}(X)$, and find the roots of $f(X_1, X_2) \in \mathbb{F}_p[X_1]$. If any of them is an x-coordinate of a point $P' \in \mathbb{G}$, we take $P = [l]P'$, $Q = [k]P$.

In practice, we find the shortest vector in the lattice generated by $\begin{pmatrix} 1 & k \\ 0 & q \end{pmatrix}$ using the Lagrange-Gauss algorithm, and take l as its first coordinate. Effectively, this increases the size of the set of all k's for which we can solve the xDCP to almost B^2, compared to B for the basic approach.

Using the greatest common divisor. To avoid expensive root-finding of a large polynomial, we suggest to construct another polynomial with the same roots, and compute the greatest common divisor (gcd). Replacing m_{kl} with $m_{|q-kl|}$ in the above method offers such a polynomial. Since $m_{|q-kl|}$ might not be directly computable, we reduce both its numerator and its denominator modulo the first polynomial at every step. This does not influence the gcd. Finally, we perform a final reduction after substituting it into f.

More precisely, let $\mathrm{num}(g)$ denote the numerator of a rational function g. Let $X_1 = m_l(X)$, $X_2 = m_{kl}(X)$, and define $F_1 = \mathrm{num}(f(X_1, X_2))$. Furthermore, let $X_2' \equiv m_{|q-kl|}(X) \bmod F_1$ and $F_2 = \mathrm{num}(f(X_1, X_2'))$. Then we efficiently compute $F = \gcd(F_1, F_2)$ using Euclid's algorithm. If any of the roots of F is an x-coordinate of a point $P' \in \mathbb{G}$, we take $P = [l]P', Q = [k]P$. Heuristically, it seems that F is always linear.

Minor scalar optimizations. The symmetry between P and Q, and the fact that $m_k(x) = m_{-k}(x)$ for all $x \in \mathbb{F}_p$, allows us to replace k with $\pm k^{\pm 1} \bmod q$. This saves up to two bits.

6.2.4 The full attack

We now show that RPA, ZVP, and EPA are all special cases of the same attack, utilizing different side channels and the dependent coordinates problem to build an instance of the special point oracle.

The adaptive approach. As mentioned in Section 6.2.1, the multiples which are input into the formulas during a scalar multiplication operation depend on the scalar. These multiples allow us to reconstruct the scalar, as they determine the corresponding addition chain. For example, step e) in Figure 6.1 computes either $\mathbf{dbl}([6]P)$ or $\mathbf{dbl}([7]P)$, depending on the third most significant bit of the scalar.

During the attack, we have a known part of the scalar. It starts empty, and we recover it in the same way the scalar multiplication algorithm processes it. Given a known part, we make a guess on the next subpart, either a single bit or a window of bits, then use some special point oracle to determine whether the guess was true. This implies some multiples derived from the known part and next subpart were input into a formula. This way, we recover the scalar in logarithmically many queries to the oracle.

The type of oracle we have access to, and the scalar multiplication algorithm used, both affect the attack. For example, if a fixed window scalar multiplication algorithm is used and we have access to a $\mathcal{O}^{\mathbf{add}}_{T,([e],\_)}$ for e ranging over all of the precomputed multiples of the input point, we can recover the window digits directly and assemble the scalar afterwards. If on the other hand a basic scalar multiplication algorithm is used and we have an $\mathcal{O}^{\mathbf{dbl}}_{B,([e])}$ for e ranging over all possible scalars, we recover the scalar adaptively. Given a known part of the scalar k', we can gain the next bit based on the output of $\mathcal{O}^{\mathbf{dbl}}_{B,([k'])}$ or $\mathcal{O}^{\mathbf{dbl}}_{B,([k'+1])}$.

All of the RPA, ZVP, and EPA attacks utilize this adaptive approach, differing only in how they construct a special point oracle (i.e. which side channel and property of the curve, formula, or implementation they use).

Constructing oracles from ZVP. Given a point addition formula, we consider the intermediate polynomials, and pick any one of them as f. A solution to the dependent coordinates problem for some k then allows us to construct a point P such that f will evaluate to zero during the computation of $P + [k]P$. Now using a suitable side channel, we can detect whether this zero appears during the scalar multiplication, and potentially localize it into an iteration of the scalar multiplication algorithm [AT03]. Thus we can construct an instance of the $\mathcal{O}^{\mathbf{add}}_{T,([1],[k])}$ oracle for all k for which we can solve the (x)DCP[6]. Similarly, considering the intermediate polynomials in a doubling formula and zeroing out some of them for an input of $[k]P$ allows us to construct an instance of the $\mathcal{O}^{\mathbf{dbl}}_{T,([k])}$ oracle. Note that in the case of the addition formula, if the chosen intermediate polynomial f depends only on one of the input points, it is possible to construct the $\mathcal{O}^{\mathbf{add}}_{T,(\_,[k])}$ and $\mathcal{O}^{\mathbf{add}}_{T,([1],\_)}$ oracles.

6. We cannot always consider affine representations as f might not be homogeneous, but in practice this is not a problem, as we have freedom in choosing f.

Constructing oracles from RPA. This is a special case of ZVP in which the intermediate value to zero out is a coordinate of an input point [Gou03]. This leads to an easy case of the (x)DCP, discussed in Section 6.2.2, as $f = X_2$ or $f = Y_2$. Because this oracle construction approach leads to an easy case of the (x)DCP, there is no bound on the multiple k in the constructed oracle instances $\mathcal{O}^{\mathbf{add}}_{T,(\_,[k])}$. One can also construct oracle instances such as $\mathcal{O}^{\mathbf{add}}_{T,([1],\_)}$ or $\mathcal{O}^{\mathbf{dbl}}_{T,([k])}$, but not $\mathcal{O}^{\mathbf{add}}_{T,([1],[k])}$ as the appearance of a zero in one of the input points necessarily does not depend on the other point.

Whether these RPA oracles can be constructed depends on the properties of the curve, i.e. whether it has the points $(x, 0)$ or $(0, y)$. Note that if both a point and its negative have a zero-coordinate (as is the case of the $(0, y)$ point on short Weierstrass curves), one can only use it to construct $\mathcal{O}^{\mathcal{F}}_{\pm T}$ and $\mathcal{O}^{\mathcal{F}}_{\pm B}$ oracles.

Constructing oracles from EPA. In this case, the side channel used to construct the oracle is an error one. The oracle detects whether a computation fails because of an undefined inversion. As explained in Section 6.1.7, this can only happen at the very end of the scalar multiplication, when mapping the result back to affine coordinates, so we can take f to be the expression by which we divide. If we can solve the (x)DCP for this f and some k, we can input this point[7] into the scalar multiplication, which will fail if it computes $P + [k]P$, enabling us to construct a $\mathcal{O}^{\mathbf{add}}_{B,([1],[k])}$. Note that this is a Boolean oracle: with the error side channel we can only detect that the map back to affine coordinates failed, and not during which iteration the zero occured.

6.2.5 Window method attack

The main limitation of the ZVP-based attacks compared to RPA-like attacks is that they allow the attacker to recover only a limited number of secret scalar bits. This is due to the need for solving a hard case of the (x)DCP with large k. We show that these attacks can extract the full scalar when the target algorithm is window-based, or more generally adds points to the accumulator point from a set of precomputed input point multiples, conditionally on secret scalar bits.

7. The homogeneity of f allows us to only consider affine representations.

The attack requires that the addition formula in question has an intermediate value which depends only on one of the operands, thus producing an easy case of the DCP as mentioned in Section 6.2.2. Together with an appropriate side channel, this allows the attacker to construct a $\mathcal{O}^{\mathbf{add}}_{T,([e],\_)}$ oracle. Note that the attacker needs a temporal special point oracle, and not a Boolean one, as the event that the e-th multiple was added to the accumulator point somewhere in the scalar multiplication is insufficient to extract information on the secret scalar. Once the attacker is able to detect the relevant iterations, the attacker varies over all values e in the set of precomputed multiples, based on the algorithm. In this way, the attacker recovers the full secret scalar.

This attack works even if the curve has no RPA points $(0,y)$, $(x,0)$, and thus RPA does not apply. However, the target must use a suitable algorithm with an addition formula that has a suitable intermediate value to zero out.

6.3 Classifying the exceptional points

While many EFD formulas [BL07a] are not complete, we are not aware of any systematic overview of the respective pairs of exceptional points. To rectify this, we implemented tooling for unrolling the formulas and tracing their intermediate values. The tooling is an extension of **pyecsca** [Jan20] (**Py**thon **E**lliptic **C**urve cryptography **S**ide-**C**hannel **A**nalysis) – a Python toolkit that aims to extract information from black-box implementations of ECC through side channels and offers extensive simulations of ECC implementations.

Our methodology loosely combines two very different, yet complementary, techniques: fuzzing and manual analysis.

Fuzzing. To quickly identify possible exceptional points (and later verify our findings heuristically), an automated approach is useful. We fuzzed small curves of all relevant types, trying all pairs of input points for all the analyzed formulas. This approach scales well, but at the cost of an inherently high number of false positives (and possibly false negatives). The results for small curves do not always generalize, and the unrolling tool might not handle all corner cases correctly.

Manual analysis. To find the sufficient and necessary conditions that classify all the exceptional points, we resort to manual inspection. Compared to the cost of fuzzing, it takes much more effort, argumentation, and attention to detail. But in the end, it provides more insight, and is applicable to all relevant curves of all sizes. In this light, the fuzzing part of our methodology focuses on breadth, while the manual analysis on depth.

We carefully went through all 111 addition formulas and 42 differential addition/ladder formulas[8] in the EFD[9], and studied when the expressions by which we divide during the conversion to affine coordinates could be zero[10]. Namely, for addition, this amounted to studying the conditions $X_3 = 0$ or $Y_3 = 0$ for (twisted) inverted Edwards coordinates, $ZZ_3 = 0$ or $ZZZ_3 = 0$ for short Weierstrass xyzz coordinates[11], and $Z_3 = 0$ for all other coordinates. The variable's subscript denotes its index in the addition formula with 1 and 2 being the inputs and 3 being the output. Similarly, for differential addition and/or ladders, we investigated when the outputs Z_4 and Z_5 equaled zero. Furthermore, the unrolled expressions could be studied to see which formulas are unified, though we did not pursue this path further.

The rest of this section describes the details of our manual analysis. See Table 6.2 for examples of the expressions we refer to.

6.3.1 Exceptional points for addition

We call a pair of points P, Q *exceptional* (w.r.t. some representation) for an addition formula $\mathcal{F}$ if $\mathcal{F}(P, Q) \neq (P + Q)$. If also $P \neq \pm Q$, and both P and Q have odd prime order, we say that P, Q are *non-trivial*. This also implies that $\mathcal{F}(P, Q)$ should always have an affine representation for all $\mathcal{F}$ we discuss.

Short Weierstrass: projective, jacobian, modified, w12, xyzz coordinates. For short Weierstrass curves, non-triviality implies $x_1 \neq$

8. Ladder formulas already include the doubling formulas for the same coordinates.
9. The formulas include various coordinates and models, but some of them are just adaptations for specific coefficients (e.g. $a = -3$ for E_W), mixed additions, etc.
10. potentially omitting the cases where the result is a point at infinity
11. ZZ_i and ZZZ_i are variables whose values equal Z_i^2 and Z_i^3 throughout the computation, respectively.

Coordinates	Formula	Expression
jacobian jacobian-0 jacobian-3	`add-1986-cc`	$Z3 = Z2 * Z1 * (X2 * Z1^2 - X1 * Z2^2)$
	`add-1998-cmo-2`	$Z3 = Z2 * Z1 * (X2 * Z1^2 - X1 * Z2^2)$
	`add-1998-cmo`	$Z3 = Z2 * Z1 * (X2 * Z1^2 - X1 * Z2^2)$
	`add-1998-hnm`	$Z3 = (-1) * Z2 * Z1 * (X2 * Z1^2 - X1 * Z2^2)$
	`add-2001-b`	$Z3 = Z2 * Z1 * (X2 * Z1^2 - X1 * Z2^2)$
	`add-2007-bl`	$Z3 = 2 * Z2 * Z1 * (X2 * Z1^2 - X1 * Z2^2)$
	`madd-2004-hmv`	$Z3 = Z1 * (X2 * Z1^2 - X1)$
	`madd-2007-bl`	$Z3 = 2 * Z1 * (X2 * Z1^2 - X1)$
	`madd-2008-g`	$Z3 = (-1) * Z1 * (X2 * Z1^2 - X1)$
	`madd`	$Z3 = 2 * Z1 * (X2 * Z1^2 - X1)$
	`mmadd-2007-bl`	$Z3 = (-1) * 2 * (X1 - X2)$
	`zadd-2007-m`	$Z3 = (-1) * Z1 * (X1 - X2)$

Table 6.2: Jacobian coordinate outputs on short Weierstrass curves.

x_2. Moreover, we do not need to consider the expression corresponding to Z_3 in the formulas by Renes, Costello, and Batina [RCB16], as Bosma and Lenstra [BL95] prove their completeness. Since none of the Z_3 expressions depend on a particular representation of a point, we can (without loss of generality) assume

$$Z_1 = Z_2 = ZZ_1 = ZZ_2 = ZZZ_1 = ZZZ_2 = 1$$

when searching for non-trivial exceptional points, implying $x_i = X_i, y_i = Y_i$. With this in mind, there is only a single factor that could possibly be zero in the studied Z_3 expressions, namely $(y_1 + y_2)^3$. This factor is present in all variants of the Brier-Joye [BJ02] formulas (`add-2002-bj`) and Bernstein-Lange [BL07a] formulas (`add-2007-bl`), illustrated in Algorithm 4. Note that $(y_1 + y_2)^3 = 0$ is equivalent to $y_1 = -y_2$, which implies

$$\begin{aligned} x_1^3 + ax_1 + b = y_1^2 = y_1^2 = y_2^2 &= x_2^3 + ax_2 + b \\ (x_1^2 + x_1x_2 + x_2^2 + a)(x_1 - x_2) &= 0 \\ x_1^2 + x_1x_2 + x_2^2 + a &= 0, \quad \text{since } x_1 \neq x_2. \end{aligned}$$

Thus, we get a family of non-trivial exceptional points

$$P = (x,y) \text{ and } Q = (x', -y) \text{ with } x \neq x',$$

equivalently characterized by $x^2 + xx' + x'^2 + a = 0$, which is a possible input to the xDCP. Izu and Takagi [IT03] previously identified this family for the `add-2002-bj` case, but not for the `add-2007-bl` one.

Algorithm 4: Point addition formula `add-2007-bl` in projective coordinates.

```
Input :$E/\mathbb{F}_p \colon y^2 = x^3 + ax + b$,
       $P, Q \in E/\mathbb{F}_p$,
       $P = (X_1 \colon Y_1 \colon Z_1)$,
       $Q = (X_2 \colon Y_2 \colon Z_2)$
Output:$(X_3 \colon Y_3 \colon Z_3) = P + Q$
$U_1 \leftarrow X_1 \cdot Z_2$;
$U_2 \leftarrow X_2 \cdot Z_1$;
$S_1 \leftarrow Y_1 \cdot Z_2$;
$S_2 \leftarrow Y_2 \cdot Z_1$;
$ZZ \leftarrow Z_1 \cdot Z_2$;
$T \leftarrow U_1 + U_2$;
$TT \leftarrow T^2$;
$M \leftarrow S_1 + S_2$;
$t_0 \leftarrow ZZ^2$;
$t_1 \leftarrow a \cdot t_0$;
$t_2 \leftarrow U_1 \cdot U_2$;
$t_3 \leftarrow TT - t_2$;
$R \leftarrow t_3 + t_1$;
$F \leftarrow ZZ \cdot M$;
$L \leftarrow M \cdot F$;
$LL \leftarrow L^2$;
$t_4 \leftarrow T + L$;
$t_5 \leftarrow t_4^2$;
$t_6 \leftarrow t_5 - TT$;
$G \leftarrow t_6 - LL$;
$t_7 \leftarrow R^2$;
$t_8 \leftarrow 2 \cdot t_7$;
$W \leftarrow t_8 - G$;
$t_9 \leftarrow F \cdot W$;
$X_3 \leftarrow 2 \cdot t_9$;
$t_{10} \leftarrow 2 \cdot W$;
$t_{11} \leftarrow G - t_{10}$;
$t_{12} \leftarrow 2 \cdot LL$;
$t_{13} \leftarrow R \cdot t_{11}$;
$Y_3 \leftarrow t_{13} - t_{12}$;
$t_{14} \leftarrow F^2$;
$t_{15} \leftarrow F \cdot t_{14}$;
$Z_3 \leftarrow 4 \cdot t_{15}$;
```

(Twisted) Edwards: projective, extended, inverted coordinates. Let $E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$ be a (twisted) Edwards curve [12] (cf. Section 6.1) (note that we do not impose any (non-)square restrictions on $a, d \in \mathbb{F}_p$). In order to go through all the Z_3 expressions and see when they are equal to zero, we introduce the following lemma.

12. We only consider Edwards curves with $c = 1$, since the others can be isomorphically rescaled to this case without affecting the nullity of the Z_3 expressions.

Lemma 2. Let $P = (x_1, y_1), Q = (x_2, y_2)$ be a pair of non-trivial exceptional points on $E_{a,d}$. Then the following holds:

$$x_1x_2y_1y_2 \neq 0, \tag{6.1}$$
$$dx_1x_2y_1y_2 \neq \pm 1, \tag{6.2}$$
$$y_1y_2 \neq -ax_1x_2, \tag{6.3}$$
$$x_1y_2 \neq x_2y_1, \tag{6.4}$$
$$x_1y_2 \neq -x_2y_1, \tag{6.5}$$
$$y_1y_2 \neq ax_1x_2. \tag{6.6}$$
$$x_1y_1 \neq \pm x_2y_2. \tag{6.7}$$

Proof. (6.1) follows from the fact that neither P nor Q are 4-torsion. Hisil et al. [His+08] (Theorem 1, Corollary 1) prove (6.2), (6.3) and (6.4).

Now consider the addition law from [Ber+08]:

$$(x_1, y_1) + (y_1, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

Assume that either (6.5) or (6.6) is not true. Since the denominators are nonzero by (6.2), one of the coordinates of $P + Q$ is zero, which implies $P + Q$ is a 4-torsion point. This is impossible, since both P and Q have odd order and $P \neq -Q$.

Finally, consider the addition law from [His+08]:

$$(x_1, y_1) + (y_1, y_2) = \left(\frac{x_1y_1 + x_2y_2}{y_1y_2 + ax_1x_2}, \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2} \right).$$

Assume that (6.7) is not true. Since the denominators are nonzero by (6.3) and (6.4), one of the coordinates of $P + Q$ is zero, which implies $P + Q$ is a 4-torsion point. This is impossible, since both P and Q have odd prime order and $P \neq -Q$. □

After factoring all the Z_3 expressions, we can (without loss of generality) set $Z_1 = Z_2 = 1$. Then we have $x_i = 1/X_i, y_i = 1/Y_i$ for inverted coordinates, and $x_i = X_i, y_i = Y_i$ for all others. Lemma 2 handles all the possible zero factors, which means that there are no non-trivial exceptional points.

Curve model	Coordinates	Formula name
E_W	projective	`add-2007-bl`
		`add-2002-bj`
	xz	`dadd-2002-it`, `mdadd-2002-it`
		`ladd-2002-it`, `mladd-2002-it`
		`dadd-2002-it-3`, `mdadd-2002-it-3`
		`ladd-2002-it-3`, `mladd-2002-it-3`
		`mdadd-2002-bj`, `mladd-2002-bj`
		`mdadd-2002-bj-2`, `mladd-2002-bj-2`
		`mladd-2002-bj-3`

Table 6.3: Formulas [BL07a; IT02; BJ02] with non-trivial exceptional points. The projective coordinates apply to all formula versions: $a = -1$, $a = -3$ and general a.

6.3.2 Exceptional points for differential addition and ladders

Recall from Section 6.1.3 that differential addition and ladder formulas take representations of three input points $(P - Q, P, Q)$ and return the representation of $P + Q$ or $([2]P, P + Q)$, respectively.

We call a triplet of points $(P - Q, P, Q)$ *exceptional* (w.r.t. a representation) for a differential addition or ladder formula $\mathcal{F}$ if $\mathcal{F}(P - Q, P, Q) \neq P + Q$ or $\mathcal{F}(P - Q, P, Q) \neq ([2]P, P + Q)$, respectively. If moreover $P \neq \pm Q$, and both P and Q have odd prime order, we say that $(P - Q, P, Q)$ are *non-trivial*. This also implies that $\mathcal{F}(P - Q, P, Q)$ should always have an affine representation for all $\mathcal{F}$ we discuss (hence Z_4 and Z_5 should be nonzero).

Short Weierstrass: xz coordinates. In this case, the inputs are $P - Q = (X_1, Z_1)$, $P = (X_2, Z_2)$, $Q = (X_3, Z_3)$ on $E_W/\mathbb{F}_p : y^2 = x^3 + ax + b$; the outputs are (X_4, Z_4) for diff. addition and $(X_4, Z_4), (X_5, Z_5)$ for ladders.

Setting $Z_1 = Z_2 = Z3 = 1$ and $x_1 = X_1, x_2 = X_2, x_3 = X_3$, the only possibilities for $Z_4 = 0$ or $Z_5 = 0$ that arise in the formulas are $x_2 = x_3$, $x_2^3 + ax_2 + b = 0$, and $x_1 = 0$. Only the latter corresponds[13] to a triplet of non-trivial exceptional points $((0 : 1) - Q, (0 : 1), Q)$, whenever b is a square in $\mathbb{F}_p$. The impacted formulas are `{d/l}add-2002-it`,

13. Note that x_1 does not directly affect X_4 nor X_5.

`{d/l}add-2002-it-3`, and their mixed variants, plus `mdadd-2002-bj`, `m{l/d}add-2002-bj-2` and `mladd-2002-bj-3`.

Montgomery: xz coordinates. Here, the inputs are $P - Q = (X_1, Z_1)$, $P = (X_2, Z_2), Q = (X_3, Z_3)$ on $E_M/\mathbb{F}_p : By^2 = x^3 + Ax^2 + x$; the outputs are (X_4, Z_4) for diff. addition and $(X_4, Z_4), (X_5, Z_5)$ for ladders.

Setting $Z_1 = Z_2 = Z_3 = 1$ and $x_1 = X_1, x_2 = X_2, x_3 = X_3$, the only possibilities for $Z_4 = 0$ or $Z_5 = 0$ that arise in the formulas are $x_1 = 0$, $x_2 = 0$, $x_2 = 1/2 \cdot (-A \pm \sqrt{A^2 - 4})$, $x_2 = x_3$ and $(x_2 - 1)(x_3 + 1) = (x_2 + 1)(x_3 - 1)$. Section 6.1 shows that the former three correspond to points of order 2 (though $\sqrt{A^2 - 4}$ might not exist over $\mathbb{F}_p$). The last one implies either $x_2 - 1 = x_3 - 1 = 0$, or $x_2 + 1 = x_3 + 1 = 0$, or else

$$1 - \frac{2}{x_2 + 1} = \frac{x_2 - 1}{x_2 + 1} = \frac{x_3 - 1}{x_3 + 1} = 1 - \frac{2}{x_3 + 1}.$$

In all of these cases, we have $x_2 = x_3$, hence the corresponding points are trivial.

Edwards: yz, yz-squared coordinates. Recall that in these cases, $d = r^2$ for some $r \neq \pm 1$ in $\mathbb{F}_p^*$. The inputs are $P - Q = (Y_1, Z_1)$, $P = (Y_2, Z_2)$, $Q = (Y_3, Z_3)$ on $E_E/\mathbb{F}_p : x^2 + y^2 = 1 + r^2x^2y^2$; the outputs are (Y_4, Z_4) for diff. addition and $(Y_4, Z_4), (Y_5, Z_5)$ for ladders. In fact, (Y_4, Z_4) for ladders is just a special case of (Y_5, Z_5) with $Y_2 = Y_3, Z_2 = Z_3$, so we may ignore it.

Setting $Z_1 = Z_2 = Z_3 = 1$, we get $y_1 = Y_1/r, y_2 = Y_2/r, y_3 = Y_3/r$ for the `yz` coordinates, and $y_1^2 = Y_1/r, y_2^2 = Y_2/r, y_3^2 = Y_3/r$ for the `yz-squared` coordinates, the ladder Z_5 and diff. addition Z_4 coincide for all of these formulas. The only conditions to analyze are $y_1 = 0$ (which is a trivial case as it corresponds to 4-torsion $P - Q$) and

$$(1 + ry_2^2)(1 + ry_3^2) = \frac{r+1}{r-1}\left(1 - ry_2^2\right)\left(1 - ry_3^2\right),$$

which implies

$$\begin{aligned}(r-1)(1 + ry_2^2 + ry_3^2 + r^2y_2^2y_3^2) &= (r+1)(1 - ry_2^2 - ry_3^2 + r^2y_2^2y_3^2)\\ -2 + 2r^2y_2^2 + 2r^2y_3^2 - 2r^2y_2^2y_3^2 &= 0\\ r^2y_3^2(1 - y_2^2) &= 1 - r^2y_2^2. \qquad (6.8)\end{aligned}$$

If $1 - r^2 y_2^2 = 0$, then either $y_3 = 0$ or $y_2^2 = 1$, implying Q or P being 4-torsion. In the other case, we get

$$y_3^2 = \frac{1 - r^2 y_2^2}{r^2(1 - y_2^2)} = \frac{1}{r^2 x_2^2},$$

and since (6.8) is symmetric, analogical arguments yield

$$y_2^2 = \frac{1 - r^2 y_3^2}{r^2(1 - y_3^2)} = \frac{1}{r^2 x_3^2}.$$

Thus the only case left to consider is $x_2^2 y_3^2 = x_3^2 y_2^2 = \frac{1}{r^2}$. But then we have $(1 + dx_2 x_3 y_2 y_3)(1 - dx_2 x_3 y_2 y_3) = 1 - r^4 x_2^2 x_3^2 y_2^2 y_3^2 = 0$, which is impossible for non-trivial exceptional points by (6.2) in Lemma 2.

6.4 Practical implications

This work has several practical implications, stemming from (i) its findings on exceptional points for EFD formulas; (ii) its development of a ZVP-like attack on windowed scalar multiplication methods; and (iii) improvements to the techniques used in the ZVP and EPA attacks.

6.4.1 Impact on cryptographic libraries

We examined the EC arithmetic implementations in 13 popular open-source cryptographic libraries. Table 6.4 lists their scalar multiplication algorithm, coordinates, and addition formulas. The focus of our analysis was on ECDH operations over E_W, and in case the library implements several algorithms, we list the one used for generic curves. Most analyzed libraries use jacobian coordinates, for which we report no classes of non-trivial exceptional points in any of the formulas on EFD. One could conclude that the impact of the new classes of exceptional points is thus negligible. However, these libraries represent only a fraction of the uses of addition formulas. Implementations of EC arithmetic, potentially using one of the addition formulas with non-trivial exceptional points, are found in pairing-based cryptography, password-authenticated key exchange, or many zero-knowledge proof system implementations.

The discovered classes of exceptional points are unexpected from the point-of-view of a developer. While many developers know that formulas which are not complete or unified need special handling, they do not expect seemingly unrelated points causing issues in the formula. We illustrate this by presenting a history of issues surrounding exceptional cases in formulas used by cryptographic libraries.

NSS: unimplemented exceptions. For generic E_W, NSS has three different implementations of EC arithmetic. The first is pure affine, which we disregard. The second is mixed point addition using an implementation of `madd-2004-hmv`, optimized for $a = -3$. However, the code failed to account for the $P = \pm Q$ cases. Furthermore, the corresponding point doubling is an implementation of `dbl-1998-cmo-2`, and failed to account for the $2P = \mathcal{O}$ case. Mozilla issued CVE-2015-2730[14] to track these issues.

NSS: more unimplemented exceptions. The last, and most generic E_W arithmetic in NSS, is mixed point addition using a `madd-2004-hmv` implementation, with no optimizations for curve coefficients. Two years after the previous issue, Valenta et al. [Val+18, Section 7.2] uncovered the analogous flaw in this code. There were no corresponding flaws in point doubling. Mozilla issued CVE-2017-7781[15] to track this issue.

OpenSSL: broken ladder. In 2018, OpenSSL switched to a ladder implementation for generic E_W scalar multiplications. Work by Tuveri et al. [Tuv+18] prompted the change. For the ladder step, the initial code, merged to the development branch, was an implementation of `ladd-2002-it-3`. Unfortunately, this code fails in the case of a particular x-coordinate being zero (Section 6.3.2). One month passed between merging the broken implementation and the fix[16], switching to `ladd-2002-it-4`. The discovery[17] was mostly luck – during standardization, GOST curves utilized generators with the smallest possible x-coordinate.

14. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2730`
15. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7781`
16. `https://github.com/openssl/openssl/pull/7000`
17. `https://github.com/openssl/openssl/issues/6999`

BoringSSL: untaken exceptions leak. Historically, Google's BoringSSL only supports a very narrow subset of curves: P-224, P-256, P-384, P-521, and Curve25519. Weiser et al. [Wei+20] discovered timing leaks in BoringSSL's point addition formulas, affecting the legacy NIST curves in the aforementioned list. The leaks were in three distinct implementations: P-224 and P-256 have dedicated EC arithmetic stacks, while P-384 and P-521 share a single stack. In all cases, the root cause is short circuit logic: a snippet from the vulnerabilities follows.

```
if (x_equal && y_equal && !z1_is_zero && !z2_is_zero)
```

The first two variables are Booleans tracking whether the two x-coordinates are equal (resp. y), and the last two ensure neither operand is $\mathcal{O}$ by checking if the z-coordinates are zero. This C statement violates the constant-address model, a requirement for constant-time code. For instance:

- if the first branch fails, this tells the attacker the x-coordinates are not equal;
- if the second branch fails, this tells the attacker the x-coordinates are equal, but the y-coordinates are not;
- if the third branch fails, this tells the attacker the x- and y-coordinates are equal, and the first operand is $\mathcal{O}$;
- if the fourth branch fails, this tells the attacker the x- and y-coordinates are equal, the first operand is not $\mathcal{O}$, yet the last operand is $\mathcal{O}$;
- if no branch fails, this tells the attacker the x- and y-coordinates are equal, and neither operand is $\mathcal{O}$ (subsequently early exiting to point doubling).

For example, this leak is relevant at the beginning of scalar multiplication, in various cases where the accumulator takes the value $\mathcal{O}$. These (probabilistically) small leaks are often sufficient for lattice-based cryptanalysis of nonce-based digital signature schemes, such as ECDSA. We feel this case is particularly interesting, since it is not the exception itself that usually leaks, but rather the *check* for the exception. Google fixed[18] the issues in 2019.

Python fastecdsa: division by zero. The Python module `fastecdsa` is an extension module, backed by GNU MP, a multiprecision arithmetic library written in C. It implements the ECDSA signature scheme[19], also providing flexible EC arithmetic with affine coordinates. The module supports generic E_W curves, as well as several standardized curves with fixed parameters, and E_W versions of modern E_E and E_T curves such as Curve25519 and Curve448. Using our Section 6.3 methodology, we discovered [20] that the point doubling code does not handle the $2P = \mathcal{O}$ case properly. The C code ignores the return code from GNU MP's modular inversion function. In the $y = 0$ case, this leads to a silent division by zero, and incorrect results for points with even order. While this naturally affected generic `fastecdsa` curves, the E_W versions of Curve25519 and Curve448 were impacted the most. This is because all other standardized curves built into `fastecdsa` have large prime order.

6.4.2 Attack improvements

Previous ZVP attacks targeting addition formulas on different scalar multiplication methods required the computation of large degree division polynomials. This limited the attack to only recover a small amount of secret scalar bits. On the other hand, our proposed attack on windowed scalar multiplication methods from Section 6.2.5 allows the attacker to recover the full scalar. Thus, this shows that windowed methods of scalar multiplication are somewhat more vulnerable to ZVP-like attacks. We simulated the attack using the **pyecsca** toolkit, and were able to recover the full secret scalar from a window NAF algorithm with `add-2016-rcb` formulas on the P-224 curve. In the attack, we do not observe a real power or electromagnetic side channel, but the toolkit simulates the computation down to individual finite field operations, and produces the side-channel output (e.g., whether a zero occurred during computation). Appendix D.1 shows the attack code

18. `https://boringssl.googlesource.com/boringssl/+/12d9ed670da3edd64ce8175c`
19. `https://pypi.org/project/fastecdsa/`
20. `https://github.com/AntonKueltz/fastecdsa/pull/58`

Library	Operation	Scalar multiplier	Coordinates	Addition formula
BouncyCastle	KeyGen	Comb	modified	`add-1998-cmo-2`
`1.68`	Derive	Window NAF	modified	`add-1998-cmo-2`
BoringSSL	KeyGen	Fixed window	jacobian	`add-2007-bl`
`9f55d97`	Derive	Fixed window	jacobian	`add-2007-bl`
Botan	KeyGen	Fixed window	jacobian-3	`add-1998-cmo-2`
`2.18.0`	Derive	Fixed window	jacobian-3	`add-1998-cmo-2`
Crypto++	KeyGen	Sliding window	affine	textbook[1]
`8.5.0`	Derive	Sliding window	affine	textbook[1]
fastecdsa	KeyGen	Ladder	affine	textbook[1]
`2.2.1`	Derive	Ladder	affine	textbook[1]
libgcrypt	KeyGen	Basic left-to-right	jacobian	`add-1998-hnm`
`1.9.3`	Derive	Basic left-to-right	jacobian	`add-1998-hnm`
LibreSSL	KeyGen	Ladder	jacobian	`add-1998-hnm`
`3.3.3`	Derive	Ladder	jacobian	`add-1998-hnm`
libtomcrypt	KeyGen	Sliding window	jacobian	`add-1998-hnm`
`0.18.2`	Derive	Sliding window	jacobian	`add-1998-hnm`
IPP-crypto	KeyGen	Window NAF	jacobian	`add-1998-cmo-2`
`2021.2`	Derive	Window NAF	jacobian	`add-1998-cmo-2`
Microsoft CNG	KeyGen	Fixed window	jacobian	`add-2007-bl`
`6d019ce`	Derive	Fixed window	jacobian	`add-2007-bl`
NSS	KeyGen	Window NAF	jacobian	`madd-2004-hmv`
`3.65`	Derive	Window NAF	jacobian	`madd-2004-hmv`
OpenSSL	KeyGen	Ladder	xz	`mladd-2002-it-4`
`1.1.1k`	Derive	Ladder	xz	`mladd-2002-it-4`
wolfSSL	KeyGen	Sliding window	jacobian	`add-1998-hnm`
`4.7.0`	Derive	Sliding window	jacobian	`add-1998-hnm`
MatrixSSL	KeyGen	Sliding window	jacobian	`add-1998-hnm`
`4.3.0`	Derive	Sliding window	jacobian	`add-1998-hnm`
Go 1.16.4	KeyGen	Basic left-to-right	jacobian	`add-2007-bl`
`crypto/elliptic`	Derive	Basic left-to-right	jacobian	`add-2007-bl`

[1] Using textbook chord-and-tangent addition formulas.

Table 6.4: Libraries analyzed in this work, in the context of ECDH over E_W, i.e., both key generation (KeyGen) and shared secret derivation (Derive). For libraries supporting multiple choices of coordinates or formulas, we report the most generic and default setting.

snippets. Note that the P-224 curve does not have any zero-coordinate point suitable for the RPA attack, and the used formulas are complete, disallowing the possibility of an EPA attack.

We also expanded the range of scalars for which the (x)DCP can be solved. While this increases the number of recovered bits only slightly, our improvements are quite general and might be combined with future ones.

6.4.3 Tooling

We released all of our code and data under an open-source license, as an extension to the **pyecsca** project [21]. This includes tooling for unrolling EFD formulas, helping analyze exceptional cases and automatically construct ZVP points (note that Akishita and Takagi [AT03] construct them manually), as well as improvements to solving the (x)DCP (Section 6.2.3).

6.4.4 Reverse engineering

Another application of our techniques is in reverse engineering black-box implementations of ECC, as suggested in [Jan20]. Many side-channel attacks critically depend on the attacker having detailed knowledge of the target implementation, such as the scalar multiplication algorithm, coordinates, or even specific formulas used. In practice (e.g., smartcards), vendors keep this information secret; de facto security-by-obscurity.

In our unified framework, reverse engineering is an easier problem than attacking. Indeed, it suffices to choose f as an intermediate value of a point addition formula, then solve the (x)DCP problem for several small values of k. Our methodology allows us to choose f in a manner that allows us to identify the target addition formulas, after confirming one of our guesses (e.g., using $k = 1$ and $k = 2$). Furthermore, as the sequence of formula executions during scalar multiplication with a fixed scalar depends on the scalar multiplication algorithm used, we can apply our technique to identify this algorithm as well.

21. `https://github.com/crocs-muni/formula-for-disaster`

6.5 Conclusions

In this chapter, we presented a unified framework for the RPA, ZVP, and EPA attacks, and demonstrated its utility by mounting an attack on window-based scalar multiplication methods (Section 6.2.5). We were also able to push the ZVP and EPA attacks further: introducing the dependent coordinates problem, and solving it for new cases. We created automated tooling that unrolls formulas and constructs ZVP points, which was only possible manually before. We released all our code and data as an open-source extension of the **pyecsca** toolkit, with the hope that they can serve as a basis for future work.

As a result of our systematic classification, we uncovered new classes of exceptional points in EFD formulas. These formulas are, however, currently not used by any of the open-source cryptographic libraries we analyzed, which we see more as happenstance than competence – for example, OpenSSL was using `ladd-2002-it-3` not that long ago.

Lessons learned. Our Section 6.4 results demonstrate Murphy's law, in action, (sometimes) in real code, with (at least) billions of deployments. Furthermore, they highlight our failure as a research community. We know of these exceptions for over two decades, yet we are still unable to eradicate legacy theoretical constructs and code from real-world standards, products, and systems. This is exacerbated by the fact that, again as a research community, we often prioritize speed over security, in the name of establishing novelty for scientific contributions. These are often then left in dubious hands, without diligent technology transfer, and with little to no knowledge of how to apply them safely. This is precisely where our Section 6.3 results help, by providing feedback on the type and nature of failures in various EC arithmetic formulas. All of these results are enabled by our unified attack framework in Section 6.2, our showcase theoretical contribution upon which we build our practical contributions.

We believe that in order to prevent future vulnerabilities, we should start paying more attention to the properties of the formulas and their assumptions, and clearly document them in libraries, papers, and the EFD.

7 Interoperable Schnorr multisignatures[1]

Multiparty protocols are becoming increasingly popular as a method of private key compromise prevention. These approaches divide the secret key among multiple devices and never reconstruct it in a single place. Such a mechanism protects not only against malware but also against code vulnerabilities or backdoors when different implementations and devices are used. Still, an issue on the protocol level may result in compromise, and up until now, it has been unknown how exactly can different unmodified multiparty protocols be combined.

Since the expiry of Schnorr's patent in 2008 [Sch91a], Schnorr signatures are making a considerable comeback, fueling digital signature specifications like EdDSA [Ber+12] – which is being gradually incorporated in state-of-the-art protocols like TLS, SSH, Tor, and WireGuard [IANb] – and BIP-Schnorr [WNT20a], a Bitcoin community specification currently being adopted in the Bitcoin blockchain [Nak08] as a part of the Taproot [WNT20b; WNT20c] consensus update.

Adopters of Schnorr signatures benefit from their efficiency as well as linearity, which provides several usability benefits. In the blockchain scenario, they can be used to enforce contracts without the requirement of the scripting capabilities of the underlying blockchain [Poe18]. More generally, they allow for an efficient construction of multiparty signatures, indistinguishable from a single-party one.

Recent schemes [AB21; KG; Max+19; NRS; Syt+16] significantly advanced the construction of Schnorr signatures by multiple co-signing parties, decreasing communication and achieving security in the plain public-key model. However, no single scheme is suitable for all applications, and they often differ in technical or design details, typically rendering them partly or fully incompatible.

Apart from the usability benefits, interoperability also brings security improvements. In real-world cryptographic systems, implementation vulnerabilities are a frequent weak point. Multiparty computation can mitigate their impact by running different implementations on different devices [Mav+17], as long as at least one implementation

1. The results in this chapter were submitted to Applied Cryptography and Network Security 2022 as "Resilience via Practical Interoperability of Multiparty Schnorr Signature Schemes"with coauthors Antonin Dufka and Petr Svenda.

remains secure. Multiparty scheme interoperability extends this idea by an additional parameter: choosing different protocols limits the threat of common implementation errors in all protocols at once.

In this work, we study the differences between multiparty Schnorr signature schemes, classify them based on their approach to the aggregated nonce agreement, and identify technical issues that could hinder the interoperability of schemes within the same class. Furthermore, we investigate the possibility of interoperability of multiparty schemes across classes. We manage to achieve it by introducing an untrusted central party that mediates communication among different protocols and arrives at the correct signature without violating the security of any of the protocols.

Based on these insights, we design a new multiparty Schnorr scheme SHINE, which is interoperable with several other Schnorr schemes and is optimized for computationally limited devices like smartcards. Unlike recent schemes that trade additional computation for low communication complexity and unlimited-parallelism security, we reduce the on-device computation, which is the major bottleneck for smartcards. SHINE is a serial, two-round scheme, where the first round can be precomputed with a novel approach to nonce caching that avoids the Drijvers attack [Dri+19] by construction.

Our main contributions are:

- Classifying the current multiparty Schnorr signature protocols based on their approach to the nonce agreement.
- Proposing a mediation layer with an untrusted third party, bridging protocol class differences to achieve interoperability.
- Designing a novel two-round (one-round plus precomputation) multiparty Schnorr signature scheme called SHINE, which features *encrypted nonce caching* and targets computationally limited devices like smartcards.
- Providing open-source implementation of SHINE on the JavaCard platform and evaluating its performance on five different smartcard models.
- Demonstrating practical interoperability of SHINE with other scheme classes via the proposed mediation layer.

This chapter is structured as follows: after the first introductory section, Section 7.1 sums up the related work, while Section 7.2 declares the used notation and presents the relevant background. We compare different approaches to nonce agreement and discuss their comparability in Section 7.3. Building upon the lessons learned, we introduce our scheme SHINE in Section 7.4 and draw conclusions in Section 7.5.

7.1 Related work

Wegner [Weg96] defined the concept of interoperability as the ability of two or more software components to cooperate despite their implementation differences and provided two approaches for achieving it. This concept has been applied to various components of computer systems, but to the best of our knowledge, no previous work focused on the interoperability of different multiparty signature schemes.

CoSi [Syt+16] is a two-round Schnorr multi-signature scheme designed for high-speed signing by many parties organized into a tree structure. The scheme has been proven secure only for logarithmically many parallel signing instances in follow-up work [Dri+19].

Myst [Mav+17] is a setup of 240 smartcards interconnected into a grid performing multiparty computations to achieve high guarantees of backdoor tolerance. Myst uses a multi-signature scheme similar to CoSi, which was optimized for limited devices. One of the optimizations the scheme uses (nonce caching) was found vulnerable to an attack by Drijvers et al. [Dri+19]

MuSig [Max+19] was originally presented as a two-round scheme that was later found vulnerable by Drijvers et al. [Dri+19] MSDL by Boneh et al. [BDN18] added a preliminary commitment round to prevent the attack, inspired by Bellare and Neven [BN06], resulting in a three-round parallel-secure scheme.

The first parallel-secure two-round Schnorr multi-signature scheme was MuSig-DN [Nic+20], which avoids the Drijvers attack by generating the nonce deterministically. The nonce needs to be supplemented with costly non-interactive zero-knowledge proofs of its correct construction to achieve security.

MuSig2 [NRS] and DWMS [AB21] made advances in secure two-round multi-signature schemes with unlimited parallelism and, independently of each other, introduced a technique preventing the Drijvers attack. This approach is much more efficient than deterministic nonce derivation but still presents a significant computation overhead, limiting its usefulness for computationally restricted devices.

FROST [KG] is a threshold signature scheme that is secure for an arbitrary threshold $t \leq n$ and, as such, provides great flexibility to its applications. Its original version was also vulnerable to the Drijvers attack, but later version employed a variation of the technique used in [AB21; NRS] to avoid the issue.

Garillot et al. [Gar+21] presented another threshold signature scheme secure in the dishonest majority setting that has the benefit of not requiring additional randomness nor state. Its construction is conceptually similar to the MuSig-DN scheme, and it uses deterministic nonce derivation supplemented by zero-knowledge proofs.

7.2 Notation and background

A *group description* is a triplet $(\mathbb{G}, q, G)$, where q is a λ-bit prime, $\mathbb{G}$ is a cyclic group of order q and G is a selected generator of $\mathbb{G}$. We denote uniform sampling of e from a non-empty set S as $e \leftarrow_{\$} S$. Unless stated otherwise, a function $H : \{0,1\}^* \to \mathbb{Z}_q$ can take an arbitrary number of arguments that are first uniquely encoded to binary strings. We reserve k for the number of signing parties and t for the threshold needed to perform signing.[2] We call schemes that allow any $t \leq k$ *threshold signatures*, while *multi-signatures* require $t = k$. Furthermore, for a secret s, we use the following notation:

- PRF_s – a cryptographic pseudorandom function seeded with s;
- KDF_s – a cryptographic key derivation function seeded with s;
- Enc_s – a symmetric encryption function with a key s;
- Dec_s – a symmetric decryption function with a key s;
- Com – a cryptographic commitment.

2. Less than t parties cannot create the signature.

7.2.1 Schnorr signatures

The original Schnorr signature scheme [Sch91b] is derived from the Schnorr identification scheme using the Fiat-Shamir transform [FS86], and it relies on the hardness of the discrete logarithm problem. The scheme outputs efficiently computable and verifiable signatures of short length. It has been proven existentially unforgeable under the chosen message attack [PS00] in the random oracle model [BR93]. Various equivalent formulations of Schnorr signature schemes have been suggested, but in this work, we choose the one typically used[Ber+12; WNT20a], as it supports efficient batch verification and prevents related-key attacks [Mor+15].

Definition 7.2.1 (Schnorr signature)**.** Let $(\mathbb{G}, q, G)$ be a group description and $H : \mathbb{G}^2 \times \mathbb{Z}_q \to \mathbb{Z}_q$ be a hash function. A Schnorr signature of a message $m \in \mathbb{Z}_q$ verifiable with public key $X \in \mathbb{G}$ is any pair $(R, s) \in \mathbb{G} \times \mathbb{Z}_q$ satisfying the verification equation $[s]G = R + [H(R, X, m)]X$.

For a random nonce $r \in \mathbb{Z}_q$ and a private key $x \in \mathbb{Z}_q$ such that $[x]G = X$, the Schnorr signature of a message m is computed as

$$(R, s) = ([r]G, r + [H(R, X, m)]x).$$

If the challenge $H(R, X, m)$ is fixed, the signing equation is linear. This property is utilized for constructing of efficient multiparty Schnorr signature schemes. First, all k parties need to agree on a collective nonce element R which is a linear combination of their individual contributions $R_i = [r_i]G$. Subsequently, they can produce partial signature shares $s_i = r_i + H(R, X, m)x_i$, which can be summed up to obtain the resulting signature $s = \sum_{i=1}^{k} s_i$, verifiable under their aggregated public key $X = \sum_{i=1}^{k} X_i$.

The simple multi-signature scheme described in the previous paragraph has a few caveats, which cause it to be insecure in many use-cases and these issues are addressed by more complex designs. The two main security obstacles are related to the key aggregation and the nonce agreement, both of which can be attacked to perform a forgery.

The key aggregation is prone to *rogue-key attacks*, where the adversary computes his key as a function of the public keys of other parties and cancels out their contribution. To illustrate the problem, assume the attacker is the first party. He can compute his key as $X_1 = [x'_1]G - \sum_{i=2}^{k} X_i$ for some $x' \in \mathbb{Z}_q$. When this rogue key is combined with the other keys, the resulting aggregated key is $X = [x']G$ and the attacker can create signatures on behalf of the group.

Rogue-key attacks can be prevented by distributed key generation, which requires fresh key pairs [Mav+17]. Alternatively, pre-existing keys can be reused when supplemented by a proof-of-knowledge of their private key [BDN18]. Another method that supports key reuse is a non-interactive key aggregation method presented in MuSig [Max+19], which avoids the attack by unpredictably altering the aggregated key whenever any of the combined keys changes.

The other problem occurs in a parallel setting. If serial signing can be enforced, the protocol is secure, and the aggregated nonce does not even need to be computed by the signer, as is the case in CoSi [Syt+16; Dri+19]. However, if signing instances with the same key can be executed in parallel (e.g., nonce contributions are shared in advance), the Drijvers attack can achieve signature forgery. The Drijvers attack relies on solving an instance of the ROS problem [Sch01], which can be solved in subexponential [Wag02] or polynomial time [Ben+21], depending on the number of parallel sessions.

7.3 Interoperability of Schnorr schemes

Different Schnorr schemes mentioned in Section 7.1 exhibit different trade-offs. Some schemes are optimized for a low number of communication rounds; others are better suited for limited devices where the computation is costly; some use only standard operations commonly available on legacy systems or can utilize dedicated co-processors, and some need to use novel cryptographic primitives. As a result, none of the schemes is ideally suitable for all computing platforms.

In this section, we attempt to address the problem of scheme heterogeneity. We surveyed current multiparty Schnorr signature designs and classified them based on their approach to the nonce agreement. With this classification, we specify what is required of the schemes from the same class to be compatible with each other. Furthermore, we inspect the differences among the classes and bridge them using an untrusted third party[3] without any changes to the underlying schemes. If this mediation is possible, we call the schemes *interoperable*.

All of the considered schemes were proven secure in the dishonest majority setting, i.e., their security does not rely on the actions of other participants, who can only cause a signing failure by providing incorrect signature shares. Therefore, if the translated communication among different schemes achieves correctness, each participant maintains their security guarantees with respect to their protocol, and they will jointly be able to produce valid signatures.

Group Key Computation. A plethora of approaches can be used to establish the group key – the public key that represents the signing group. Some of them require an interactive fresh key generation for every new signing group [Mav+17], while some allow reusing a previously generated key in a non-interactive key aggregation [BDN18; Max+19]. Yet to achieve interoperability, the signature schemes need to use compatible key generation and aggregation methods.

The group key computations typically result in additive or Shamir secret sharing [Sha79] of the underlying secret key. The former approach is more efficient but can only be used with multi-signatures, while the latter is more general as it allows for an arbitrary $t \leq k$. However, these approaches cannot be directly combined. If some schemes rely on having Shamir secret shares, they are inherently incompatible with additive sharing schemes.

Nonce Agreement. The method of nonce agreement is the fundamental consideration in computing multiparty Schnorr signatures. All signing parties need to contribute to the nonce agreement with their fresh partial nonce, which they later reflect in signing. After the nonce is known, the partial signatures can be computed and aggregated non-interactively.

There are four main approaches to the nonce agreement: 1) nonce exchange, 2) nonce commitment, 3) nonce delinearization, and 4) deterministic nonce derivation. These methods differ in the number of communication rounds, operational assumptions, and computational complexity. We analyze them in the following subsections.

3. The party can cause signing to fail, but it cannot forge signatures.

7.3.1 Nonce exchange

The simplest approach to the nonce agreement is *nonce exchange*, where each signer i uniformly samples a random nonce $r_i \leftarrow_{\$} \mathbb{Z}_q$, computes the corresponding element $R_i = [r_i]G$, and transmits this element. The elements are then summed up into the aggregate nonce $R = \sum_{i=1}^{k} R_i$, which is used in the signing.

The simplicity of this approach comes at a cost. Signing with nonce exchange is only secure when the scheme is executed serially, i.e., no parallel signing sessions are allowed [Dri+19]. If parallel signing sessions can occur, a practical message forgery can be performed by the Drijvers attack [Dri+19]. This problem needs to be addressed by an implementation that enforces correct operation.

Nonetheless, if implemented correctly, it results in a secure and very efficient two-round signing protocol adopted by applications focusing on performance [Mav+17; Syt+16]. These applications also utilize that the security of nonce exchange does not rely on the specific construction of the aggregate nonce. Therefore, the aggregate nonce does not need to be computed by the signers, further decreasing the computational requirements.

Furthermore, since there are no constraints on the construction of the aggregate nonce, nonce exchange schemes are convenient for achieving interoperability with other nonce agreement approaches that are more restrictive but provide parallel security.

7.3.2 Nonce commitment

The Drijvers attack requires the attacker to be able to choose their nonce depending on the nonces of other parties[4]. This precondition can be broken by a preliminary communication round, in which each signer i first shares a commitment to his nonce element $\mathsf{Com}(R_i)$ and only after that reveals the actual nonce element R_i. The provided nonce elements need to be verified against the commitments, and if an inconsistency is discovered, the protocol must be aborted.

This three-round approach is called textitnonce commitment and has been used by the MuSig and MSDL [Max+19; BDN18]. It provides parallel security with only a minimal computation overhead over

4. Assuming the other parts of the challenge hash are already fixed.

the nonce exchange. If the additional communication round is not too costly, e.g., the devices are co-located, this approach is also quite efficient and suitable for computationally limited devices.

However, the requirement put on the nonce construction introduces a limitation to interoperability. To achieve interoperability with different implementations of nonce commitment, the implementations need to compute the Com using the same (typically hash) function. It is a consequence of the hiding property of the Com function, which prevents the commitment from being readjusted by a third party.

Still, nonce commitment is interoperable with serial schemes. It can be achieved via a translation layer that simulates the additional commitment round on behalf of the serial schemes in the following way: First, the nonce exchange schemes begin their nonce element sharing. The translation layer computes commitments for these elements with an appropriate Com function and simulates the commitment round with nonce commitment schemes. Afterward, the nonce commitment schemes and the translation layer (on behalf of exchange schemes) reveal the nonce elements that successfully verify against the commitments. As a result, each party arrives at the same aggregate nonce that can be used for signing.

7.3.3 Nonce delinearization

The more recent *nonce delinearization* is secure under parallel execution with just two communication rounds, the first of which can be precomputed. The main downside of this approach is its high computational cost, as it requires the signers to perform multi-scalar multiplication in the second round. Nonetheless, the practical benefits often outweigh the cost, and this technique has been used in the design of multiple schemes [AB21; KG; NRS].

With nonce delinearization, the aggregate nonce is constructed as a linear combination of ν pre-nonces, where the coefficients are non-linearly dependent on the pre-nonces and the signed message via a cryptographic hash function. Consequently, any change to the pre-nonces or the signed message results in an unpredictable non-linear change of the aggregate nonce, which thwarts the Drijvers attack.

When this process is applied to multiparty schemes, each signer i independently generates partial pre-nonces $r_{i,1}, \ldots, r_{i,\nu}$ and transmits the corresponding elements $R_{i,1} = r_{i,1}G, \ldots, R_{i,\nu} = r_{i,\nu}G$. Once the signed message m is known, the pre-nonces are aggregated into $R = \sum_{i=1}^{n} \sum_{j=1}^{\nu} \beta_{i,j} R_{i,j}$, where $\beta_{i,j}$ are the delinearization coefficients.

The actual schemes differ in the choice of $\beta_{i,j}$. DWMS [AB21] computes a different coefficient for each partial pre-nonce: $\beta_{i,j} = H(i, j, R_{i,1}, \ldots, R_{n,\nu}, m)$. Alternatively, in FROST [KG] and MuSig2 [NRS], $\beta_{i,j}$ does not depend on i, allowing parties to compute aggregated pre-nonces $R_j = \sum_{i=1}^{n} R_{i,j}$ in advance – thus avoiding some multiplications when computing the aggregate nonce as $R = \sum_{j=1}^{\nu} [\beta_{1,j}] R_j$. In such cases, the coefficients $\beta_{1,j}$ can be, for example, chosen as $H(j, R_1, \ldots, R_\nu, m)$ or as powers $H(R_1, \ldots, R_\nu, m)^{j-1}$ [NRS].

The specific nonce aggregation used by delinearization schemes introduces severe limitations to interoperability. It requires the schemes to be able to construct the nonce non-interactively, based on the initial pre-nonces and the signed message. Thus the only way of achieving interoperability among different implementations of nonce delinearization schemes is to compute the same coefficients $\beta_{i,j}$ and use the same number of pre-nonces ν. Otherwise, if the schemes use, e.g., a different hash function in the coefficient computation, they do not arrive at the same aggregated nonce.

Interoperability of nonce delinearization with schemes of other classes is a bit more nuanced and cannot be achieved in general. Equation (7.1) shows a signature produced by a nonce exchange scheme, while Equation (7.2) displays a signature produced by a nonce delinearization scheme ($\nu = 2$ for brevity).

$$s = r_{i,1} + e x_i \tag{7.1}$$

$$s = \beta_{i,1} r_{i,1} + \beta_{i,2} r_{i,2} + e x_i \tag{7.2}$$

The difference between (7.1) and (7.2) cannot be reconciled by an intermediating party because it cannot multiply the nonce $r_{i,1}$ without also changing the $e x_i$ component. However, if $\beta_{i,1} = 1$, the intermediating party can simulate the second pre-nonce on behalf of the nonce exchange schemes and add $\beta_{i,2} r_{i,2}$ to the produced partial signature. This results in a valid signature contribution, without any change to the underlying scheme.

Such preconditions occur in FROST [KG] and have also been suggested as an optimization of MuSig2 [NRS] that became the default choice in a later revision of the scheme. We call this variation, where the first pre-nonce is not multiplied by the coefficient, as *half-nonce delinearization*. This variant is not a mere performance optimization (as presented in the original paper), but importantly, it also enables interoperability with nonce exchange schemes, as illustrated above.

The simulation can then proceed in the following way. First, signers begin by sharing their nonces or pre-nonces. The single nonce provided by nonce exchange schemes is used as their first pre-nonce, and the coordinator computes the other pre-nonces on their behalf. These simulated pre-nonces can be sent to the delinearization schemes that can now compute the aggregated nonce. At this point, the aggregated nonce is also computed by the intermediating party that provides it to the nonce exchange schemes, which reply with their partial signatures $s_i = r_{i,1} + ex_i$. The intermediating party augments the partial signatures by the simulated pre-nonces $s_i + \beta_{i,2}r_{i,2} + \cdots + \beta_{i,\nu}r_{i,\nu}$, making them compatible with signatures of nonce delinearization schemes.

For this reason, we suggest preferring schemes with half-nonce delinearization, i.e., schemes where the first pre-nonce is not multiplied by a coefficient. This choice still allows adding arbitrarily many pre-nonces to tweak the difficulty of the underlying problem[5] while remaining interoperable with nonce exchange schemes and incurring no additional cost to them.

Achieving interoperability is not possible with nonce commitment schemes, as those schemes need to receive a commitment to the single nonce produced by each party before revealing their own nonce, but nonce delinearization schemes use multiple nonces and cannot combine them before all other nonces are known. If the nonce commitment scheme was changed to cover the additional pre-nonces so that each signer can derive the delinearization coefficients itself, it would collapse into a nonce delinearization scheme with a redundant commitment round.

7.3.4 Deterministic nonce derivation

Another approach to parallel-secure two-round scheme construction is *deterministic nonce derivation*. Deterministic nonce derivation has been

5. A MuSig2 variant proposed four pre-nonces under weaker security assumptions.

used in standard signature schemes, preventing attacks due to biased randomness in the nonce generation. However, these approaches are not directly applicable to the multiparty setting, as a malicious party that diverges from the correct computation could force an honest signer to reuse a nonce, which would result in a key compromise [Nic+20].

MuSig-DN [Nic+20] solves this problem using non-interactive zero-knowledge proofs. In the scheme, the participants derive the nonce using an algebraic pseudorandom function provided the signed message and public keys of other participants and compute a proof of the correct construction. This proof is provided along with the nonce, and other protocol participants need to verify it. If the verification fails, the signing must be aborted.

The described construction does not suffer from the problems of insufficiently random nonces. However, it requires the computation of a zero-knowledge proof, which is relatively costly even for regular devices, let alone for smartcards.

The need for the zero-knowledge proof also severely limits the interoperability of the scheme. Different implementations of deterministic nonce derivation schemes require a consensus on the used derivation function and the proof construction to work together. Interoperability with schemes using a different approach to the nonce agreement is not possible, as it would break the security assumption of the deterministic nonce derivation.

7.3.5 Summary

Table 7.1 displays the interoperability matrix of different schemes. Nonce exchange schemes stand out among others as the most flexible ones due to their ability to accept an externally provided nonce without any knowledge of its construction and remain secure with a sequential execution. Interactions of nonce exchange and nonce delinearization schemes are denoted with a question mark, as they are interoperable only if the half-nonce delinearization method is used. Deterministic nonce derivation is compatible only with itself.

Nonce exchange schemes are interoperable with nonce commitment schemes and half-nonce delinearization schemes separately. Yet nonce commitment schemes and nonce delinearization schemes are not interoperable as they have contradictory requirements on the construction of the nonce. Therefore, non-interactive setups including schemes using both nonce commitment and nonce delinearization are not possible.

	Nonce exchange	Nonce commitment	Nonce delinearization	Deterministic nonce
Nonce exchange	✓	✓	?	
Nonce commitment	✓	✓		
Nonce delinearization	?		✓	
Deterministic nonce				✓

Table 7.1: Interoperability of nonce agreement approaches.

Lastly, even though the schemes are interoperable in theory, several technical details need to be addressed in practice. The schemes need to produce compatible signatures – not only Schnorr signatures but the same instance of Schnorr signatures, e.g., Ed25519 [Ber+12] or BIP-Schnorr [WNT20a]. That requires a precise specification of the used elliptic curve, point encoding, and hash function(s). If this requirement is satisfied, the interoperability discussed in this section is achievable.

7.4 Multiparty scheme SHINE

This section describes SHINE (Smartcard Highly-Interoperable Nonce Encryption scheme) – our multiparty Schnorr signature scheme optimized for computations on cryptographic smartcards while being interoperable with the majority of pre-existing Schnorr multiparty schemes. The design relies on a central party that is trusted only to mediate communication among individual signers.[6] We utilize this

6. Violating this trust does not endanger shares of the secret held by honest signers.

central party for the precomputation of inputs, data storage, and also as a translation layer for achieving interoperability, as described in Section 7.3.

SHINE can create a signature in two communication rounds, the first of which can be securely precomputed. The scheme uses a variant of nonce exchange that enables interoperability with all classes except deterministic nonce derivation. But it also requires serial execution to be secure, which we enforce by design. Additionally, to avoid randomness generation failure attacks and minimize storage requirements, we derive nonce using a secret pseudorandom function that depends on an internal counter.

7.4.1 Attacker Model

We assume that the attacker is able to control the central party and at most $t - 1$ of the signing parties. Since the central party is under the control of the attacker, she can drop or alter messages, and as a result, cause a denial of service. We also make the standard assumption that the attacker's computational power is bound by a polynomial.

Additionally, we assume that correct group establishment was verified by secondary channels, and thus the attacker could not exclude parties from the group establishment. In practice, this could be achieved by querying each of the smartcards for its public key by different readers, verifying that each of them obtained the same value, aggregating the public keys, comparing the result against the reported group key.

7.4.2 Group establishment

Before SHINE can be used to sign messages, the signing group needs to be established. The signing group consists of a set of signers, who need to generate their private key shares, compute the group key, and initialize context information. In this process, SHINE relies on the central party to include all participants in the key generation, i.e., to not ignore or simulate messages by some parties. After the key generation, the validity of this assumption can be externally verified by querying individual participants for their group key contribution.

By default, SHINE uses a three-round distributed key generation protocol, which avoids the rogue-key attack by committing to key contributions (for a detailed description, see Figure E.21 in Appendix E.1). We chose this approach as it trades computation in favor of communication suitable for the smartcard environment. However, if another method suits the particular application better, it can be used instead. For example, non-interactive key aggregation of MuSig [Max+19] can be selected to achieve interoperability with schemes that use it.

Different type of key generation protocols needs to be used to support threshold signing, i.e., protocols resulting in Shamir secret sharing of the group key. This secret sharing can be obtained using the protocol of Komlo et al. [KG], though it relies on secure channels between the parties and is quite costly, as it requires $\mathcal{O}(kt)$ group operations by signers.

After a new group key is successfully computed, each party i finalizes the group establishment process by initializing its signing context. Each signer stores in its internal state a λ-bit secret p_i that is used for deterministic nonce derivation and an increase-only counter c_i that tracks the index of the most recently used nonce. The former value is uniformly sampled, and the latter is initially set to zero. These values are later used in the signing protocol.

7.4.3 Nonce caching with encryption

Mavroudis et al. [Mav+17] introduced nonce caching for a smartcard setup. The technique optimizes the signing speed by generating nonces in advance and storing the corresponding elements on a central server. This way, the expensive scalar multiplication can be performed during down-time when there are no signing requests, and the precomputed nonces can be provided to the signers when needed.

However, this approach turned out to be vulnerable to the Drijvers attack [Dri+19]. Since the benefits of nonce caching for smartcards are meaningful, we designed a technique that we call *nonce caching with encryption* or *nonce encryption* for short, which avoids the Drijvers attack and still allows for nonce caching with some restrictions.

The vulnerability to the Drijvers attack occurs only if the scheme is used in a parallel, i.e., an adversary can open multiple signing sessions in parallel or cache multiple nonces before they are used for signing.

Algorithm 5: Cache

```
Input: Session index $j$
Output: Encrypted nonce $E_{i,j}$
$r_{i,j} := \mathsf{PRF}_{p_i}(j)$
$R_{i,j} := [r_{i,j}]G$
$K_{i,j} := \mathsf{KDF}_{p_i}(c)$
$E_{i,j} := \mathsf{Enc}_{K_{i,j}}(R_{i,j})$
return $E_{i,j}$
```

Algorithm 6: Reveal

```
Input: Session index $j$
Output: Decryption key $K_{i,j}$
if $j \geq c_i$ then
    $c_i := j$
$K_{i,j} := \mathsf{KDF}_{p_i}(c_i)$
return $K_{i,j}$
```

Figure 7.1: SHINE nonce caching and revealing algorithms.

Nonce encryption leverages this property and enforces serial execution to ensure at most a single cached nonce is revealed at a time without having a signature created with it.

Nonce encryption works in two phases – *cache* and *reveal*, shown on Figure 7.1. During the caching phase, the smartcard computes the nonce element and sends it encrypted to the central party. The reveal phase is then used to reveal only a single nonce element at a time.

The cache phase is initiated by the central party, who sends a signing instance identifier j. The smartcard i uses a pseudorandom function, keyed with a secret p_i (generated during the group establishment), to derive a nonce $r_{i,j}$ corresponding to the signing instance j. Subsequently, the smartcard computes the nonce element $R_{i,j} = [r_{i,j}]G$. This computation is the demanding operation that would be the bottleneck during the signing.

In the standard nonce caching, $R_{i,j}$ would be transmitted to the central party; however, if this step was repeated, it would lead to the vulnerability to the Drijvers attack. With nonce encryption, the nonce is not transmitted in plaintext but rather encrypted with symmetric key $K_{i,j}$ derived from the signing instance identifier j using a key derivation function known only to the signer i. The central party stores the received value for future use.

The *reveal* phase uses the smartcard internal increase-only counter c_i to track already revealed nonce elements. When prompted by the central party to send $K_{i,j}$ for a signing instance $j \geq c$, the smartcard derives $K_{i,j}$ and sends it back, but at the same time increases its inner counter $c_i := j$. With the knowledge of $K_{i,j}$, the central party can decrypt the cached nonce $R_{i,j}$, possibly combine it with nonces of other parties, and use it in a subsequent signing.

So far, the counter c_i did not limit the attacker in any way. It plays a role only during the signing, where the smartcard must not produce a signature for a signing instance $j < c_i$. When combined, these restrictions ensure that only the signing instance $j = c_i$ can succeed since nonces for $j > c_i$ are not known yet, and a signature for $j < c_i$ will not be created. The technique already enforces the signing of only monotonous sequences, but to achieve serial signing and thus security, only strictly increasing sequences can be allowed; otherwise, the key would be extractable by reusing the same nonce. Therefore, the smartcard must increase the internal counter $c_i := c_i + 1$ when producing a signature.

Comparison to Plain Nonce Caching. Just as in plain nonce caching, we manage to avoid costly group operations during the signing. In this subsection, we highlight the differences.

Storage-wise, SHINE still requires only constant memory on smartcards, but the central party cannot pre-aggregate nonces anymore as only a single nonce is known at a time. Therefore, the space required by the central party grows linearly in the number of signing parties. However, this is not necessarily a disadvantage because when the nonces are not aggregated in advance, they can be adapted to an arbitrary signing subset. For example, if a smartcard malfunctioned and needed to be removed from the signing set, all cached nonces would need to be discarded. But without the aggregation, it suffices to delete the nonces corresponding to the malfunctioning smartcard.

Communication-wise, SHINE seemingly needs one additional round to transmit the decryption key. However, this transmission can be automatically piggybacked with the previous signing, where the decryption key for $c_i + 1$ can be revealed, as c_i was already invalidated. As a result, the number of communication rounds can remain the same; only the additional decryption key is transmitted.

Finally, computation-wise, the symmetric encryption can be realized efficiently so that the additional demands on the smartcards are minimal. The only introduced concern is the aggregate nonce computation by the central party. The aggregate nonces cannot be precomputed, as only a single decryption key is known at a time. In the case of burst signing requests, the overall solution would result in a performance decrease compared to the vulnerable nonce caching. But assuming the central party is significantly more capable than the smartcards (which in practice is), the aggregation can be performed relatively quickly and is not a limiting factor.

7.4.4 Signing protocol

In Figure 7.2, we describe the signing protocol that uses encrypted nonce caching together with the piggybacking of the decryption key with the signature output.

Given a message m to be signed, the central party initiates the signing phase by computing the aggregated nonce. If it already has a cached nonce with the appropriate decryption key for each participant, it can sum them as $R \leftarrow \sum_{i=1}^{k} R_{i,c_i}$. Otherwise, the central party might need to exchange an additional `cache` or `reveal` message with some smartcards prior to the aggregation. When the aggregated nonce is computed, the central party sends a signature request to every participating smartcard with the given signing index (which may vary among the smartcards), the aggregated nonce R, and the message m.

When a smartcard receives a signing request, it first checks whether the signing index is greater or equal to its internal counter. If not, it replies with an abort message and its internal counter value, and aborts the protocol. Otherwise, the smartcard sets its internal counter $c_i \leftarrow j + 1$ and continues with the signing. It derives its nonce $r_{i,j}$ and computes its partial signature $s_{i,j} := r_{i,j} + H(R, X, m)x_i$.[7] Once that is done, the smartcard also derives the decryption key for the next cached nonce and transmits these values to the central party.

Finally, the central party can sum the partial signatures to obtain the resulting signature (R, s). If the partial nonces for the next signing

7. If the key shares x_i are Shamir secret shares and not additive shares, they first need to be multiplied by a Lagrange coefficient corresponding to the signing party in the signing subset.

Algorithm 7: Sign

```
Input: Aggregate nonce $R$, message $m$, signing index $j$
Output: Partial signature $s_{i,j}$, decryption key $K_{i,j+1}$ for $j+1$-th
        nonce
if $j < c_i$ then
    abort
$c_i := j + 1$
$r_{i,j} := \mathsf{PRF}_{p_i}(j)$
$s_{i,j} := r_{i,j} + H(R, X, m)x_i$
$K_{i,c_i} := \mathsf{KDF}_{p_i}(c_i)$
return $(s_{i,j}, K_{i,c_i})$
```

Figure 7.2: SHINE signing algorithm.

round have been already cached, the central party can decrypt them with the provided decryption keys.

Security of SHINE. Drijvers et al. [Dri+19] have shown that the CoSi scheme [Syt+16] is secure in the random oracle model (ROM) [BR93], assuming the one-more discrete logarithm (OMDL) problem [Bel+03] is hard, parallel signing instances do not occur, the number of signers is bound by a polynomial, and the signers committed to their keys in the group key computation. We claim that all of these prerequisites also hold in SHINE and that the security of SHINE is the same as of CoSi, as the scheme differences do not affect the security proof.

One of the differences between CoSi and SHINE is the used topology. CoSi considers tree topology with a layered nonce and signature aggregation to minimize communication, whereas SHINE relies on the star topology to facilitate interoperability mediation. The star topology can be considered a special case of the tree topology, where the central party is the root, and the other parties are its children; thus, this version is also covered by the CoSi security proof.

Another difference is in the nonce generation process. SHINE uses a PRF to deterministically derive nonces based on a random secret seed, whereas CoSi uses fresh randomness for each nonce. We model the PRF as a random oracle, so as long as the same input to the function is never reused in signing, the generated nonces are indistinguishable by the adversary from the nonces produced by CoSi. SHINE prevents the PRF input reuse by the increase-only counter c_i.

The last difference between the schemes is the technique of nonce caching with encryption featured in SHINE. In the ROM, the encrypted precomputed nonces sent to the central party convey no information unless the central party knows the key $K_{i,j}$. Thus until $K_{i,j}$ is transmitted, the central party learns nothing about the nonces, just as if they were computed on the fly during the signing, as in CoSi.

7.4.5 Implementation and evaluation

We implemented SHINE for the JavaCard platform using a modified version of the JCMathLib library [MS20] that provides the necessary low-level operations without relying on proprietary smartcard API and thus enables our results to be reproduced on a wide variety of supported smartcards. While the solution achieves a decent speed with JCMathLib, using proprietary API calls will significantly increase its performance in practice, especially for the `GET_NONCE` and `CACHE_NONCE` operations.

We tested and evaluated our implementation on five JavaCards: `NXP J2E145G` (1), `NXP J3H145` (2), `GD SmartCafe 6.0` (3), `GD SmartCafe 7.0` (4), and `NXP J3R180` (5). We measured the time required to compute each phase of the protocol and also their counterparts in a variant without nonce caching so that we would be able to assess its impact. We repeated each measurement 100 times and averaged the results. The summary of the results is presented in Table 7.2.

The implementation achieves an average signing speed of around 700 ms, comparable to Myst [Mav+17]. The signing slowdown caused by key derivation for piggybacking did not exceed 45 ms for any of the cards and thus increased signing latency by at most 6%. Encryption operation added to the nonce caching resulted in less than 36 ms slowdown, which was at most 23% (but mostly only 1%) of the time required to perform the nonce computation.

	Card 1	Card 2	Card 3	Card 4	Card 5
`GET_NONCE`	2826	194	2764	1962	60
`CACHE_NONCE`	2854	217	2801	1984	74
Overhead	28	24	36	23	14
Overhead (%)	1	12	1	1	23
`SIGN`	802	737	768	637	457
`SIGN_REVEAL`	842	756	813	660	472
Overhead	28	19	45	23	15
Overhead (%)	5	3	6	4	3
Signing w/o caching	3627	931	3532	2599	518
Signing w/ caching	842	756	813	660	472
Speedup	2785	175	2719	1938	46
Speedup (%)	77	19	77	75	9

Table 7.2: Time (ms) to compute steps of SHINE on different JavaCards.

The differences in time required to compute the nonce are caused by different native algorithm support. For example, cards 2 and 5 supported `ALG_EC_SVDP_DH_PLAIN_XY` algorithm, which made the nonce computation quite close to the native performance of the hardware, while on other cards, additional adjustments in code had to be made. This computation is the precomputed part, and therefore it influences the overall speedup over signing without caching the most.

We estimate that implementing the signing part using native low-level operations instead of slower software emulation via JCMathLib would achieve performance comparable to ECDSA on a given platform (e.g., around 200 ms for common smartcards [Dzu+17]). Nonce caching with encryption would provide an overall speedup of around 50% in such implementations.

Our implementation of SHINE uses a central party written in Rust, which mediates communication among different devices and serves as storage for cached nonces. It includes nonce exchange, nonce commitment, and nonce delinearization schemes, usable by themselves or jointly with the SHINE applet, demonstrating its interoperability. Both implementations are available at GitHub repositories.[8] [9]

8. `https://github.com/dufkan/SHINE`
9. `https://github.com/dufkan/mpcd`

7.5 Conclusions

We classified the existing schemes based on their approach to a nonce agreement into four categories We discovered that the most interoperable schemes utilize nonce exchange, which is also the least computationally demanding, as a consequence of its ability to accept an externally provided nonce without any knowledge of its construction.

In the case of nonce commitment schemes, an untrusted central party can mediate the interoperability with nonce exchange schemes. The central party in this setup needs only to simulate the commitment round on behalf of the nonce exchange signers.

The situation is more complex with nonce delinearization schemes, which are not interoperable with others schemes in general. However, when the half-nonce delinearization is used, the interoperability with nonce exchange schemes can be fully mediated via a central party that simulates the delinearized nonce contributions on their behalf. Thus half-nonce delinearization is not merely an optimization; it also has interoperability benefits.

Deterministic nonce derivation schemes need to receive additional data about the nonce construction, i.e., the proof of its correctness, that cannot be simulated externally. Therefore, it seems that these schemes can only be combined with their own instances and are not interoperable with any other approach.

Based on these observations, we propose to focus on three designs of Schnorr schemes:

1. The serially constrained nonce exchange schemes for computationally restricted devices, which benefit the most from the more efficient computation;

2. the half-nonce delinearization schemes for devices where the additional computations are not a limiting factor, and that could benefit from interoperability;

3. the deterministic nonce derivation for cases where the performance is not an issue, deterministic nonces are crucial, and a setup of homogeneous implementations can be guaranteed.

We followed up on our first recommendation and designed the smartcard-optimized SHINE scheme, which uses nonce exchange for signing and thus is interoperable with other approaches.

To ensure serial signing and avoid nonce reuse, we introduced an internal increase-only counter to the design, which is used to derive nonces and ensures that for any given nonce, at most, one signature is produced. This approach is complemented by nonce caching with encryption, a novel optimization technique that allows precomputing and sharing nonces in advance while avoiding the Drijvers attack, which threatened previous attempts at nonce caching.

We have implemented SHINE as an applet for the JavaCard platform and evaluated its performance on five smartcards. The performance measurement empirically confirms the benefits of nonce caching on computationally restricted devices. Furthermore, we provide a Rust implementation of the central party that practically demonstrates the interoperability of SHINE with nonce commitment and nonce delinearization schemes.

8 DiSSECT: Distinguisher of Standard & Simulated Elliptic Curves via Traits[1]

8.1 Introduction

Properly selecting elliptic curves suitable for cryptographic applications is a difficult task and many standards do not tackle it very well [ANSI98; Cer10; ANS14; PLK06; SSL14]. Sometimes, their parameter choice is not explained at all; in other cases, it is still unsatisfactory. This was already criticized by Scott [Sco99], who introduced the notion of rigidity: the origin of all constants in cryptographic standards should be clearly explained, otherwise there can be a hidden vulnerability. Many cryptographers agree [Sch13; Loc+; Ber+15; CLN15], even if no major security problems have been found so far.

In the light of the documented instance of a standard being manipulated – the Dual EC DRBG incident [Hal13; BLN16; Che+16], this is a potentially serious matter, considering that the popular NIST curves have also been chosen by the NSA. Bernstein et al. [Ber+15] build upon Scott's idea, showing how to insert backdoors to certain standards, assuming enough weak curves exist, though Koblitz and Menezes [KM16] dispute this assumption.

Even if newer, more rigidly generated curves like Curve25519 [Ber06], Ed448-Goldilocks [Hamb] or NUMS curves [Bla+14] are on the rise, the old curves still remain in wide use [Val+18]. Moreover, Lochter et al. [Loc+] argue that *"perfect rigidity, i.e., defining a process that is accepted as completely transparent and traceable by everyone, seems to be impossible"*. Thus, we believe a thorough wide-scale analysis of the standard curves is important to re-establish the trust, especially if the standards are not transparent enough.

Our main idea is that if a weakness is present in a curve, it will manifest via a statistical deviation of some property compared to many similar pseudorandom curves. We design trait functions that cover a wide range of such properties.

1. The results in this chapter were submitted to Public Key Cryptography 2022 as "DiSSECT: Distinguisher of Standard & Simulated Elliptic Curves via Traits" with coauthors Vojtech Suchanek and Antonin Dufka.

Contributions. Our work includes the following contributions:

- We assemble the first public database of all standard[2] curves (to the best of our knowledge), with comprehensive parameter details and visualisation.
- We develop an open-source extensible framework[3] for generating curves according to known standards.
- We propose a large number of trait functions and offer a methodology for distinguishing the standard curves from the simulated ones on a large scale.
- By a systematic analysis of standard curves, we rule out certain types of problems, potentially raising the level of trust in most of them.
- We find properties of the standardized GOST curves [PLK06; Fed16] that are inconsistent with the claimed [ANS18] generation method and present unreported properties of the BLS12-381 [BLS] curve.

This chapter is organized as follows: In Section 8.2, we briefly survey the major elliptic curve standards. Section 8.3 introduces our database and explains our methodology for simulating and distinguishing curves. Section 8.4 gives an overview of our proposed traits and presents the findings of the outlier detection. We report on the technicalities of our tool in Section 8.5 and draw conclusions in Section 8.6. Finally, Appendix F.1 describes all our traits in more detail, and Appendix F.2 lists all standard curves in our database.

8.2 Overview of standard curve generation

Even though pairing-friendly curves and curves over binary and extension fields have found some applications, we focus mainly[4] on the

2. In this work, we use the adjective "standardized" for curves appearing in a standard, and the adjective "standard" for widely used curves (that are/were a de facto standard).
3. `https://dissect.crocs.fi.muni.cz/`
4. However, our database contains binary, extension and pairing-friendly curves as well, see Appendix F.2.

prevalent category of prime field curves that are recommended for ECDLP-based cryptosystems. We include Montgomery and (twisted) Edwards curves as well, though for unification purposes, we convert them to the short Weierstrass form, which is universal.

The main approach for finding a curve that satisfies the public security criteria (see Section 2.2), including a sufficiently large CM discriminant, is repeatedly selecting curve parameters until the desieed conditions hold. The main bottleneck here is point-counting: we can use the polynomial SEA algorithm [Sch95], 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 [Sol11] or Montgomery-friendly primes [CLN15]. When the field and the curve form are fixed, it remains to pick the two[5] coefficients, though one of them is often fixed.

In this section, 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 with 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.

Unknown or ambiguous origin. The following is a list of curves we analyze and whose method of generations 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 [Jou11]. The document does not specify any method of generation.

- The Chinese SM2 standard [SSL14] was published in 2010 [DY15] by the Office of State Commercial Cryptography Administration (OSCCA). The standard recommends one 256-bit curve but does not specify the generation method.

5. At least for short Weierstrass, Montgomery and twisted Edwards curves.

- In addition to the Russian GOST R 34.10 standard from 2001, six standardized curves were provided in [PLK06] and [Fed16]. Again, no method of generation was specified.

- The Wireless Application Protocol Wireless Transport Layer Security Specification [Wir00] 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) [Cer] recommends so-called Koblitz curves[6], 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 Weierstrass a, b coefficients in a so-called *verifiably pseudorandom* way, as Figure 8.1 demonstrates.

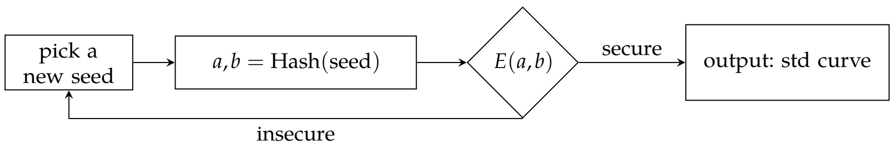

Figure 8.1: A simplified template for generating verifiably pseudorandom curves over fixed $\mathbb{F}_p$.

The details differ and we study them more closely in Section 8.3.2. The common idea is that the seed is made public, which limits the curve designer's freedom to manipulate the curve. However, Bernstein et al. [Ber+15] 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.

6. Analyzing these curves is of great importance due to the usage of `secp256k1` in Bitcoin [Nak08].

In the rest of this work, 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 titles "ECC Brainpool Standard Curves and Curve Generation" [Bra10] 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 was in FIPS 186-2 in 2000 [NIST00]. However, the used generation method already appeared in ANSI X9.62 in 1998 [ANSI98]. Other standards recommend these curves (e.g. SECG [Cer] or WTLS [Wir00]).

"The Million dollar curve"[Bai+a] took another approach to the verifiably pseudorandom strateg, proposing a new source of public entropy for the seeds, combining lotteries from several different countries.

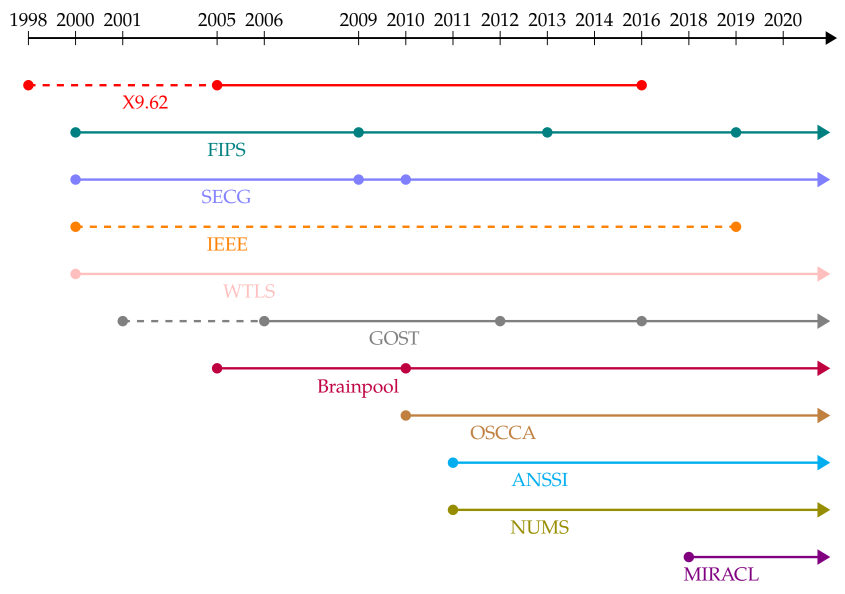

Figure 8.2: Timeline of standardization of publicly available prime-field curves. Dashed line indicates that the source does not specify curve parameters. Individual curves (e.g., Curve25519) are omitted.

Other rigid methods.

As a reaction to the NIST standardized 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. [Bla+14] as an Internet draft called "Elliptic Curve Cryptography (ECC) Nothing Up My Sleeve (NUMS) Curves and Curve Generation" [Bla+14]. We analyze the generation method further in this work.

- MIRACL library [MIR18] 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 [Ber06] 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. [Ber+13] 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, Chuengsatiansup, and Lange [BCL], Ed448-Goldilocks by Hamburg [Hama], E-3363 by Scott [Sco15] or M, E curves by Aranha1 et al. [Ara+].

8.3 Methodology

Our ultimate goal would be assessing the true security of standard curves. Yet since it is not clear how to look for unknown curve vulnerabilities, we instead aim to pinpoint possible problems via identifying standard curves that differ from other curves in specific aspects.

To find these deviations, we mimic the generation process of the standards to create a large set of simulated curves. 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, employing one of the following strategies (complementing manual inspection with automatic outlier detection):

1. **Compare curves from a standard to corresponding simulated ones.** This is the most natural strategy. We are looking for a standard curve achieving a value that almost no other simulated curve achieved.
2. **Compare curves from a standard to Pseudorandom[7] 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 standard and simulated curves. The second strategy tries to detect any sort of unexpected behaviour by comparing curves from a given standard to our *Pseudorandom* curves.
3. **Compare curves from all standardized curves to simulated curves of a single standard.** Unlike the previous two, this strategy is designed to find differences across standards. Even though not all of the comparisons are fair, the strategy still revealed relevant results.

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 return numerical results. We run these traits on standard curves as well as simulated ones whenever it is computationally feasible. Once we compute the trait results, we divide them by curve bitlengths and optionally apply transformations to keep the set of measured values comparable.

8.3.1 Standard curve database

DiSSECT is, to the best of our knowledge, the first public database of all standard elliptic curves, featuring 18 curve categories according to their source.

7. In the remainder of this work, the word Pseudorandom with capital P will refer to the curves we simulated by our own method.

The database includes:

- verifiably pseudorandom curves (X9.62, NIST, SEC, Brainpool);
- pairing-friendly curves: Barreto-Lynn-Scott [BLS], Barreto-Naehrig [Per+], Miyaji-Nakabayashi-Takano [MNT01]);
- amicable curves: Tweedledee/Tweedledum [BGH], Pallas/Vesta [Hop20];
- rigidly generated NUMS curves and curves from the MIRACL library [MIR18];
- Bernstein's high performance curves [Ber06] and M, E curves [Ara+];
- curves from the standards ANSSI [Jou11], OSCCA [SSL14], GOST [PLK06; Fed16], OAKLEY [Orm98], ISO/IEC [ISO17], WTLS [Wir00] 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 which 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 four categories of simulated curves:

- $\text{X9.62}_{\text{sim}}$;
- $\text{Brainpool}_{\text{sim}}$;
- NUMS_{sim};
- Pseudorandom[8].

For each curve, we also precomputed usual properties such as the CM discriminant, the j-invariant, the trace of Frobenius t, and the embedding degree. This precomputation significantly speeds up any analysis on these curves.

8. The set of Pseudorandom curves and their trait results is currently still evolving.

Source	#	Source	#
X9.62	40	BARP	6
Brainpool	14	BLS	6
NUMS	24	GOST	9
SECG	33	ISO	4
NIST	15	MNT	10
MIRACL	8	OAKLEY	2
X9.62$_{sim}$	120k	OSCCA	1
Brainpool$_{sim}$	12k	WTLS	8
NUMS$_{sim}$	1.2k	BN	16
AMIC	4	DJB	10
ANSSI	1	other	11
Pseudorandom	250k		

Table 8.1: Numbers of elliptic curves in our database grouped by their source.

8.3.2 Simulations

We have picked three major standards X9.62 [ANSI98], Brainpool [Bra10] and NUMS [Bla+14] 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 generated over 120 000 simulated X9.62 curves, 12 000 Brainpool curves and 1 200 NUMS curves[9].

The aim of this part is not to give 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. Although both NUMS and Brainpool 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 [ANSI98]. 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:

9. This took up to a week per standard on 40-core cluster of Intel Xeon Gold 5218.

- "Near-primality": The curve order shall have a prime factor l 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 of Frobenius t shall not be equal to 1.

Given a seed and a prime p, the standard computes a $\log(p)$-bit integer r using[10] the function (8.1). 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.

$$H(\text{seed}) = \underbrace{\texttt{SHA-1}(\text{s}}_{\text{discard}}\underbrace{\text{eed})||\texttt{SHA-1}(\text{seed}+1)||\ldots||\texttt{SHA-1}(\text{seed}+i)}_{\log(p)-\text{bit integer as output}}. \tag{8.1}$$

X9.62 – our approach. For each of the bit-lengths 128, 160, 192, 224 and 256, we have fixed the same prime as the standard (hence all curves of the same bit-length are defined over the same field). Since the standard offers no guidance 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 [CC86]) 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 notably accelerated the point counting. This choice also agrees with the fact that the standardized curves all have the 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 [Sch95]. For each of the five bit-lengths we have tried 5 million seeds, resulting in over 120 000 elliptic curves. Figure 8.3 captures a simplified overview of the algorithm.

10. More precisely, H also changes the most significant bit of the output to 0.

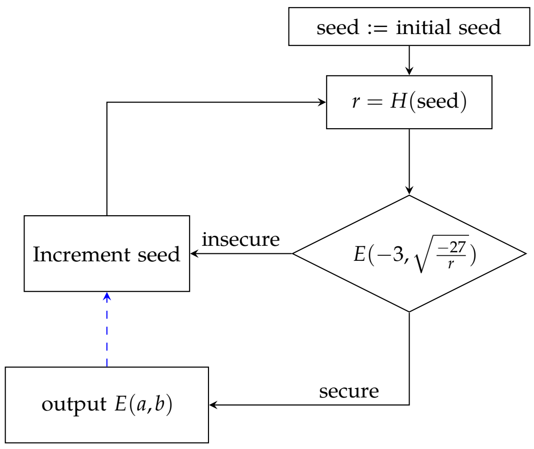

Figure 8.3: X9.62 algorithm adjusted (indicated by dashed line) for large-scale generation.

Brainpool – the standard. The Brainpool standard proposes an algorithm for generating both the prime p and the curve over $\mathbb{F}_p$. Since in our 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 generation process outputs an elliptic curve in Weierstrass form satisfying the following security conditions:

- The cofactor shall be 1, i.e., the group order n shall be prime.
- The embedding degree shall be greater than $(n-1)/100$.
- The trace of Frobenius t shall not be equal to 1. Technical requirements then state that $t > 1$.
- The class number 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 8.4. This time, the H function (8.1) 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 initial seed. See the standard for details of generation, GenA and GenB represent generation of a and b in Figure 8.4.

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 tries, after each fail attempt to find an appropriate parameter the seed is incremented. 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 drop in the proportion of generated curves compared to X9.62 standard is caused by stricted 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 [BL].

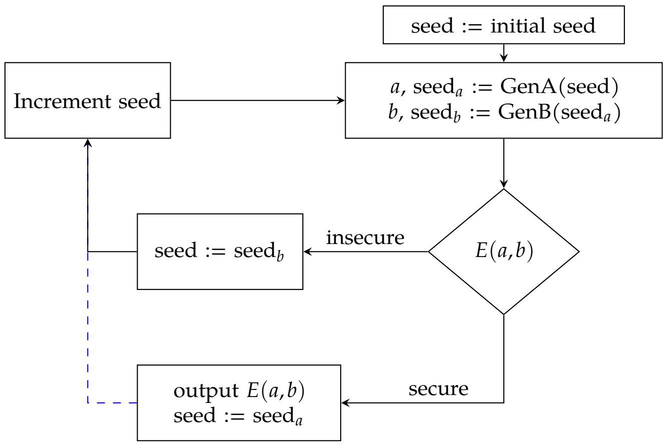

Figure 8.4: Brainpool algorithm adjusted (indicated by dashed line) for large-scale generation.

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 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 of Frobenius 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. Even 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.

Pseudorandom – our approach. The final category of simulated curve is our own and does not relate to any standard. Since all of the three simulated standards contain certain unique properties (small b for NUMS, 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 pseudorandom curves for bit-lengths 128, 160, 192, 224, 256 using the method depicted in Figure 8.5. Moreover, for each curve, we generated a prime for the base field by hashing a seed and taking the smallest prime bigger than this hash when interpreted as an integer. We kept such a curve if it is not anomalous, has the cofactor 1, 2, 4 or 8 and satisfies the SafeCurves criteria on embedding degree and complex multiplication discriminant. We have tried 5 million seeds, resulting in over 250 000 Pseudorandom curves.

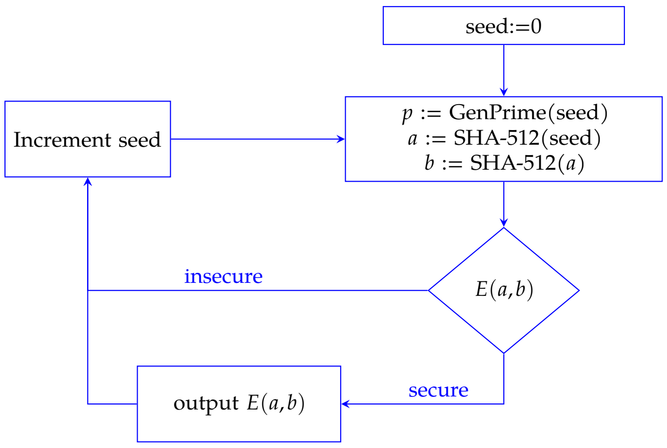

Figure 8.5: Pseudorandom simulation.

8.3.3 Outlier detection

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

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 bitlength that were generated by the same method. Table 8.2 shows numbers of curves contained within each dataset. The rows correspond to the used generation method and the columns to the bitlength of the selected curves.

	256 bits	224 bits	192 bits	160 bits	128 bits
X9.62	18 502	22 211	18 836	27 780	36 126
Brainpool	1 677	2 361	2 640	3 184	0
NUMS	83	109	191	325	0
Pseudorandom	18 636	21 226	24 805	29 639	37 311

Table 8.2: Numbers of curves in our datasets of simulated curves.

Additionally, in cases when trait results are (mostly) independent of curve bitlength, we analyzed curves of all bitlengths together. This approach allowed us to analyze even curves that did not match any of the generated categories.

In order to use outlier detection in conformity with the distinguishing strategies, we augmented each dataset with the corresponding standard curves. Based on these datasets, we derived feature vectors consisting of all computed trait results, as well as their interesting 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).

We ran the k-nearest neighbors algorithm to identify outliers within each augmented dataset, focusing on the results that reported few outliers. However, as some of the strategies inherently output many false positives, we also inspected the results with few standard curve outliers.

8.4 Traits

DiSSECT currently contains 22 trait functions – traits for brevity – designed to test a wide range of elliptic curve properties. We have divided them into two categories: algebraic[11] traits focusing on mathematical characteristics of curves (denoted with a prefix `a`) and traits targeting properties connected to curves standards or implementations (denoted with a prefix `i`). We can subdivide the traits even further:

Potential attacks. Traits `a01`, `a03`, `a07` test the classical properties of elliptic curves relevant to known attacks (group structure of the curve; the quadratic twist; the embedding degree). However, we also cover lesser-known and threatening attacks. Trait `a04` analyzes factorization of values of the form $kn \pm 1$ as a generalization of [Che06]. To test scalar multiplication, essential to all ECC protocols, trait `a12` inspects multiplicative orders of small values modulo the group order n. Trait `i06` follows the idea from [Che02], with a possible connection of the discrete logarithm on ECC to factorization.

11. Introduction to the algebraic properties of curves mentioned in this work can be found in [Was08].

Complex multiplication. Although 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. Traits `a02`, `a06`, `a25` deal with the factorization of t, the size and the factorization of the Frobenius discriminant $D := t^2 - 4p$, as well as how D changes as we move the defining base field to its extensions. Trait `a08` computes bounds on the class number using the Dirichlet class number formula [Dav80].

Torsion. To directly analyze the torsion points of a given elliptic curve, we have created traits `a05`, `a22` and `a29`. Trait `a05` computes the degree of the extension $\mathbb{F}_{p^k}/\mathbb{F}_p$ over which the torsion subgroup is (partially) defined. Trait `a22` approaches torsion from the division polynomials direction and computes their factorization. Trait `a29` considers the lift $E(\mathbb{F}_p) \to E(\mathbb{Q})$ and computes the size of the torsion subgroup over $\mathbb{Q}$.

Isogenies. Both torsion and complex multiplication can be described using isogenies. 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 $\mathbb{Q}(\sqrt{D})$. Trait `a24`, similarly as `a05`, computes the degree of the extension where some/all isogenies of a given degree are defined. Isogenies of ordinary curves form, in general, isogeny volcanoes; trait `a23` measures their depth and the shape of the so-called crater. Trait `a28` computes the number of neighbors for a given vertex in the isogeny graph.

The so far mentioned four categories of traits systematically examine algebraic properties of elliptic curves, thus providing a comprehensive analysis tool. In particular, it covers all SafeCurves ECDLP requirements.

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 8.4) causes overlaps in the binary representations of the curve coefficients in roughly half of the generated curves (trait `i14`). The NUMS standard, as well as the Koblitz curves, use small curve coefficients by design (trait `i15`). The X9.62 standard uses the relation $b^2r = a^3$ to determine the curve coefficients from r (trait `i13`). The SECG standard recommends the curves secp224k1 and secp256k1 together with the group generators; the x-coordinate of half of both of these generators is the same small number (trait i08).

Most attacks on the ECDLP are aiming at specific implementation vulnerabilities. To address this, we propose traits that target possible irregularities in practical implementations. Trait i07 follows the idea of [Wei+20], where the authors analyze the side-channel leakage caused by improper representation of large integers in memory. Based on [Bai+b], trait i04 analyzes the number of points with a low Hamming weight on a given curve.

For a detailed description of each trait, see Appendix F.1.

8.4.1 Notable findings

GOST curves. Trait i15 analyzes the size of the Weiestrass form coefficients. The trait was motivated by the NUMS generation method, which produces curves with a small b coefficient – as can be seen in Figure 8.7, where we compare the standard NUMS curves with Pseudorandom curves according to strategy 2. In Figure 8.7, there are two Weierstrass NUMS curves corresponding to the leftmost values and two Edwards curves corresponding to the rightmost value (there is no reason to expect a small b coefficient after a transformation from the Edwards form to the Weierstrass form).

However, our outlier detection also recognized two 256-bit GOST curves (CryptoPro-A-ParamSet, CryptoPro-C-ParamSet) and a closer inspection in Figure 8.7 revealed that both curves have small b coefficients (166 and 32858). This contradicts Alekseev, Nikolaev, and Smyshlyaev [ANS18], who claim that all of the standardized GOST R curves were generated in the following way:

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 embeddding 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. We hypothesise that the `CryptoPro-A-ParamSet` curve was generated by incrementing b from 0 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 $b = 36$). On the other hand, the `CryptoPro-C-ParamSet` does not have this property, as its b coefficient 32858 is only the 80th smallest such value.

Furthermore, the `a02` trait found that the third curve from [PLK06], called `CryptoPro-B-ParamSet`, has a CM discriminant of -619. Such a small value is extremely improbable, unless the curve was generated by the CM method [Bro06]. (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.)

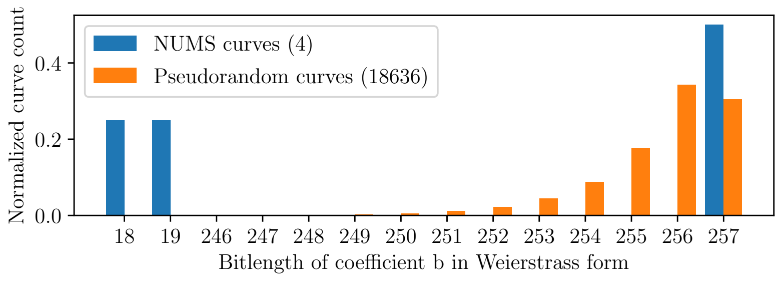

Figure 8.6: Trait `i15` for 256-bit NUMS curves and Pseudorandom curves.

The BLS12-381 curve. The possibilities for the multiplicative order of an element of $\mathbb{Z}_n^*$ depend on the factorization of n. Trait `a12` measures $\phi(n)$ divided by the order of 2, so low orders translate to high values and vice versa. The outlier in Figure 8.8 is the BLS12-381 curve [Bow17].

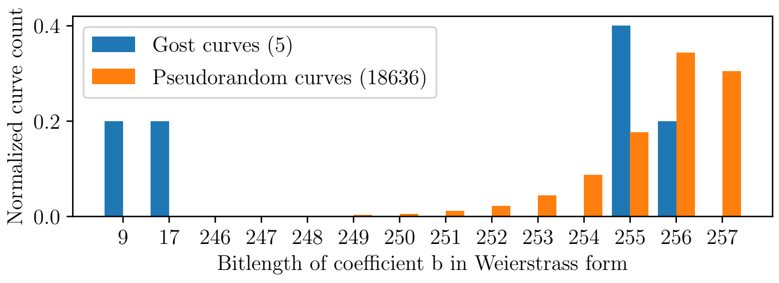

Figure 8.7: Trait i15 for 256-bit GOST curves and Pseudorandom curves.

Indeed, its closer inspection revealed that for all prime p_i dividing n, the values $p_i - 1$ are quite smooth and often share a factor, imposing an upper bound on the order of $\mathbb{Z}_n^*$ elements, which was the real cause of our observation. It seems that this property might be common to the BLS generation process, but this is not documented and requires further investigation.

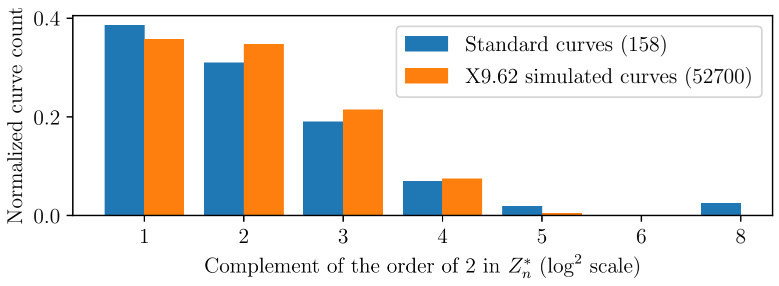

Figure 8.8: Trait a12 for all standard and X9.62 simulated curves.

The Bitcoin curve. Trait i08 inspects the x-coordinate of inverted generator scalar multiples, i.e., x-coordinates of points $[k^{-1}]G$, where $k \in \{1, \ldots, 8\}$. The motivation behind this trait is due to Brengel et al. [BR18], who reported an unexpectedly low value for $k = 2$ on the Bitcoin curve secp256k1. Furthermore, Maxwell [Max15] pointed out that secp224k1 yields exactly the same result. Pornin [Por19] 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 8.9) 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 bitlength.

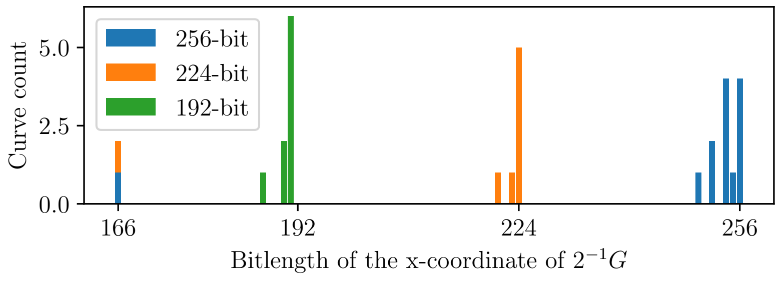

Figure 8.9: Trait `i08` for all 192-, 224-, and 256-bit standard curves.

Brainpool overlaps. Trait `i14` – designed to detect bit-overlaps in the Weierstrass coefficients, see Appendix F.1 for details – revealed a structure in the Brainpool curves (Figure 8.10), already observed by Bernstein et al. [Ber+15].

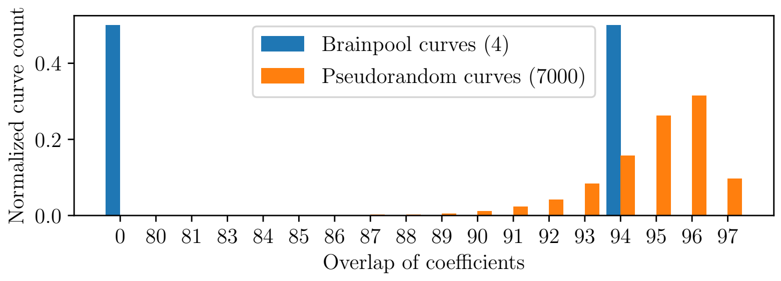

Figure 8.10: Trait `i14` for 256-bit Brainpool and Pseudorandom curves.

Inspecting `brainpoolP256r1`, we see the identical segments in a, b:

$a =$ `0x7d5a0975fc2c3057eef67530417affe7fb8055c126dc5c6ce94a4b44f330b5d9`

$b =$ `0x26dc5c6ce94a4b44f330b5d9bbd77cbf958416295cf7e1ce6bccdc18ff8c07b6`

Such overlaps occur for roughly half of the curves (Figure 8.11); e.g., this is the case for `brainpoolP{192,256,384}r1` curves, but not for `brainpoolP{160,224,320}r1`. More specifically, after generating a, the seed is incremented, and b is produced from this seed using Function 8.1 (which itself has an incremental character). This is repeated until b is not a square. The overlaps occur when the loop only executes once. E.g., for `brainpoolP320r1`, if we assume that b was a

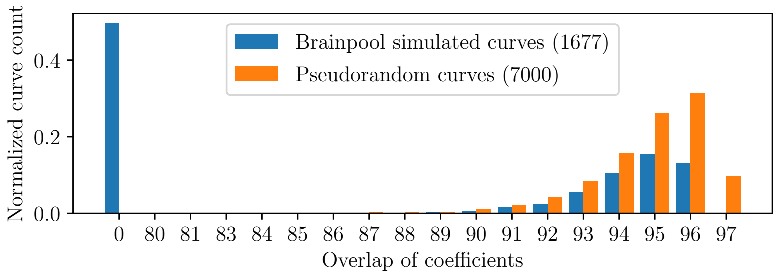

Figure 8.11: Trait `i14` for 256-bit simulated Brainpool and Pseudorandom curves.

square on the second pass, we can actually reconstruct the discarded b' of the first try without the original seed:

a = `0x3ee30...a73513f5eb79da66...d860eb4`

b' = `0x75eb79da66...d860eb4d20883949dfdbc42d3ad198640688a6fe13f4134`

b = `0x520883949dfdbc42d3ad198640688a6fe13f41349554b...b1f1a6`

Indeed, b' is a square (else we could construct b'', b''', and so on).

8.5 Our tool 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. Its code contains implementations of the 22 traits described in the previous section, but it can be easily extended using a trait addition script. 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 any given curve.

We offer two method to analyze trait results: a Jupyter notebook and an automated outlier detection. Both of these tools allow configuring the subset of analyzed curves, traits, and their parameters, and can access locally stored data as well as our publicly available database.

Our website[12] contains detailed information about curves in our database and their trait results, and trait descriptions with statistics.

12. `https://dissect.crocs.fi.muni.cz/`

For a more complex data inspection, the analysis environment configured with access to our database can be launched directly from the website. Additionally, the frontend provides an API for accessing the database to analyze the data without any need for its local storage.

8.6 Conclusions

Our framework DiSSECT[13] aspires to survey all standard elliptic curves and find any 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 hope that more cryptographers and mathematicians will join the project.

Our tool revealed that the generation process of three GOST curves described by Alekseev, Nikolaev, and Smyshlyaev [ANS18] is inconsistent with the sizes of the b cofficient in two cases and with the size of the CM discriminant in the third one. We also found an interesting property of the BLS12-381 curve (related to smoothness) that might be caused by its generation, but is not documented anywhere.

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 and Lenstra [ML15], 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 choice and curve sets manually. Thus we employed an automatic outlier detection method, confirming our prior findings. 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.

13. `https://github.com/crocs-muni/DiSSECT`

Conclusions

While each chapter of this thesis includes its individual conclusions, let us come back to the big picture and highlight the most important lessons learned. Unlike RSA, ECC provides us with a hard problem that we are unable to solve with a subexponential algorithm, resulting in short keys well suited for many practical applications. However, ECC is also vastly more complex than RSA and should be treated as such.

The first problem is choosing a curve where the discrete logarithm problem is hard, because there is no known way to prove this. While RSA uses a new group for each instance, ECC supports only a few standardized choices, thus centralizing the whole system. This takes one burden off the implementors, but greatly increases the impact of any potential backdoors. Since the standard curves were not always generated in a transparent way, it is critical to develop mechanisms to analyze them and increase the public trust where it is deserved. We believe that our tool DiSSECT (introduced in Chapter 8) is a glimpse of hope in this direction.

Even when using secure curves (and protocols), the story does not end yet. The path to implementing ECC is loaded with a plethora of traps at many levels. Chapter 4 stresses the importance of correctly validating all domain parameters as a defence against attackers who can manipulate them. Choosing a secure and efficient addition formula and implementing it correctly is also hard, especially in the light of the attacks described in Chapter 6. If the addition formula is incomplete, many scalar multiplication algorithms might leak secrets – indeed, this ultimately led to our real-world attack Minerva (presented in Chapter 5).

Our research adds more evidence to the long line of historical problems with ECC implementations. Indeed, security is often tricky to achieve in the real world, where we often value performance, seek backward compatibility and build both software and hardware upon older layers in complex ways.

The bottom line is that ECC is a great tool (at least in the pre-quantum world), but we should use it carefully, building on the best practices from both research and industry that surfaced over the years.

Bibliography

[ACL] R. Abarzua, C. V. Cordero, and J. Lopez. *Survey for Performance & Security Problems of Passive Side-channel Attacks Countermeasures in ECC.* IACR Cryptology ePrint Archive, Report 2019/010.

[Adr+15] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thome, L. Valenta, et al. "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice". In: *22nd ACM CCS*. 2015, pp. 5–17.

[Age20] N. S. Agency. *Windows CryptoAPI Spoofing Vulnerability (CVE-2020-0601)*. 2020. URL: `https://nvd.nist.gov/vuln/detail/CVE-2020-0601` (visited on 09/24/2021).

[AT03] T. Akishita and T. Takagi. "Zero-Value Point Attacks on Elliptic Curve Cryptosystem". In: *Information Security, 6th International Conference, (ISC)*. Vol. 2851. Lecture Notes in Computer Science. Springer, 2003, pp. 218–233.

[Alb+19] M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens. "The General Sieve Kernel and New Records in Lattice Reduction". In: *EUROCRYPT, Proceedings, Part II*. Vol. 11477. Lecture Notes in Computer Science. Springer, 2019, pp. 717–746.

[Alb+18] M. R. Albrecht, J. Massimo, K. G. Paterson, and J. Somorovsky. "Prime and Prejudice: Primality Testing Under Adversarial Conditions". In: *25th ACM CCS*. 2018, pp. 281–298.

[Ald+19] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. P. García, and N. Tuveri. "Port Contention for Fun and Profit". In: *IEEE Symposium on Security and Privacy*. 2019, pp. 870–887.

[ANS18] E. K. Alekseev, V. Nikolaev, and S. V. Smyshlyaev. *On the security properties of Russian standardized elliptic curves*. 2018.

[AB21] H. K. Alper and J. Burdges. "Two-Round Trip Schnorr Multi-signatures via Delinearized Witnesses". In: *CRYPTO*. Springer. 2021.

[ANSI98] *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.

[ANS14] ANSSI. *Référentiel Général de Sécurité, version 2.0, Annexe B1*. `https://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B1.pdf`. 2014. (Visited on 09/24/2021).

[Aon+16] Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. "Improved Progressive BKZ Algorithms and Their Precise Cost Estimation by Sharp Simulator". In: *EUROCRYPT, Proceedings, Part I*. 2016, pp. 789–819.

[Ara+14] D. F. Aranha, P. Fouque, B. Gerard, J. Kammerer, M. Tibouchi, and J. Zapalowicz. "GLV/GLS Decomposition, Power Analysis, and Attacks on ECDSA Signatures with Single-Bit Nonce Bias". In: *ASIACRYPT, Proceedings, Part I*. 2014, pp. 262–281.

[Ara+20] D. F. Aranha, F. R. Novaes, A. Takahashi, M. Tibouchi, and Y. Yarom. "Ladderleak: Breaking ecdsa with less than one bit of nonce leakage". In: *27th ACM CCS*. 2020, pp. 225–242.

[Ara+] D. F. Aranha1, 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.

[Arn95a] F. Arnault. "Constructing Carmichael numbers which are strong pseudoprimes to several bases". In: *Journal of Symbolic Computation* 20.2 (1995), pp. 151–161.

[Arn95b] F. Arnault. "Rabin-Miller primality test: composite numbers which pass it". In: *Mathematics of Computation* 64.209 (1995), pp. 355–361.

[Ath12a] Athena Smartcard. *IDProtect with LASER PKI*. 2012. URL: `https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp1711.pdf`.

[Ath12b] Athena Smartcard. *OS755/IDProtect v6 SSCD – Security Target*. 2012. URL: `https://www.ssi.gouv.fr/uploads/IMG/certificat/ANSSI-CC-cible_2012-23en.pdf`.

[Atm09] Atmel. *Atmel toolbox 00.03.11.05 on the AT90SC Family of Devices: Security target lite*. 2009. URL: `https://www.ssi.gouv.fr/uploads/IMG/certificat/dcssi-cible_2009-11en.pdf`.

[Aza+17] 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". In: *Submission to the NIST Post-Quantum Standardization project* (2017).

[Bab86] L. Babai. "On Lovasz' lattice reduction and the nearest lattice point problem". In: *Combinatorica* 6.1 (1986), pp. 1–13.

[BS85] E. Bach and J. Shallit. "Factoring with Cyclotomic Polynomials". In: *Mathematics of Computation*. Vol. 52. IEEE, 1985, pp. 443–450.

[Bai+a] T. Baignères, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain. *Trap Me If You Can – Million Dollar Curve*. Cryptology ePrint Archive, Report 2015/1249.

[Bai+b] D. V. Bailey et al. *Breaking ECC2K-130*. Cryptology ePrint Archive, Report 2009/541.

[BLS] P. S. L. M. Barreto, B. Lynn, and M. Scott. *Constructing Elliptic Curves with Prescribed Embedding Degrees*. Cryptology ePrint Archive, Report 2002/088.

[Bel+16] P. Belgarric, P. Fouque, G. Macario-Rat, and M. Tibouchi. "Side-Channel Analysis of Weierstrass and Koblitz Curve ECDSA on Android Smartphones". In: *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*. 2016, pp. 236–252.

[Bel+03] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. "The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme." In: *Journal of Cryptology* 16.3 (2003).

[BN06] M. Bellare and G. Neven. "Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma". In: *13th ACM CCS*. 2006, pp. 390–399.

[BR93] M. Bellare and P. Rogaway. “Random oracles are practical: A paradigm for designing efficient protocols”. In: *1st ACM CCS*. 1993, pp. 62–73.

[Bel+20] 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: *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC)*. ACM, 2020, pp. 760–771.

[Ben+14] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom. “"Ooh Aah... Just a Little Bit" : A Small Amount of Side Channel Can Go a Long Way”. In: *Cryptographic Hardware and Embedded Systems (CHES)*. 2014, pp. 75–92.

[Ben+21] F. Benhamouda, T. Lepoint, J. Loss, M. Orru, and M. Raykova. “On the (in) security of ROS”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, pp. 33–53.

[Ber06] D. J. Bernstein. “Curve25519: new Diffie-Hellman speed records”. In: *International Workshop on Public Key Cryptography*. Springer. 2006, pp. 207–228.

[Ber14] D. J. Bernstein. *The cr.yp.to blog: How to design an elliptic-curve signature system*. `https://blog.cr.yp.to/20140323-ecdsa.html`. 2014. (Visited on 09/24/2021).

[Ber19] D. J. Bernstein. *The cr.yp.to blog: Why EdDSA held up better than ECDSA against Minerva*. `https://blog.cr.yp.to/20191024-eddsa.html`. 2019. (Visited on 09/24/2021).

[Ber+08] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. “Twisted Edwards Curves”. In: *AFRICACRYPT, 1st International Conference on Cryptology in Africa*. Vol. 5023. Lecture Notes in Computer Science. Springer, 2008, pp. 389–405.

[BCL] D. J. Bernstein, C. Chuengsatiansup, and T. Lange. *Curve41417: Karatsuba revisited*. Cryptology ePrint Archive, Report 2014/526.

[BL07a] D. J. Bernstein and T. Lange. *Explicit-Formulas Database (EFD)*. 2007. URL: `https://www.hyperelliptic.org/EFD/`.

[BL07b] D. J. Bernstein and T. Lange. “Faster Addition and Doubling on Elliptic Curves”. In: *ASIACRYPT*. Vol. 4833. Lecture Notes in Computer Science. Springer, 2007, pp. 29–50.

[BL07c] D. J. Bernstein and T. Lange. “Inverted Edwards Coordinates”. In: *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC-17), 17th International Symposium*. Vol. 4851. Lecture Notes in Computer Science. Springer, 2007, pp. 20–27.

[BL] D. J. Bernstein and T. Lange. *SafeCurves: choosing safe curves for elliptic-curve cryptography*. URL: `https://safecurves.cr.yp.to/` (visited on 09/24/2021).

[BLS12] D. J. Bernstein, T. Lange, and P. Schwabe. “The Security Impact of a New Cryptographic Library”. In: *LATINCRYPT, 2nd International Conference on Cryptology and Information Security in Latin America*. 2012, pp. 159–176.

[Ber+15] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hulsing, E. Lambooij, T. Lange, R. Niederhagen, and C. Van Vredendaal. “How to Manipulate Curve Standards: A White Paper for the Black Hat”. In: *International Conference on Research in Security Standardisation*. Springer. 2015, pp. 109–139. URL: `https://bada55.cr.yp.to` (visited on 09/24/2021).

[Ber+12] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. “High-speed high-security signatures”. In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89.

[Ber+13] 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.

[Ber+] D. J. Bernstein, S. Josefsson, T. Lange, P. Schwabe, and B.-Y. Yang. *EdDSA for more curves*. IACR Cryptology ePrint Archive, Report 2015/677.

[BL16] D. J. Bernstein and T. Lange. *Failures in NIST’s ECC standards*. `https://cr.yp.to/newelliptic/nistecc-20160106.pdf`. 2016. (Visited on 09/24/2021).

[BLN16] D. J. Bernstein, T. Lange, and R. Niederhagen. "Dual EC: A standardized back door". In: *The New Codebreakers*. Springer, 2016, pp. 256–281.

[BMM00] I. Biehl, B. Meyer, and V. Muller. "Differential fault attacks on elliptic curve cryptosystems". In: *CRYPTO*. Springer, 2000, pp. 131–146.

[Bla+14] 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.

[Bla+06a] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*. RFC 4492. RFC Editor, 2006, pp. 1–35. URL: `https://tools.ietf.org/html/rfc4492`.

[Bla+06b] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. *Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS)*. Tech. rep. RFC 4492, 2006.

[Ble05] D. Bleichenbacher. "Breaking a Cryptographic Protocol with Pseudoprimes". In: *Public Key Cryptography (PKC)*. 2005, pp. 9–15.

[BDN18] D. Boneh, M. Drijvers, and G. Neven. "Compact multi-signatures for smaller blockchains". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2018, pp. 435–464.

[BDH99] D. Boneh, G. Durfee, and N. Howgrave-Graham. "Factoring $N = p^r q$ for Large r". In: *CRYPTO*. Springer-Verlag, 1999, pp. 326–337. URL: `http://dl.acm.org/citation.cfm?id=646764.703963`.

[BV96] D. Boneh and R. Venkatesan. "Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes". In: *CRYPTO*. Springer, 1996, pp. 129–142.

[BL95] W. Bosma and H. W. Lenstra Jr. "Complete systems of two addition laws for elliptic curves". In: *Journal of Number Theory* 53.2 (1995), pp. 229–240.

[Bou+20] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé, and P. Zimmermann. "Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment". In: *CRYPTO*. Springer. 2020, pp. 62–91.

[Bow17] S. Bowe. *BLS12-381: New zk-SNARK Elliptic Curve Construction*. `https://electriccoin.co/blog/new-snark-curve/`. 2017. (Visited on 09/24/2021).

[BGH] S. Bowe, J. Grigg, and D. Hopwood. *Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021.

[Bra10] Brainpool. *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*. Tech. rep. IETF RFC 5639, 2010.

[BH19] J. Breitner and N. Heninger. "Biased Nonce Sense: Lattice Attacks Against Weak ECDSA Signatures in Cryptocurrencies". In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 3–20.

[BR18] 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. Cham: Springer International Publishing, 2018, pp. 623–643.

[BJ02] E. Brier and M. Joye. "Weierstraß Elliptic Curves and Side-Channel Attacks". In: *Public Key Cryptography (PKC)*. Vol. 2274. Lecture Notes in Computer Science. Springer, 2002, pp. 335–345.

[Bro06] R. Broker. "Constructing elliptic curves of prescribed order". PhD thesis. Thomas Stieltjes Institute for Mathematics, 2006.

[BS07] R. Broker and P. Stevenhagen. "Efficient CM-constructions of elliptic curves over finite fields". In: *Mathematics of Computation*. Vol. 76. AMS, 2007, pp. 2161–2179.

[BH09] B. B. Brumley and R. M. Hakala. "Cache-Timing Template Attacks". In: *ASIACRYPT*. Vol. 5912. Lecture Notes in Computer Science. Springer, 2009, pp. 667–684.

[BT11] B. B. Brumley and N. Tuveri. "Remote Timing Attacks Are Still Practical". In: *16th European Symposium on Research in Computer Security (ESORICS)*. 2011, pp. 355–371.

[Bun18] Bundesamt fur Sicherheit in der Informationstechnik. *Cryptographic Mechanisms: Recommendations and Key Lengths*. Technical Guideline: TR-02102-1. BSI, 2018. URL: `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=7`.

[CADO17] CADO-NFS Development Team. *CADO-NFS, An Implementation of the Number Field Sieve Algorithm*. Release 2.3.0. 2017. URL: `https://cado-nfs.gforge.inria.fr/` (visited on 09/24/2021).

[CGF] W. Castryck, S. D. Galbraith, and R. R. Farashahi. *Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation*. IACR Cryptology ePrint Archive, Report 2008/218.

[Cas+18] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. "CSIDH: an efficient post-quantum commutative group action". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2018, pp. 395–427.

[Cer10] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0.* 2010. URL: `https://secg.org/` (visited on 09/24/2021).

[Cer] Certicom Research. *Standards for Efficient Cryptography Group*. `https://secg.org/`. (Visited on 09/24/2021).

[Che+16] 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: *23rd ACM CCS*. 2016, pp. 468–479.

[Che02] Q. Cheng. *A New Special-Purpose Factorization Algorithm*. Citeseer. 2002. URL: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.9071&rep=rep1&type=pdf` (visited on 09/24/2021).

[Che] Q. Cheng. *A New Class of Unsafe Primes.* IACR Cryptology ePrint Archive, Report 2002/109.

[Che06] J. H. Cheon. "Security Analysis of the Strong Diffie-Hellman Problem". In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by S. Vaudenay. Springer Berlin Heidelberg, 2006.

[Chro19] Chromium OS. *U2F ECDSA vulnerability - The Chromium Projects*. URL: `https://sites.google.com/a/chromium.org/dev/chromium-os/u2f-ecdsa-vulnerability` (visited on 09/24/2021).

[CC86] D. V. Chudnovsky and G. V. Chudnovsky. "Sequences of numbers generated by addition in formal groups and new primality and factorization tests". In: *Advances in Applied Mathematics* 7.4 (1986), pp. 385–434.

[CMO98] H. Cohen, A. Miyaji, and T. Ono. "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates". In: *ASIACRYPT*. Vol. 1514. Lecture Notes in Computer Science. Springer, 1998, pp. 51–65.

[Col85] Collectif. "Nombres de classes des corps quadratiques imaginaires". In: *Séminaire Bourbaki: 1983/84, exposés 615-632*. Astérisque 121-122. Société mathématique de France, 1985. URL: `http://www.numdam.org/item/SB_1983-1984__26__309_0/` (visited on 09/24/2021).

[Cor+16] J.-S. Coron, J.-C. Faugere, G. Renault, and R. Zeitoun. "Factoring $N = p^r q^s$ for large r and s". In: *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*. Springer-Verlag, 2016, pp. 448–464.

[CLN10] C. Costello, T. Lange, and M. Naehrig. "Faster Pairing Computations on Curves with High-Degree Twists". In: *Public Key Cryptography (PKC)*. Vol. 6056. Lecture Notes in Computer Science. Springer, 2010, pp. 224–242.

[CLN15] C. Costello, P. Longa, and M. Naehrig. "A brief discussion on selecting new elliptic curves". In: *Microsoft Research. Microsoft* 8 (2015).

[CS18] C. Costello and B. Smith. "Montgomery curves and their arithmetic - The case of large characteristic fields". In: *J. Cryptographic Engineering* 8.3 (2018), pp. 227–240.

[CK] C. Crepeau and R. A. Kazmi. *An Analysis of ZVP-Attack on ECC Cryptosystems*. IACR Cryptology ePrint Archive, Report 2012/329.

[Dal+18] F. Dall, G. D. Micheli, T. Eisenbarth, D. Genkin, N. Heninger, A. Moghimi, and Y. Yarom. "CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.2 (2018), pp. 171–191.

[Dan+13a] J.-L. Danger, S. Guilley, P. Hoogvorst, C. Murdica, and D. Naccache. "A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards". In: *Journal of Cryptographic Engineering* 3.4 (2013), pp. 241–265.

[Dan+13b] J. Danger, S. Guilley, P. Hoogvorst, C. Murdica, and D. Naccache. "Dynamic countermeasure against the Zero Power Analysis". In: *IEEE International Symposium on Signal Processing and Information Technology*. 2013, pp. 140–147.

[Dav80] H. Davenport. *Multiplicative number theory*. 1980, pp. 43–53.

[DY15] 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 09/24/2021).

[DCE] K. Dorey, N. Chang-Fong, and A. Essex. *Indiscreet Logs: Persistent Diffie-Hellman Backdoors in TLS*. IACR Cryptology ePrint Archive, Report 2016/999.

[Dri+19] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs. "On the Security of Two-Round Multi-Signatures". In: *IEEE Symposium on Security and Privacy*. 2019, pp. 1084–1101.

[Dzu+17] P. Dzurenda, S. Ricci, J. Hajny, and L. Malina. "Performance analysis and comparison of different elliptic curves on smart cards". In: *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE. 2017, pp. 365–36509.

[Edw07] H. M. Edwards. "A normal form for elliptic curves". In: *AMS. Bulletin. New Series* 44.3 (2007), pp. 393–422.

[Eni19] EnigmaBridge. *Curated list of JavaCard applications*. 2019. URL: `https://github.com/EnigmaBridge/javacard-curated-list` (visited on 09/24/2021).

[FAIL10] failOverflow. *Console Hacking 2010: PS3 Epic Fail*. URL: `https://www.cs.cmu.edu/~dst/GeoHot/1780_27c3_console_hacking_2010.pdf` (visited on 09/24/2021).

[FGV11] J. Fan, B. Gierlichs, and F. Vercauteren. "To infinity and beyond: Combined attack on ECC using points of low order". In: *Cryptographic Hardware and Embedded Systems (CHES)*. Springer. 2011, pp. 143–159.

[Fan+10] J. Fan, X. Guo, E. De 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: *IEEE international symposium on hardware-oriented security and trust (HOST)*. 2010, pp. 76–87.

[FWC16] S. Fan, W. Wang, and Q. Cheng. "Attacking OpenSSL implementation of ECDSA with a few signatures". In: *23rd ACM CCS*. 2016, pp. 1505–1515.

[FGR12] J.-C. Faugere, C. Goyet, and G. Renault. "Attacking (EC)DSA given only an implicit hint". In: *International Conference on Selected Areas in Cryptography*. Springer. 2012, pp. 252–274.

[Fed16] Federal Agency on Technical Regulating and Metrology. *Information technology. Cryptographic data security. Parameters of elliptic curves for cryptographic algorithms and protocols*. 2016.

[Fei19] Feisty Duck. *Elliptic curve implementations vulnerable to Minerva timing attack*. 2019. URL: `https://www.feistyduck.com/bulletproof-tls-newsletter/issue_58_elliptic_curve_implementations_vulnerable_to_minerva_timing_attack` (visited on 09/24/2021).

[FS86] A. Fiat and A. Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194.

[FIDO18] FIDO Alliance. *FIDO ECDAA Algorithm*. 2018. URL: `https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html`.

[Fou+08] P.-A. Fouque, R. Lercier, D. Real, and F. Valette. "Fault attack on elliptic curve Montgomery ladder implementation". In: *Fault Diagnosis and Tolerance in Cryptography*. IEEE. 2008, pp. 92–98.

[FPL16] FPLLL development team. "fplll, a lattice reduction library". 2016. URL: `https://github.com/fplll/fplll`.

[FR94] G. Frey and H.-G. Ruck. "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves". In: *Mathematics of Computation* 62.206 (1994), pp. 865–874.

[Fri+17] J. Fried, P. Gaudry, N. Heninger, and E. Thome. "A Kilobit Hidden SNFS Discrete Logarithm Computation". In: *EUROCRYPT, Proceedings, Part I*. 2017, pp. 202–231.

[GMP19] S. D. Galbraith, J. Massimo, and K. G. Paterson. "Safety in Numbers: On the Need for Robust Diffie-Hellman Parameter Validation". In: *Public Key Cryptography (PKC), Proceedings, Part II*. 2019, pp. 379–407.

[GLV01] R. P. Gallant, R. J. Lambert, and S. A. Vanstone. "Faster point multiplication on elliptic curves with efficient endomorphisms". In: *CRYPTO*. Springer, 2001, pp. 190–200.

[GNR10] N. Gama, P. Q. Nguyen, and O. Regev. "Lattice Enumeration Using Extreme Pruning". In: *EUROCRYPT*. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 257–278.

[Gar+20] C. P. Garcia, S. ul Hassan, N. Tuveri, I. Gridin, A. C. Aldaya, and B. B. Brumley. "Certified Side Channels". In: *The 29th USENIX Security Symposium*. 2020, pp. 2021–2038.

[Gar+21] F. Garillot, Y. Kondi, P. Mohassel, and V. Nikolaenko. "Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions". In: *CRYPTO*. Springer. 2021, pp. 127–156.

[Gau06] P. Gaudry. "Variants of the Montgomery form based on Theta functions". In: *Toronto* (2006).

[Gen+16] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom. "ECDSA key extraction from mobile devices via nonintrusive physical side channels". In: *23rd ACM CCS*. 2016, pp. 1626–1638.

[GVY17] D. Genkin, L. Valenta, and Y. Yarom. "May the fourth be with you: A microarchitectural side channel attack on several real-world applications of Curve25519". In: *24th ACM CCS*. 2017, pp. 845–858.

[Gir19] D. Giry. *Cryptography Key Length Recommendations*. `https://www.keylength.com`. 2019. (Visited on 09/24/2021).

[Gol73] L. J. Goldstein. "A history of the prime number theorem". In: *The American Mathematical Monthly* 80.6 (1973), pp. 599–615.

[Gou03] L. Goubin. "A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems". In: *Public Key Cryptography (PKC)*. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 199–210.

[GRV16] D. Goudarzi, M. Rivain, and D. Vergnaud. "Lattice Attacks Against Elliptic-Curve Signatures with Blinded Scalar Multiplication". In: *Selected Areas in Cryptography (SAC), Revised Selected Papers*. 2016, pp. 120–139.

[Hal13] T. C. Hales. "The NSA back door to NIST". In: *Notices of the AMS* 61.2 (2013), pp. 190–192.

[Hama] M. Hamburg. *A note on high-security general-purpose elliptic curves*. Cryptology ePrint Archive, Report 2015/625.

[Hamb] M. Hamburg. *Ed448-Goldilocks, a new elliptic curve.* IACR Cryptology ePrint Archive, Report 2015/625.

[HMV04] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Professional Computing. Springer, 2004.

[Har08] D. Harkins. *Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)*. RFC 5297. RFC Editor, 2008, pp. 1–26. URL: `https://tools.ietf.org/html/rfc5297`.

[HNM98] T. Hasegawa, J. Nakajima, and M. Matsui. "A Practical Implementation of Elliptic Curve Cryptosystems over GF(p) on a 16-bit Microcomputer". In: *Public Key Cryptography (PKC)*. Vol. 1431. Lecture Notes in Computer Science. Springer, 1998, pp. 182–194.

[Hen+12] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. “Mining your Ps and Qs: Detection of widespread weak keys in network devices”. In: *The 21st USENIX Security Symposium*. 2012, pp. 205–220.

[HCD07] H. Hisil, G. Carter, and E. Dawson. “New Formulae for Efficient Elliptic Curve Arithmetic”. In: *INDOCRYPT, 8th International Conference on Cryptology in India*. Vol. 4859. Lecture Notes in Computer Science. Springer, 2007, pp. 138–151.

[His+08] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. “Twisted Edwards curves revisited”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2008, pp. 326–343.

[HR06] M. Hlavac and T. Rosa. “Extended Hidden Number Problem and Its Cryptanalytic Applications”. In: *Selected Areas in Cryptography (SAC), Revised Selected Papers*. Vol. 4356. Lecture Notes in Computer Science. Springer, 2006, pp. 114–133.

[Hol+08] A. Hollosi, G. Karlinger, T. Rossler, M. Centner, and et al. *Die österreichische bürgerkarte*. `https://www.buergerkarte.at/konzept/securitylayer/spezifikation/20080220/`. 2008. (Visited on 09/24/2021).

[Hop20] D. Hopwood. *The pasta curves*. `https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/`. 2020. (Visited on 09/24/2021).

[HS01] N. Howgrave-Graham and N. P. Smart. “Lattice Attacks on Digital Signature Schemes”. In: *Designs, Codes and Cryptography* 23.3 (2001), pp. 283–290.

[IANa] IANIX. *Things that use Curve25519*. URL: `https://ianix.com/pub/curve25519-deployment.html` (visited on 09/24/2021).

[IANb] IANIX. *Things that use Ed25519*. URL: `https://ianix.com/pub/ed25519-deployment.html` (visited on 09/24/2021).

[ICAO15] *Doc 9303 - Machine Readable Travel Documents*. Document. International Civil Aviation Organization, 2015.

[IEEE00] *IEEE Standard - Specifications for Public-Key Cryptography*. Standard. IEEE Std 1363-2000 Working Group, 2000.

[ISO17] ISO/IEC 15946. *Information technology — Security techniques — Cryptographic techniques based on elliptic curves — Part 5: Elliptic curve generation*. 2017.

[IT02] T. Izu and T. Takagi. “A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks”. In: *Public Key Cryptography (PKC)*. Vol. 2274. Lecture Notes in Computer Science. Springer, 2002, pp. 280–296.

[IT03] T. Izu and T. Takagi. “Exceptional Procedure Attack on Elliptic Curve Cryptosystems”. In: *Public Key Cryptography (PKC)*. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 224–239.

[JSS15] T. Jager, J. Schwenk, and J. Somorovsky. “Practical invalid curve attacks on TLS-ECDH”. In: *20th European Symposium on Research in Computer Security*. Springer. 2015, pp. 407–425.

[Jan19] J. Jancar. *ecgen*. 2019. URL: `https://github.com/J08nY/ecgen` (visited on 09/24/2021).

[Jan20] 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/`.

[Jan+20] J. Jancar, V. Sedlacek, P. Svenda, and M. Sys. “Minerva: The curse of ECDSA nonces; Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces”. In: *Cryptographic Hardware and Embedded Systems (CHES)*. Ruhr-University of Bochum, 2020.

[JS19] J. Jancar and P. Svenda. *ECTester*. 2019. URL: `https://crocs-muni.github.io/ECTester/`.

[Jou11] Journal officiel de la republique francaise. *Avis relatif aux paramètres de courbes elliptiques définis par l’Etat français*. 2011. URL: `https://www.legifrance.gouv.fr/download/pdf?id=QfYWtPSAJVtAB_c6Je5tAv0OOY2r1ad3LaVVmnStGvQ=`.

[Kob87] N. Koblitz. “Elliptic curve cryptosystems”. In: *Mathematics of Computation* 48.177 (1987), pp. 203–209.

[KM16] N. Koblitz and A. Menezes. “A riddle wrapped in an enigma”. In: *IEEE Security & Privacy* 14.6 (2016), pp. 34–42.

[KMV00] N. Koblitz, A. Menezes, and S. Vanstone. "The State of Elliptic Curve Cryptography". In: *Towards a Quarter-Century of Public Key Cryptography: A Special Issue of DESIGNS, CODES AND CRYPTOGRAPHY An International Journal. Volume 19, No. 2/3*. Springer, 2000, pp. 103–123.

[KG] C. Komlo and I. Goldberg. *FROST: Flexible Round-Optimized Schnorr Threshold Signatures*. IACR Cryptology ePrint Archive, Report 2020/852.

[Len87] H. W. Lenstra. "Factoring Integers with Elliptic Curves". In: *Annals of Mathematics*. Vol. 126. Princeton University, 1987, pp. 649–673. URL: `https://www.jstor.org/stable/1971363` (visited on 09/24/2021).

[LLL+82] H. W. Lenstra, A. K. Lenstra, L. Lovfiasz, et al. "Factoring polynomials with rational coeficients". In: (1982).

[LL97] C. H. Lim and P. J. Lee. "A key recovery attack on discrete log-based schemes using a prime order subgroup". In: *CRYPTO*. Springer, 1997, pp. 249–263.

[LCL13] M. Liu, J. Chen, and H. Li. "Partially Known Nonces and Fault Injection Attacks on SM2 Signature Algorithm". In: *Information Security and Cryptology - 9th International Conference, Inscrypt 2013, Guangzhou, China, November 27-30, 2013, Revised Selected Papers*. 2013, pp. 343–358.

[Loc+15] M. Lochter, J. Merkle, J.-M. Schmidt, and T. Schutze. *Requirements for Elliptic Curves for High-Assurance Applications*. `https://csrc.nist.gov/csrc/media/events/workshop-on-elliptic-curve-cryptography-standards/documents/presentations/session4-merkle-johannes.pdf`. 2015. (Visited on 09/24/2021).

[Loc+] M. Lochter, J. Merkle, J.-M. Schmidt, and T. Schutze. *Requirements for Standard Elliptic Curves.* IACR Cryptology ePrint Archive, Report 2014/832.

[Mar+13] S. Martinez, D. Sadornil, J. Tena, R. Tomas, and M. Valls. "On Edwards curves and ZVP-attacks". In: *Appl. Algebra Eng. Commun. Comput.* 24.6 (2013), pp. 507–517.

[MP20] J. Massimo and K. G. Paterson. "A Performant, Misuse-Resistant API for Primality Testing". In: *27th ACM CCS*. 2020, pp. 195–210.

[Mav+17] V. Mavroudis, A. Cerulli, P. Svenda, D. Cvrcek, D. Klinec, and G. Danezis. "A touch of evil: High-assurance cryptographic hardware from untrusted components". In: *24th ACM CCS*. 2017, pp. 1583–1600.

[MS20] V. Mavroudis and P. Svenda. "JCMathLib: Wrapper Cryptographic Library for Transparent and Certifiable JavaCard Applets". In: *IEEE European Symposium on Security and Privacy Workshops* (2020), pp. 89–96.

[Max15] G. Maxwell. *The most repeated R value on the blockchain*. `https://bitcointalk.org/index.php?topic=1118704.0`. 2015. (Visited on 09/24/2021).

[Max+19] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. "Simple Schnorr multi-signatures with applications to Bitcoin". In: *Designs, Codes and Cryptography* 87.9 (2019), pp. 2139–2164.

[Mel07] N. Meloni. "New Point Addition Formulae for ECC Applications". In: *Arithmetic of Finite Fields, First International Workshop, (WAIFI)*. Vol. 4547. Lecture Notes in Computer Science. Springer, 2007, pp. 189–201.

[MOV93] A. J. Menezes, T. Okamoto, and S. A. Vanstone. "Reducing elliptic curve logarithms to logarithms in a finite field". In: *IEEE Transactions on information Theory* 39.5 (1993), pp. 1639–1646.

[ML15] A. Miele and A. K. Lenstra. "Efficient ephemeral elliptic curve cryptographic keys". In: *International Conference on Information Security*. Springer. 2015, pp. 524–547.

[Mil75] G. L. Miller. "Riemann's Hypothesis and Tests for Primality". In: *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*. STOC '75. 1975, pp. 234–239.

[Mil85] V. S. Miller. "Use of Elliptic Curves in Cryptography". In: *CRYPTO*. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426.

[MIR18] MIRACL UK Ltd. *Multiprecision Integer and Rational Arithmetic Cryptographic Library*. `https://github.com/miracl/MIRACL`. 2018.

[MNT01] A. Miyaji, M. Nakabayashi, and S. Takano. "New Explicit Conditions of Elliptic Curve Traces for FR-Reduction". In: *IEICE Transactions on Fundamentals of Electronics, Com-*

munications and Computer Sciences 84 (2001), pp. 1234–1243.

[Mog+20] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger. “TPM-FAIL:TPM meets Timing and Lattice Attacks”. In: *The 29th USENIX Security Symposium*. 2020, pp. 2057–2073.

[Mon80] L. Monier. “Evaluation and comparison of two efficient probabilistic primality testing algorithms”. In: *Theoretical Computer Science* 12.1 (1980), pp. 97–108.

[Mon87] P. L. Montgomery. “Speeding the Pollard and elliptic curve methods of factorization”. In: *Mathematics of Computation* 48.177 (1987), pp. 243–264.

[Mor+15] H. Morita, J. C. Schuldt, T. Matsuda, G. Hanaoka, and T. Iwata. “On the security of the schnorr signature scheme and DSA against related-key attacks”. In: *ICISC 2015*. Springer. 2015, pp. 20–35.

[Mul+13] E. D. Mulder, M. Hutter, M. E. Marson, and P. Pearson. “Using Bleichenbacher’s Solution to the Hidden Number Problem to Attack Nonce Leaks in 384-Bit ECDSA”. In: *Cryptographic Hardware and Embedded Systems (CHES)*. 2013, pp. 435–452.

[Mur+12] C. Murdica, S. Guilley, J. Danger, P. Hoogvorst, and D. Naccache. “Same Values Power Analysis Using Special Points on Elliptic Curves”. In: *Constructive Side-Channel Analysis and Secure Design (COSADE) - Third International Workshop*. Vol. 7275. Lecture Notes in Computer Science. Springer, 2012, pp. 183–198.

[Nak08] S. Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: *Decentralized Business Review* (2008), p. 21260.

[Nem+17] 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: *24th ACM CCS*. 2017, pp. 1631–1648.

[NS02] P. Q. Nguyen and I. E. Shparlinski. “The Insecurity of the Digital Signature Algorithm with Partially Known Nonces”. In: *J. Cryptology* 15.3 (2002), pp. 151–176.

[NS03] P. Q. Nguyen and I. E. Shparlinski. “The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces”. In: *Designs, Codes and Cryptography* 30.2 (2003), pp. 201–217.

[NRS] J. Nick, T. Ruffing, and Y. Seurin. *MuSig2: Simple Two-Round Schnorr Multi-Signatures*. IACR Cryptology ePrint Archive, Report 2020/1261.

[Nic+20] J. Nick, T. Ruffing, Y. Seurin, and P. Wuille. “Musig-DN: Schnorr multi-signatures with verifiably deterministic nonces”. In: *27th ACM CCS*. 2020, pp. 1717–1731.

[NJP18] Y. Nir, S. Josefsson, and M. Pegourie-Gonnard. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier*. RFC 8422. RFC Editor, 2018, pp. 1–34. URL: `https://tools.ietf.org/html/rfc8422`.

[NIST06] National Institute for Standards and Technology. *Special Publication 800-89: Recommendation for Obtaining Assurances for Digital Signature Applications*. Standard. 2006.

[NIST00] 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`.

[NIST07] National Institute of Standards and Technology. *Security Requirements for Cryptographic Modules*. FIPS 140-2. 2007.

[NIST13] National Institute for Standards and Technology. *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION 186-4 Digital Signature Standard (DSS)*. Standard. 2013.

[Noe] S. Noether. *Ring Signature Confidential Transactions for Monero*. IACR Cryptology ePrint Archive, Report 2015/1098.

[Ora19] Oracle. *Java Card API 3.0.5, Classic Edition*. 2019. URL: `https://docs.oracle.com/javacard/3.0.5/api/index.html` (visited on 09/24/2021).

[Orm98] H. Orman. *The OAKLEY Key Determination Protocol*. RFC 2412. Nov. 1998. URL: `https://rfc-editor.org/rfc/rfc2412.txt`.

[Per+] G. C. C. F. Pereira, M. A. S. Jr, M. Naehrig, and P. S. L. M. Barreto. *A Family of Implementation-Friendly BN Elliptic Curves*. Cryptology ePrint Archive, Report 2010/429.

[Poe18] A. Poelstra. *Mimblewimble and Scriptless Scripts*. 2018. URL: `https://diyhpl.us/wiki/transcripts/realworldcrypto/2018/mimblewimble-and-scriptless-scripts/` (visited on 09/24/2021).

[PH78] S. Pohlig and M. Hellman. "An Improved Algorithm for Computing Logarithms over $GF(p)$ and Its Cryptographic Significance". In: *IEEE Transactions on Information Theory* 24.1 (1978), pp. 106–110.

[PS00] D. Pointcheval and J. Stern. "Security arguments for digital signatures and blind signatures". In: *Journal of cryptology* 13.3 (2000), pp. 361–396.

[PSY15] J. van de Pol, N. P. Smart, and Y. Yarom. "Just a Little Bit More". In: *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*. 2015, pp. 3–21.

[PHB02] T. Polk, R. Housley, and L. Bassham. *Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 3279. RFC Editor, 2002, pp. 1–27. URL: `https://tools.ietf.org/html/rfc3279`.

[Pol74] J. M. Pollard. "Theorems on factorization and primality testing". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 76.3 (1974), pp. 521–528.

[Pol75] J. M. Pollard. "A Monte Carlo method for factorization". In: *BIT Numerical Mathematics*. Vol. 15. Springer-Verlag, 1975, pp. 331–334.

[Pol78] J. M. Pollard. "Monte Carlo methods for index computation $\pmod p$". In: *Mathematics of Computation* 32.143 (1978), pp. 918–924.

[Pol93] J. M. Pollard. "Factoring with cubic integers". In: *The development of the number field sieve*. Springer-Verlag, 1993, pp. 4–10.

[Pom85] C. Pomerance. "The Quadratic Sieve Factoring Algorithm". In: *EUROCRYPT*. Springer-Verlag, 1985, pp. 169–182.

[PLK06] 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`.

[Por13] T. Pornin. *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*. RFC 6979. RFC Editor, 2013, pp. 1–79. URL: `https://tools.ietf.org/html/rfc6979`.

[Por19] T. Pornin. `https://crypto.stackexchange.com/questions/60420/what-does-the-special-form-of-the-base-point-of-secp256k1-allow`. 2019. (Visited on 09/24/2021).

[Qui+20] E. P. Quiroz, A. Cuno, W. R. Lovon, and E. Cruzado. "ECC usage on X. 509 digital certificates". In: *IEEE Engineering International Research Conference (EIRCON)*. 2020, pp. 1–4.

[Rab80] M. O. Rabin. "Probabilistic algorithm for testing primality". In: *Journal of Number Theory* 12 (1 1980), pp. 128–138.

[RCB16] J. Renes, C. Costello, and L. Batina. "Complete Addition Formulas for Prime Order Elliptic Curves". In: *EUROCRYPT, Proceedings, Part I*. 2016, pp. 403–428.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. RFC Editor, 2018, pp. 1–160. URL: `https://tools.ietf.org/html/rfc8446`.

[Roe+17] M. Roetteler, M. Naehrig, K. M. Svore, and K. Lauter. "Quantum resource estimates for computing elliptic curve discrete logarithms". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 241–270.

[RS07] K. Rubin and A. Silverberg. "Choosing the correct elliptic curve in the CM method". In: *Mathematics of Computation*. Vol. 79. AMS, 2007, pp. 545–561.

[Rya19a] K. Ryan. "Hardware-Backed Heist: Extracting ECDSA Keys from Qualcomm's TrustZone". In: *26th ACM CCS*. 2019, pp. 181–194.

[Rya19b] K. Ryan. "Return of the Hidden Number Problem. A Widespread and Novel Key Extraction Attack on ECDSA and DSA". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)* (2019), pp. 146–168.

[Sas+14] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. "Zerocash: Decentralized anonymous payments from bitcoin". In: *IEEE Symposium on Security and Privacy*. 2014, pp. 459–474.

[SA+98] T. Satoh, K. Araki, et al. "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves". In: *Rikkyo Daigaku sugaku zasshi* 47.1 (1998), pp. 81–92.

[Sch15] M. Schmid. *ECDSA-application and implementation failures*. 2015.

[Sch13] B. Schneier. *The NSA Is Breaking Most Encryption on the Internet*. 2013. URL: `https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html#c1675929` (visited on 09/24/2021).

[Sch91a] C. P. Schnorr. *Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system*. US Patent 4,995,082. 1991.

[Sch91b] C. P. Schnorr. "Efficient signature generation by smart cards". In: *Journal of Cryptology* 4.3 (1991), pp. 161–174.

[Sch01] C. P. Schnorr. "Security of blind discrete log signatures against interactive attacks". In: *International Conference on Information and Communications Security*. Springer. 2001, pp. 1–12.

[Sch89] C. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO*. Springer, 1989, pp. 239–252.

[Sch95] R. Schoof. "Counting points on elliptic curves over finite fields". In: *Journal de Théorie des Nombres de Bordeaux* (1995), pp. 219–254.

[Sco99] M. Scott. *Re: NIST annouces set of Elliptic Curves*. 1999. URL: `%7Bhttps://web.archive.org/web/20160313065951/https://groups.google.com/forum/message/raw?msg=sci.crypt/mFMukSsORmI/FpbHDQ6hM_MJ%7D` (visited on 09/24/2021).

[Sco15] M. Scott. *A new curve*. `https://moderncrypto.org/mail-archive/curves/2015/000449.html`. 2015. (Visited on 09/24/2021).

[Sed20] V. Sedlacek. *Examining and improving the security of elliptic curve cryptography [online]*. Advanced Master's thesis. 2020. URL: `%5Curl%7Bhttps://is.muni.cz/th/vzvjz/%7D` (visited on 09/24/2021).

[Sed+21] 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: *ASIACRYPT*. Lecture Notes in Computer Science. In preparation. Springer, 2021.

[SJS20] V. Sedlacek, J. Jancar, and P. Svenda. "Fooling primality tests on smartcards". In: *25th European Symposium on Research in Computer Security (ESORICS)*. Springer, 2020.

[Sed+19] V. Sedlacek, D. Klinec, M. Sys, P. Svenda, and V. Matyas. "I Want to Break Square-free: The $4p-1$ Factorization Method and Its RSA Backdoor Viability". In: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE)- Volume 2: SECRYPT,* INSTICC. SciTePress, 2019, pp. 25–36.

[Sem96] I. Semaev. "On computing logarithms on elliptic curves". In: *Discrete Mathematics and Applications* 6.1 (1996), pp. 69–76.

[Sem98] I. Semaev. "Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p". In: *Mathematics of Computation* 67.221 (1998), pp. 353–356.

[Sha79] A. Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[SSL14] 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`.

[Shi] M. Shirase. *Condition on composite numbers easily factored with elliptic curve method.* IACR Cryptology ePrint Archive, Report 2017/403.

[Sma99] N. P. Smart. “The discrete logarithm problem on elliptic curves of trace one”. In: *Journal of cryptology* 12.3 (1999), pp. 193–196.

[Sma03] N. P. Smart. “An Analysis of Goubin’s Refined Power Analysis Attack”. In: *Cryptographic Hardware and Embedded Systems (CHES)*. Vol. 2779. Lecture Notes in Computer Science. Springer, 2003, pp. 281–290.

[Sol11] 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.

[SG09] D. Stebila and J. Green. “Elliptic curve algorithm integration in the secure shell transport layer”. In: *RFC 5656* (2009).

[SM16] R. Susella and S. Montrasio. “A Compact and Exception-Free Ladder for All Short Weierstrass Elliptic Curves”. In: *Smart Card Research and Advanced Applications (CARDIS), Revised Selected Papers*. 2016, pp. 156–173.

[Sut11] A. V. Sutherland. “Computing Hilbert class polynomials with the Chinese remainder theorem”. In: *Mathematics of Computation*. Vol. 80. AMS, 2011, pp. 501–538.

[Sve19] P. Svenda. *JCAlgTest: Detailed analysis of cryptographic smart cards running with JavaCard platform*. 2019. URL: `https://www.fi.muni.cz/~xsvenda/jcalgtest/` (visited on 09/24/2021).

[Sve+16] P. Svenda, M. Nemec, P. Sekan, R. Kvasnovsky, D. Formanek, D. Komarek, and V. Matyas. “The Million-Key Question – Investigating the Origins of RSA Public Keys”. In: *The 25th USENIX Security Symposium*. 2016, pp. 893–910.

[Syt+16] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. “Keeping authorities "honest or bust" with decentralized witness cosigning”. In: *IEEE Symposium on Security and Privacy*. 2016, pp. 526–545.

[TT19] A. Takahashi and M. Tibouchi. “Degenerate fault attacks on elliptic curve parameters in OpenSSL”. In: *IEEE European Symposium on Security and Privacy*. 2019, pp. 371–386.

[SAGE19] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.9)*. 2019. URL: `https://www.sagemath.org`.

[Tra19] Trail of Bits. *ECDSA: Handle with Care*. 2019. URL: `https://blog.trailofbits.com/2020/06/11/ecdsa-handle-with-care/` (visited on 09/24/2021).

[Tuv+18] N. Tuveri, S. ul Hassan, C. Pereida Garcia, and B. B. Brumley. "Side-Channel Analysis of SM2: A Late-Stage Featurization Case Study". In: *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*. ACM, 2018, pp. 147–160.

[Val+18] L. Valenta, N. Sullivan, A. Sanso, and N. Heninger. "In Search of CurveSwap: Measuring Elliptic Curve Implementations in the Wild". In: *IEEE Symposium on Security and Privacy*. 2018, pp. 384–398.

[Wag02] D. Wagner. "A generalized birthday problem". In: *Annual International Cryptology Conference*. Springer. 2002, pp. 288–304.

[Was08] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. 2nd ed. Chapman & Hall/CRC, 2008.

[Weg96] P. Wegner. "Interoperability". In: *ACM Computing Surveys (CSUR)* 28.1 (1996), pp. 285–287.

[Wei+20] S. Weiser, D. Schrammel, L. Bodner, and R. Spreitzer. "Big Numbers - Big Troubles: Systematically Analyzing Nonce Leakage in (EC)DSA Implementations". In: *The 29th USENIX Security Symposium*. 2020, pp. 1767–1784.

[Wil82] H. C. Williams. "A $p+1$ Method of Factoring". In: *Mathematics of Computation*. Vol. 39. AMS, 1982, pp. 225–234.

[Wir00] 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.

[Wis] WiseKey. *Secure Microcontrollers*. URL: `https://www.wisekey.com/vaultic/secure-microcontrollers/` (visited on 09/24/2021).

[Woo+14] G. Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[WNT20a] P. Wuille, J. Nick, and A. Towns. *Schnorr signatures for secp256k1*. 2020. URL: `https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki` (visited on 09/24/2021).

[WNT20b] P. Wuille, J. Nick, and A. Towns. *Taproot: SegWit Version 1 Spending Rules*. 2020. URL: `https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki` (visited on 09/24/2021).

[WNT20c] P. Wuille, J. Nick, and A. Towns. *Validation of Taproot scripts*. 2020. URL: `https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki` (visited on 09/24/2021).

[YY97] A. L. Young and M. Yung. "Kleptography: Using Cryptography Against Cryptography". In: *EUROCRYPT*. 1997, pp. 62–74.

[ZDN19] ZDNet. *Minerva attack can recover private keys from smart cards, cryptographic libraries*. 2019. URL: `https://www.zdnet.com/article/minerva-attack-can-recover-private-keys-from-smart-cards-cryptographic-libraries/` (visited on 09/24/2021).

[ZLL12] F. Zhang, Q. Lin, and S. Liu. "Zero-Value Point Attacks on Kummer-Based Cryptosystem". In: *Applied Cryptography and Network Security (ACNS)- 10th International Conference*. Vol. 7341. Lecture Notes in Computer Science. Springer, 2012, pp. 293–310.

Appendices

A Author's publications

The author's publications relevant to the thesis follow.

- [Sed+19]: V. Sedlacek, D. Klinec, M. Sys, P. Svenda, and V. Matyas. "I Want to Break Square-free: The $4p - 1$ Factorization Method and Its RSA Backdoor Viability". In: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE)- Volume 2: SECRYPT*, INSTICC. SciTePress, 2019, pp. 25–36.
  Contribution: 43% (5 authors). The author was responsible for the theoretical analysis and improvements as well as the design of the models and experiments.

- [SJS20]: V. Sedlacek, J. Jancar, and P. Svenda. "Fooling primality tests on smartcards". In: *25th European Symposium on Research in Computer Security (ESORICS)*. Springer, 2020.
  Contribution: 43% (3 authors). The author was responsible for the methodology and experiments design, as well as for generating the testing parameters.

- [Jan+20]: J. Jancar, V. Sedlacek, P. Svenda, and M. Sys. "Minerva: The curse of ECDSA nonces; Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces". In: *Cryptographic Hardware and Embedded Systems (CHES)*. Ruhr-University of Bochum, 2020.
  Best Paper Award received. Contribution: 30% (4 authors). The author was responsible for the formalization of the vulnerability as well as for the theoretical improvements of the attack.

- [Sed+21]: 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: *ASIACRYPT*. Lecture Notes in Computer Science. In preparation. Springer, 2021.
  Contribution: 40% (4 authors). The author was responsible for the formalization, theoretical attacks, some of the experiments and the exceptional point classification.

Publications waiting to be reviewed

- *Resilience via Practical Interoperability of Multiparty Schnorr Signature Schemes*, submitted to ACNS 2022.

  Contribution: 30% (3 authors). The author was responsible for the formalization and security analysis.

- *DiSSECT: Distinguisher of Standard & Simulated Elliptic Curves via Traits*, submitted to PKC 2022.

  Contribution: 40% (3 authors). The author was responsible for the methodology, foundations of the framework and all the initial implementations.

B Fooling primality tests on smartcards

B.1 The Miller-Rabin primality test

The MR test [Mil75; Rab80] was one of the first practical primality tests and to this day remains very popular because of its simplicity and efficiency. In particular, we believe that if a low-resource device such as a smartcard (shortened as *card* for the rest of text) uses a primality test, MR is the most probable choice (perhaps followed by the Lucas test, which does not seem to be that widespread, and a Ballie-PSW test, which is a combination of these two), as most other tests are too resource-heavy.

However, the MR test cannot be used to prove that a number is prime; only compositeness can be proven. It relies on the fact that there exist no nontrivial roots of unity modulo a prime. More precisely, let n be the number we want to test for primality and let $n - 1 = 2^s d$, where d is odd. If n is prime, Fermat's Little Theorem implies that for any $1 \leq a < n$, we have either $a^d \equiv 1 \pmod{n}$ or $a^{2^i d} \equiv -1 \pmod{n}$ for some $0 \leq i < s$. By taking the contrapositive, if there is some $1 \leq a < n$ such that none of these congruences hold, then n is composite (and a is called a *witness of compositeness* for n). However, if at least one of the congruences holds, then we say that n is *pseudoprime with respect to base a* (or that a is a *non-witness of compositeness* for n, or also a *liar* for n). There is the Monier-Rabin bound [Mon80] for the number $S(n)$ of such bases (that are less than n): $S(n) \leq \frac{\varphi(n)}{4}$, where φ is the Euler totient function.

Since $\varphi(n) \approx n$ for large n, we get a practical upper bound for the number of inputs that pass the test for a given a. Thus if we repeat the test t times for random a's, the probability of fooling the MR test will be at most $(\frac{1}{4})^t$.

The fact that the a's were picked randomly is crucial for the guarantees above. If the bases are fixed and known in advance (as in [Alb+18]), it is possible to construct a pseudoprime (see Appendix B.2), i.e., a number that passes the test with respect to these bases.

B.2 Constructing pseudoprimes

We will briefly describe how to generate pseudoprimes having 3 prime factors with respect to given distinct prime bases $a_1, \dots, a_t$ according to [Alb+18] and [Arn95a], where more details can be found. The whole method can be summarised as follows:

1. Choose t odd prime bases $a_1 < \cdots < a_t$ (we always choose the first t smallest primes) and let $A := \{a_1, \dots, a_t\}$.

2. Let $k_1 = 1$ and choose distinct coprime $k_2, k_3 \in \mathbb{Z}$, $k_2, k_3 > a_t$ (see Table 4.1).

3. For each $a \in A$, compute the set S_a of primes p reduced modulo $4a$ s.t. $\left(\frac{a}{p}\right) = -1$. This can be done constructively by looping over values $x \in \{1, 2, \dots, 4a-1\}$ and adding x to S_a iff

   $$\left(\frac{x}{a}\right)(-1)^{(x-1)(a-1)/4} = -1$$

   (using quadratic reciprocity).

4. For each $a \in A$, compute the intersection $R_a := \bigcap_{j=1}^{3} k_j^{-1}(S_a + k_j - 1)$, where $k_j^{-1}(S_a + k_j - 1)$ denotes the set $\{k_j^{-1}(s + k_j - 1) \bmod 4a \mid s \in S_a\}$ for each $a \in A$. If any are empty, go back to step 2.

5. For each $a \in A$, randomly pick an element $r_a \in R_a$.

6. Using the CRT, find p_1 such that
   $p_1 \equiv k_3^{-1} \pmod{k_2}, p_1 \equiv k_2^{-1} \pmod{k_3}$ and $p_1 \equiv r_a \pmod{4a}$ for all $a \in A$.

7. Compute $p_2 = k_2(p_1 - 1) + 1$ and $p_3 = k_3(p_1 - 1) + 1$. If all p_1, p_2, p_3 are primes, then $p_1 p_2 p_3$ is pseudoprime with respect to all bases $a \in A$. Otherwise, go back to step 4 (or even 2 or 1 after a certain amount of time has passed).

If we take $a_1 = 2$ and enforce the condition $p_1 \equiv 3 \pmod 8$ (by slightly tweaking some steps above), the constructed pseudoprimes will meet the Monier-Rabin bound (maximizing the probability of passing the test for a random base choice) and will also pass the MR test for any composite base with no prime divisors greater than a_t [Alb+18].

Carmichael numbers are composite n that divide $a^{n-1} - 1$ for all $a \in \mathbb{Z}$ coprime to n. Equivalently, a composite integer n is a Carmichael number if and only if n is square-free, and $p - 1 \mid n - 1$ for all prime divisors p of n [Mon80]. The pseudoprimes generated in this way are automatically Carmichael numbers [Alb+18] and we are using this fact in Section 4.3.4.

B.3 Generated domain parameters

The domain parameters and scripts used to generate them and produce our results are available at `https://crocs.fi.muni.cz/papers/primality_esorics20`.

B.4 Examples of attacks

ECDSA/ECDH: Composite n. This case uses the 10-factor n parameters as specified in Appendix B.3. Such a smooth order of the curve allows for a direct application of the Pohlig-Hellman algorithm for computing discrete logarithms to obtain the private key.

Our Sage [SAGE19] script (Listing B.1) recovered the private key on a 256-bit curve in just about 7 seconds on an ordinary laptop. Computing such a discrete logarithm on a standard 256-bit curve is currently computationally infeasible.

ECDSA/ECDH: Composite p. This case uses the 10-factor p parameters as specified in Appendix B.3. Such a curve with composite p can be decomposed into ten much smaller curves modulo the prime divisors of p. On these curves, it is trivial to compute the discrete logarithm of the public key. The resulting discrete logarithm (and the private key) is then recovered via the CRT.

Our Sage script (Listing B.2) recovered the private key on a 256-bit curve in about 9 seconds on an ordinary laptop.

DSA/DH: Composite q In case of composite q in DSA/DH, the Pohlig-Hellman algorithm for computing discrete logarithms applies

```
from sage.groups.generic import discrete_log_rho
p  = 0x8b7dada7aa2173f4a3ed9139570386fd2b65eb9ed2232e749385df5532e8349d
k  = GF(p)
a  = k(0x1b27b49f431ab73930736bea17cee09d455a91997a986029807e399713a25ffd)
b  = k(0x6a5d9b63f85d937c868241fb54b5a4671556d46fd92aca1e20b312970b4e759f)
gx = 0x39494395fa2fa85ef2e6d441493e70b1adedaaf74360b9a9cc038c9897fbb42e
gy = 0x4b065332f5369883087e3943518b2da10cf9aa5e28a08f74968206bc2cc9b33e
n  = 27424609 * 33419179 * 37898257 * 39440263 * 49818481 * 52559371 \
     * 53216161 * 59617639 * 61332769 * 90393689

e = EllipticCurve([a, b]); e.set_order(n)
g = e(gx, gy)

privkey = randrange(0, n); pubkey = privkey * g

dlogs = []; mods = []
for factor, power in factor(n):
   mul = Integer(n/factor)
   dlog = discrete_log_rho(mul * pubkey, mul * g, factor, operation="+")
   dlogs.append(dlog); mods.append(factor)

result = CRT_list(dlogs, mods)
print(result == privkey)
```

Listing B.1: Simple key recovery via the Pohlig-Hellman algorithm

again. Our Sage script computed the private key of a public key using the 1024 bit DSA/DH parameters given in Appendix B.3 in 35 minutes on one Intel Xeon X7560 @ 2.26 GHz processor.

DSA/DH: Composite p**.** We have used the CADO-NFS [CADO17] implementation of the Number Field Sieve, to demonstrate the ease of computing the discrete logarithm of a public key using the 1024 bit DSA/DH parameters given in Appendix B.3. We computed the discrete logarithm in the order q subgroup of $\mathbb{Z}^*_{p_1}$ as it defined the smallest group of only 336 bits.

The computation took 70 minutes to recover the private key on three Intel Xeon X7560 @ 2.26 GHz processors (24 cores total), with total CPU time of 22 hours. Furthermore, this computation is generic for all public keys using the given domain parameters. The per-key computation is trivial and takes a few minutes at most.

Only one computation of the discrete logarithm on prime 1024 bit DSA/DH parameters is publicly known [Fri+17]. It used the fact that the prime was trapdoored and ran much faster than random

```
from sage.groups.generic import discrete_log
p  = 28260319 * 30235481 * 39172037 * 39191063 * 41237249 * 47624921 \
    * 51042223 * 71578097 * 77171399 * 107659879
a  = 0x84a477c83f88e833a49b562869f1553a4abbf7ffe29893ca272bf85b300cfe43
b  = 0x9567df38696eec2e80b4f43d056621c639938361b58260e12df91ac528c1ee2c
gx = 0x8e6da816bc1bd86cc7b9d393c08bcb9cdb44a016f44890419542ae43f34f9041
gy = 0x68e8c1d9e8ad5d256cfcf161c41090b5a7bbd3c7ca83f3cc185e289d8ce6ca0e
n  = 0xacc602e17e38aa923887566d83b95ec21b72368cc6a8565bd907f71d4824e67d

e = EllipticCurve(Integers(p), [a, b])
g = e(gx, gy)

privkey = randrange(0, n); pubkey = privkey * g
pub_x, pub_y = pubkey.xy()
pub_x, pub_y = lift(pub_x), lift(pub_y)

dlogs = []; mods = []
for factor, power in factor(p):
   kf = GF(factor)
   ef = EllipticCurve([kf(a), kf(b)])
   gf = ef(kf(gx), kf(gy))
   pf = ef(kf(pub_x), kf(pub_y))
   dlog = discrete_log(pf, gf, n, operation="+")
   dlogs.append(dlog); mods.append(gf.order())

result = CRT_list(dlogs, mods) % n
print(result == privkey)
```

Listing B.2: Simple key recovery via decomposition into smaller curves

parameters. Even then, it took two months on a large computation cluster, with a total CPU time of 385 CPU years.

C Minerva: The curse of ECDSA nonces

C.1 Additional figures

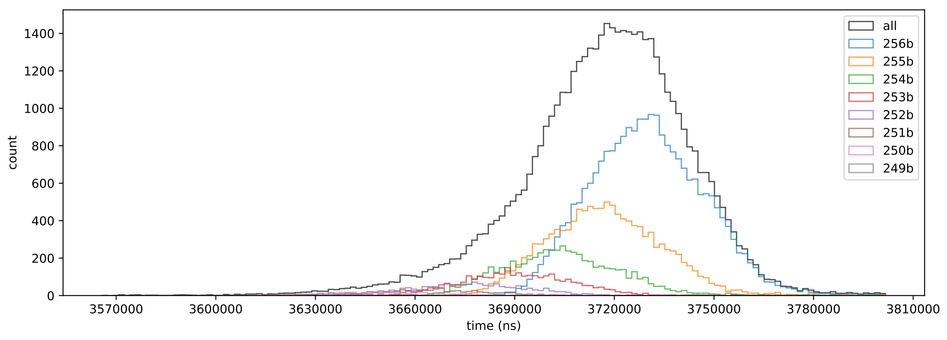

Figure C.12: Histogram of signing duration for different number of leading zeros of the random nonces on the `secp256r1` curve using the libgcrypt library.

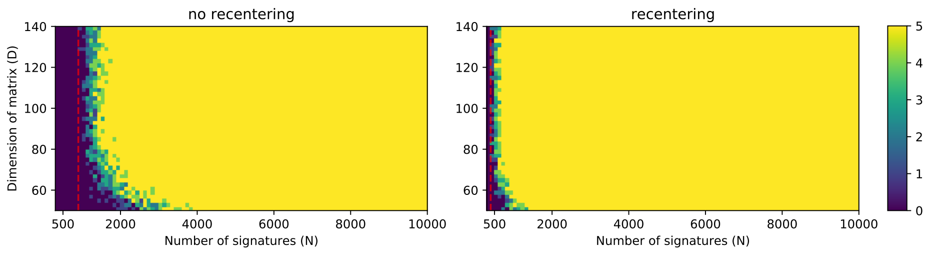

Figure C.13: Heatmap of success rate (out of 5 tries) when the true leading zero bits are used as bounds, thus showing the real minimum of the signatures required.

C.2 Impact of recentering with biased bounds

During our early experiments, we observed a counter-intuitive behavior of the recentering technique. Figure C.16 illustrates the strange behavior – the success rate decreases with the increasing number of signatures that represents the amount of gained information. This behavior occurs when the upper bounds $n/2^{l_i}$ for the nonces k_i are too conservative (too large upper bound, too small l_i), i.e., when number of errors is lowered.

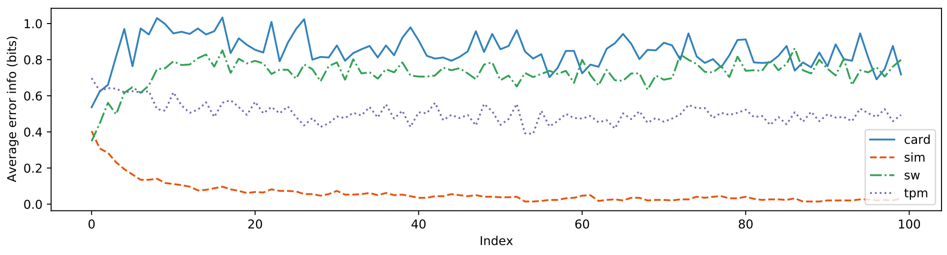

Figure C.14: Average amount of erroneous information per index with $d = 100$, when the geometric bounds are used with the four datasets. Here, average erroneous information means the average of error probability multiplied with the error depth (e.g. by how many bits was the bound overstated) at each index.

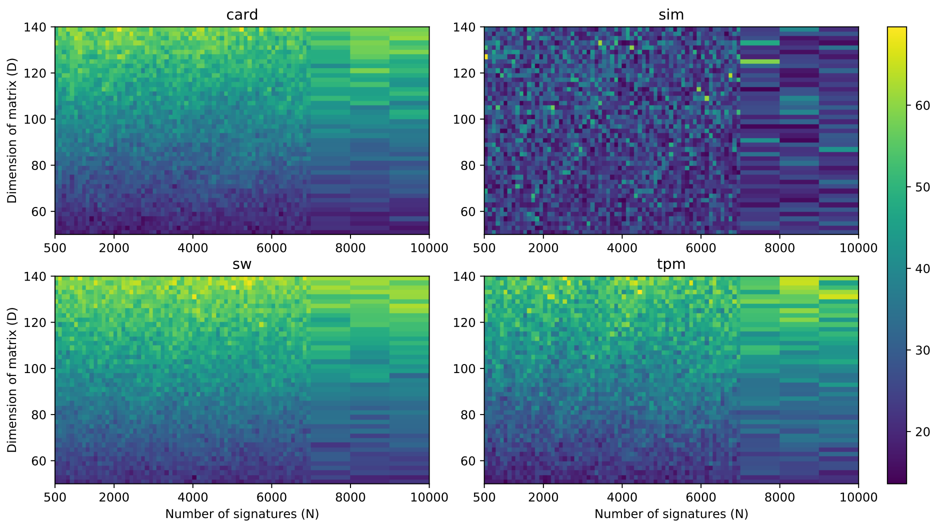

Figure C.15: Heatmap of average number of errors for the various datasets, when using geometric bounds.

We believe that the reason is a shift of the nonces caused by recentering when using biased bounds. The bounded nonces $\lfloor k_i \rceil_n < n/2^{l_i}$ are shifted towards the new bounds $n/2^{l_i+1}$ for biased bounds. The recentered nonces are bounded as follows: $|k_i - n/2^{l_i+1}|_n < n/2^{l_i+1}$. This shift and the distribution of k_i for $i = 50, N = 5000$ is illustrated by Figure C.17. Note that k_i represents the i-th nonce in the set of N nonces sorted by their bit-length. The figure shows that the recentering acts as a shift of the nonces towards the bound $n/2^{l_i+1}$ when a conservative bound l_i is used.

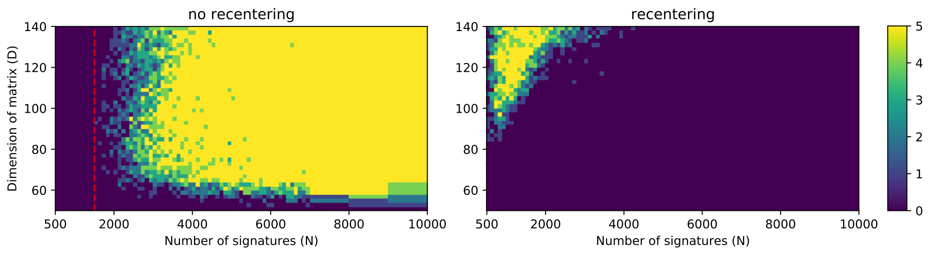

Figure C.16: Heatmap of success rate (out of 5 tries) for recentering with biased bounds. Geometric bounds were used as if $N = 4d$, which lead the bounds to be overly conservative for large N.

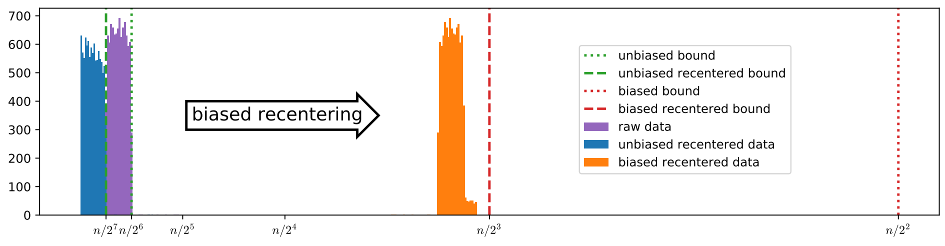

Figure C.17: Distributions of the $i = 50$-th out of $N = 5000$ nonces, sorted by bit-length, with both the biased bound $l_i = 2$ and unbiased bound $l_i = 6$. The effect of biased recentering manifests as a shift of the raw data to the right.

The overall conclusion is that recentering is usually beneficial, but only if the bounds are reasonable estimates of the actual ones, as seems to be the case for our geometric bounds.

D A unified approach to elliptic curve special-point-based attacks

D.1 Example: ZVP attack on window NAF scalar multiplication

To demonstrate the ZVP attack on a window NAF scalar multiplication algorithm (window size of 5), we used the **pyecsca** toolkit. We attack NIST's P-224 curve, which has no points suitable for RPA, so we use ZVP as the basis. Figure D.18 shows the basic setup of the attack, with `zvp_p0` being a point which zeros out an intermediate value when input into the `add-2016-rcb` formulas in projective coordinates, regardless of the second input point.

```
x = Mod(0xd83d7049c30873afc4893bf229d1c1ccb9eefd30f62ec71504b65fdc, p)
y = Mod(0x27c28fb63cf78c503b76c40dd62e3e32461102cf09d138eafb49a025, p)
z = Mod(1, p)

zvp_p0 = Point(coords, X=x, Y=y, Z=z)

def zvp_c(c):
    """Compute [c^-1]P_0"""
    return params.curve.affine_multiply(zvp_p0.to_affine(),
            int(Mod(c, params.order).inverse())).to_model(coords, params.curve)

def query(pt: Point) -> Tuple[int, List[int]]:
    """Query the implementation and observe the ZVP side-channel,
       i.e. at which iterations a zero in the intermediate value appeared.
       Returns the total number of formula applications and indexes
       where a zero in the intermediate value appeared."""
    with local(DefaultContext()) as ctx:
        mult.init(params, pt)
        mult.multiply(scalar)
    smult, subtree = ctx.actions.get_by_index([1])
    iterations = []
    for i, formula_action in enumerate(subtree):
        for intermediate in formula_action.intermediates.values():
            values = [j.value for j in intermediate]
            if 0 in values:
                iterations.append(i)
                break
    return len(subtree), iterations

def try_guess(guess) -> bool:
    """Test if we have the right private key."""
    return params.curve.affine_multiply(g, guess) == pubkey
```

Figure D.18: Setup for the ZVP window NAF attack.

```
wnaf_multiples = [1, 3, 5, 7, 9, 11, 13, 15, -1, -3, -5, -7, -9, -11, -13, -15]
all_iters = {}
for multiple in wnaf_multiples:
    rpa_point = zvp_c(multiple)
    num_iters, iters = query(rpa_point)
    all_iters[multiple] = (iters)
    print(multiple, num_iters, iters)
full = [0 for _ in range(num_iters)]
for multiple, iters in all_iters.items():
    for i in iters:
        full[i] = multiple
full_wnaf = [e for i, e in enumerate(full) if (not full[i - 1] != 0) or i in (0, 1)]
full_wnaf[0] = 1
```

Figure D.19: ZVP attack demonstration on window NAF scalar multiplication.

D.2 Example: unrolled formula

To analyze the ZVP and EPA attacks, we developed tooling for "unrolling" EFD formulas. The tooling expresses all the intermediate values in the formula as polynomials in the input variables. Figure D.20 gives an excerpt of the unrolled `add-2007-bl` formula in projective coordinates on short Weierstrass curves.

D.3 Example: output expressions

Appendix D.3 shows examples of unrolled Z_3 values on short Weierstrass curve with projective coordinates, which suggest the possible use of xDCP for EPA.

Formula	**Expression**
`add-2002-bj`	$Z3 = 2 * Z2^3 * Z1^3 * (Y2 * Z1 + Y1 * Z2)^3$
`add-1998-cmo-2`	$Z3 = Z2 * Z1 * (X2 * Z1^2 - X1 * Z2^2)$
`add-1998-cmo`	$Z3 = Z2 * Z1 * (X2 * Z1 - X1 * Z2)^3$
`madd-1998-cmo`	$Z3 = Z1 * (X2 * Z1 - X1)^3$
`mmadd-1998-cmo`	$Z3 = (-1) * (X1 - X2)^3$
`add-2007-bl`	$Z3 = 2^2 * Z2^3 * Z1^3 * (Y2 * Z1 + Y1 * Z2)^3$
`add-2002-bj-2`	$Z3 = 2 * Z2^3 * Z1^3 * (Y2 * Z1 + Y1 * Z2)^3$

Table D.3: Projective coordinate systems on short Weierstrass curves.

```
U1 = Z2 * X1
U2 = Z1 * X2
S1 = Z2 * Y1
S2 = Z1 * Y2
ZZ = Z2 * Z1
T  = X2*Z1 + X1*Z2
TT = (X2*Z1 + X1*Z2)^2
M  = Y2*Z1 + Y1*Z2
t0 = Z2^2 * Z1^2
t1 = a * Z2^2 * Z1^2
t2 = Z2 * Z1 * X2 * X1
t3 = X2^2*Z1^2 + X1*X2*Z1*Z2 + X1^2*Z2^2
R  = a*Z1^2*Z2^2 + X2^2*Z1^2 + X1*X2*Z1*Z2 + X1^2*Z2^2
F  = Z2 * Z1 * (Y2*Z1 + Y1*Z2)
L  = Z2 * Z1 * (Y2*Z1 + Y1*Z2)^2
LL = Z2^2 * Z1^2 * (Y2*Z1 + Y1*Z2)^4
t4 = Y2^2*Z1^3*Z2 + 2*Y1*Y2*Z1^2*Z2^2 + Y1^2*Z1*Z2^3 + X2*Z1 + X1*Z2
t5 = (Y2^2*Z1^3*Z2 + 2*Y1*Y2*Z1^2*Z2^2 + Y1^2*Z1*Z2^3 + X2*Z1 + X1*Z2)^2
t6 = Z2 * Z1 * (Y2*Z1 + Y1*Z2)^2 * (Y2^2*Z1^3*Z2 + 2*Y1*Y2*Z1^2*Z2^2 +
     Y1^2*Z1*Z2^3 + 2*X2*Z1 + 2*X1*Z2)
G  = 2 * Z2 * Z1 * (Y2*Z1 + Y1*Z2)^2 * (X2*Z1 + X1*Z2)
t7 = (a*Z1^2*Z2^2 + X2^2*Z1^2 + X1*X2*Z1*Z2 + X1^2*Z2^2)^2
t8 = 2 * (a*Z1^2*Z2^2 + X2^2*Z1^2 + X1*X2*Z1*Z2 + X1^2*Z2^2)^2
W  = 2 * (a^2*Z1^4*Z2^4 + 2*a*X2^2*Z1^4*Z2^2 + 2*a*X1*X2*Z1^3*Z2^3 +
     2*a*X1^2*Z1^2*Z2^4 + X2^4*Z1^4 + 2*X1*X2^3*Z1^3*Z2 - X2*Y2^2*Z1^4*Z2 +
     3*X1^2*X2^2*Z1^2*Z2^2 - 2*X2*Y1*Y2*Z1^3*Z2^2 - X1*Y2^2*Z1^3*Z2^2 +
     2*X1^3*X2*Z1*Z2^3 - X2*Y1^2*Z1^2*Z2^3 - 2*X1*Y1*Y2*Z1^2*Z2^3 +
     X1^4*Z2^4 - X1*Y1^2*Z1*Z2^4)
t9 = 2 * Z2 * Z1 * (Y2*Z1 + Y1*Z2) * (a^2*Z1^4*Z2^4 + 2*a*X2^2*Z1^4*Z2^2 +
     2*a*X1*X2*Z1^3*Z2^3 + 2*a*X1^2*Z1^2*Z2^4 + X2^4*Z1^4 + 2*X1*X2^3*Z1^3*Z2 -
     X2*Y2^2*Z1^4*Z2 + 3*X1^2*X2^2*Z1^2*Z2^2 - 2*X2*Y1*Y2*Z1^3*Z2^2 -
     X1*Y2^2*Z1^3*Z2^2 + 2*X1^3*X2*Z1*Z2^3 - X2*Y1^2*Z1^2*Z2^3 -
     2*X1*Y1*Y2*Z1^2*Z2^3 + X1^4*Z2^4 - X1*Y1^2*Z1*Z2^4)
X3 = 2^2 * Z2 * Z1 * (Y2*Z1 + Y1*Z2) * (a^2*Z1^4*Z2^4 + 2*a*X2^2*Z1^4*Z2^2 +
     2*a*X1*X2*Z1^3*Z2^3 + 2*a*X1^2*Z1^2*Z2^4 + X2^4*Z1^4 + 2*X1*X2^3*Z1^3*Z2 -
     X2*Y2^2*Z1^4*Z2 + 3*X1^2*X2^2*Z1^2*Z2^2 - 2*X2*Y1*Y2*Z1^3*Z2^2 -
     X1*Y2^2*Z1^3*Z2^2 + 2*X1^3*X2*Z1*Z2^3 - X2*Y1^2*Z1^2*Z2^3 -
     2*X1*Y1*Y2*Z1^2*Z2^3 + X1^4*Z2^4 - X1*Y1^2*Z1*Z2^4)
...
```

Figure D.20: An excerpt of an unrolled formula, `add-2007-bl` in projective coordinates on short Weierstrass curves.

E Interoperable Schnorr multisignatures

E.1 Multi-signature distributed key generation

The following multi-signature distributed key generation is based on the distributed key generation of Myst [Mav+17].

Smartcard i		**Central Party**
	$\xleftarrow{(\mathsf{keygen\text{-}init}, k)}$	
$x_i \leftarrow_{\$} \mathbb{Z}_q$ $X_i := [x_i]G$ $h_i := H(X_i)$		
	$\xrightarrow{(\mathsf{keygen\text{-}init}, h_i)}$	
		$\vec{h} := (h_1, \ldots, h_k)$
	$\xleftarrow{(\mathsf{keygen\text{-}reveal}, \vec{h})}$	
	$\xrightarrow{(\mathsf{keygen\text{-}reveal}, X_i)}$	
		$\vec{X} := (X_1, ..., X_k)$
	$\xleftarrow{(\mathsf{keygen\text{-}finalize}, \vec{X})}$	

if $\exists j \in \{1, \ldots, n\} : H(X_j) \neq h_j$
   then abort
 fi
 $X := \sum_{j=1}^{k} X_j$		

	$\xrightarrow{(\mathsf{keygen\text{-}finalize}, X)}$	

Figure E.21: Multi-signature distributed key generation

F DiSSECT: Distinguisher of Standard & Simulated Elliptic Curves via Traits

F.1 List of traits

Here we list all traits used to analyze curves, including their inputs and outputs, basic motivation and estimated time complexity expressed using known operations or algorithms. We use $\beta := log(n)$ for brevity.

a01
Input: Elliptic curve $E/\mathbb{F}_p$ and an integer r.
Output: 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 $n_1|n_2$ (in particular, $n_1 = 1$ for cyclic groups).
Motivation: The group structure determines the complexity of all general DLP attacks. Compared to the number of points (e.g. $n = n_1 n_2$), the output (n_1, n_2) is not an isogeny invariant.
Time complexity: Baby-step giant-step algorithm: $O(\sqrt{n})$.

a02
Input: Ordinary elliptic curve $E/\mathbb{F}_p$.
Output: The factorization of $D = t^2 - 4p = v^2 d_K$, where d_K is the discriminant of the endomorphism algebra of E.
Motivation: A large square factor of D implies non-random curve and an existence of multiple isogenies from the curve.
Time complexity: Factorization of a β-bit number.

a03
Input: Ordinary elliptic curve $E/\mathbb{F}_p$ and integer r.
Output: The factorization of the cardinality of the quadratic twist of $E(\mathbb{F}_{p^r})$.
Motivation: A smooth cardinality of a quadratic twist might allow attacks on some implementations.
Time complexity: Factorization of a β-bit number.

a04
Input: Elliptic curve $E/\mathbb{F}_p$ and integer k.
Output: The factorizations of $kn+1$, $kn-1$, where $n = \#E(\mathbb{F}_p)$.
Motivation: A generalization of [Che06].
Time complexity: Factorization of a β-bit number.

a05
Input: Elliptic curve $E/\mathbb{F}_p$ and prime l
Output: k_1,k_2,k_2/k_1, where k_1,k_2 are the smallest integers satisfying $E[l] \cap E(\mathbb{F}_{p^{k_1}}) \neq$ and $E[l] \subseteq E(\mathbb{F}_{p^{k_2}})$.
Motivation: Low k_1,k_2 might lead to computable pairings.
Time complexity: Up to 20 scalar multiplications on a β-bit curve.

a06
Input: Ordinary elliptic curve $E/\mathbb{F}_p$ and integer r
Output: The factorization of D_r/D_1 (which is a square), where $D_r = t_r^2 - 4p^r$ and t_r is the trace of Frobenius of $E/\mathbb{F}_{p^r}$.
Motivation: The prime factors of D_r/D_1 determine for which l does the l-isogeny crater of E grow in the r-th extension.
Time complexity: Factorization of a $t/2$-bit number.

a07
Input: Ordinary elliptic curve $E/\mathbb{F}_p$.
Output: The embedding degree complement, i.e., $\phi(l)/e$, where l is the generator order and e is the multiplicative order of $q \pmod{l}$.
Motivation: A low embedding degree might allow the MOV attack.
Time complexity: Computation of multiplicative order in a β-bit prime field.

a08
Input: Ordinary elliptic curve $E/\mathbb{F}_p$.
Output: Upper bound u [Dav80] and lower bound l [Col85] on the class number of the endomorphism algebra of E given by $u = \lceil \log(-d_K)\sqrt{-dk}/\pi \rceil$ and $l = \frac{1}{7000}\prod_{p|d_K*}(1 - \frac{\lfloor 2\sqrt{p} \rfloor}{p+1})\ln(d)$ where d_K is the absolute discriminant and $*$ indicates that the largest prime factor of d_K is omitted from the product.

Motivation: The class number is a classical invariant determining the number of curves sharing an endomorphism algebra.
Time complexity: Factorization of a β-bit number.

a12
Input: Elliptic curve $E/\mathbb{F}_p$ and prime l.
Output: $\phi(n)/m$ where m is the order of l in the multiplicative group $\mathbb{Z}_n^\times$.
Motivation: A small m might have implications for scalar multiplication.
Time complexity: Computation of multiplicative order in a β-bit prime field.

a22
Input: Elliptic curve $E/\mathbb{F}_p$ and prime l.
Output: The factorization of the l-th division polynomial.
Motivation: The factorization determines the structure of l-torsion in different extensions.
Time complexity: Factorization of a $(l^2 - 1)$-degree polynomial over a β-bit prime field.

a23
Input: Ordinary elliptic curve $E/\mathbb{F}_p$ and prime l.
Output: The depth of the l-volcano and the degree of the crater subgraph (i.e., 2 is the degree of a circle crater, 1 for a segment and 0 for a point; more precisely: the degree is $1 + (\frac{d_K}{l})$.)
Motivation: The volcano structure determines Frobenius action and transfers between levels is relevant for post-quantum cryptoanalysis.
Time complexity: Computing the Legendre symbol of a β-bit number modulo a small prime.

a24
Input: Elliptic curve $E/\mathbb{F}_p$ and prime l.
Output: $i_1, i_2, i_2/i_1$ where i_1, i_2 are the smallest integers such that there exists a $\mathbb{F}_{p^{i_1}}$-rational l-isogeny from E and there exist all $l + 1$ $\mathbb{F}_{p^{i_2}}$-rational l-isogenies.

Motivation: This determines l-isogeny volcano structure in different extension fields.
Time complexity: Up to 20 scalar multiplications on a β-bit curve.

a25
Input: Elliptic curve $E/\mathbb{F}_p$ and integer r.
Output: A factorization of the trace of Frobenius of $E/\mathbb{F}_{p^r}$.
Motivation: This value, loosely speaking, measures the "extent of supersingularity".
Time complexity: Factorization of a $(r \cdot \beta)$-bit number.

a28
Input: Elliptic curve $E/\mathbb{F}_p$ and small prime l.
Output: A number of roots of $\Phi_l(j(E), x)$ where Φ_l is the l-th modular polynomial.
Motivation: These roots correspond to l-isogenous curves.
Time complexity: Factorization of $(l+1)$-degree polynomial over a β-bit prime field.

a29
Input: Elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + ax + b$
Output: Torsion order of $E'(\mathbb{Q})$ where E' is is given by the same equation $y^2 = x^3 + ax + b$.
Motivation: Inspired by the lifting of ECDLP to curve over $\mathbb{Q}$.
Time complexity: Doud's algorithm: $O(\beta^3)$.

i04
Input: Elliptic curve $E/\mathbb{F}_p$ and integer k.
Output: A number of points on E with the Hamming weight of the x-coordinate equal to k.
Motivation: Inspired by [Bai+b].
Time complexity: Computing Binomial(β,k) Legendre symbols in a β-bit prime field.

i06
Input: Elliptic curve $E/\mathbb{F}_p$.
Output: The factorization of square-free parts of $4p-1$ and $4n-1$ where n is the order of the generator point of E

Motivation: Inspired by [Che02].
Time complexity: Factorization of a β-bit number.

i07
Input: Elliptic curve $E/\mathbb{F}_p$.
Output: The distance of n to the nearest power of 2 and the nearest multiple of 32 and 64.
Motivation: The first part is related to scalar multiplication bias when not using rejection sampling, the second is inspired by the paper [Wei+20].
Time complexity: Division in a β-bit prime field.

i08
Input: Elliptic curve $E/\mathbb{F}_p$ with generator G and integer k.
Output: The x-coordinate of $\frac{1}{k}G$.
Motivation: The strange behaviour of `secp{224,256}k1` for $k = 2$.
Time complexity: One scalar multiplication on a β-bit curve.

i13
Input: Elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + ax + b$.
Output: The value $r = \frac{a^3}{b^2}$.
Motivation: The value r is used for the generation of curves in various standards including X9.62, FIPS, SECG etc.
Time complexity: Division in a β-bit prime field.

i14
Input: Elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + ax + b$ where p has bit-length s.
Output: $a_{160} - b_{-160}$ where a_{160} are the $s - 160$ rightmost bits of a and b_{-160} are the $s - 160$ leftmost bits of b.
Motivation: The Brainpool curve generation method.
Time complexity: Subtraction in a β-bit prime field.

i15
Input: Elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + ax + b$.
Output: The parameters a, b.
Motivation: NUMS generation method causes small b.
Time complexity: Constant

F.2 List of standard curves

Name	Category	Form	Field type	Bitlength	Cofactor
`{Tweedledum, Tweedledee}`	`amicable`	W	Prime	255	1
`{Pallas, Vesta}`	`amicable`	W	Prime	255	1
`FRP256v1`	`anssi`	W	Prime	256	1
`M-{221,383,511}`	`barp`	M	Prime	{221, 383, 511}	8
`E-{222,382,521}`	`barp`	E	Prime	{222, 382, 521}	4
`BLS12-381`	`bls`	W	Prime	381	7.63e+37
`BLS12-446`	`bls`	W	Prime	446	2.68e+44
`BLS12-455`	`bls`	W	Prime	455	1.90e+45
`BLS12-638`	`bls`	W	Prime	638	4.94e+63
`BLS24-477`	`bls`	W	Prime	477	2.02e+28
`Bandersnatch`	`bls`	TE	Prime	255	4
`bn{158,190,224}`	`bn`	W	Prime	{158, 190, 224}	1
`bn{254,286,318}`	`bn`	W	Prime	{256, 286, 318}	1
`bn{350,382,414}`	`bn`	W	Prime	{350, 382, 414}	1
`bn{446,478,510}`	`bn`	W	Prime	{446, 478, 510}	1
`bn{542,574,606}`	`bn`	W	Prime	{542, 574, 606}	1
`bn638`	`bn`	W	Prime	638	1
`brainpoolP{160,192,224}r1`	`brainpool`	W	Prime	{160, 192, 224}	1
`brainpoolP{256,320,384}r1`	`brainpool`	W	Prime	{256, 320, 384}	1
`brainpoolP512r1`	`brainpool`	W	Prime	512	1
`brainpoolP{160,192,224}t1`	`brainpool`	W	Prime	{161, 192, 224}	1
`brainpoolP{256,320,384}t1`	`brainpool`	W	Prime	{256, 320, 384}	1
`brainpoolP512t1`	`brainpool`	W	Prime	512	1
`Curve25519`	`djb`	M	Prime	255	8
`Curve1174`	`djb`	W	Prime	251	4
`Ed25519`	`djb`	TE	Prime	255	8
`Curve41417`	`djb`	TE	Prime	414	8
`BADA55-R-256`	`djb`	W	Prime	256	1
`BADA55-VR-{224,256,384}`	`djb`	W	Prime	{224, 256, 384}	1
`BADA55-{VPR,VPR2}-224`	`djb`	W	Prime	224	1
`gost{256,512}`	`gost`	W	Prime	{256, 512}	1
`id-tc26-...mSet{A,B}`	`gost`	W	Prime	512	1
`id-Gost...{A,B,C}-ParamSet`	`gost`	W	Prime	256	1
`id-tc26-...mSet{A,C}`	`gost`	TE	Prime	{256, 512}	4
`Fp{224,256}BN`	`iso`	W	Prime	{224, 256}	1
`Fp{384,512}BN`	`iso`	W	Prime	{384, 512}	1
`ssc-{160,192,224}`	`miracl`	W	Prime	{160, 192, 224}	1
`ssc-{256,288,320}`	`miracl`	W	Prime	{256, 288, 320}	1
`ssc-{384,512}`	`miracl`	W	Prime	{384, 512}	1
`mnt1`	`mnt`	W	Prime	170	15337
`mnt2/{1,2}`	`mnt`	W	Prime	159	1
`mnt3/{1,2,3}`	`mnt`	W	Prime	160	1
`mnt4`	`mnt`	W	Prime	240	1
`mnt5/{1,2,3}`	`mnt`	W	Prime	240	1
`P-{192,224,256}`	`nist`	W	Prime	{192, 224, 256}	1
`P-{384,521}`	`nist`	W	Prime	{384, 512}	1
`K-163`	`nist`	W	Binary	163	2
`K-{233,283,409}`	`nist`	W	Binary	{233, 283, 409}	4
`K-571`	`nist`	W	Binary	571	4
`B-{163,233,283}`	`nist`	W	Binary	{163, 233, 283}	2
`B-{409, 571}`	`nist`	W	Binary	{409, 571}	2
`numsp{256,384,512}d1`	`nums`	W	Prime	{256, 384, 512}	1
`numsp{256,384,512}t1`	`nums`	TE	Prime	{256, 384, 512}	4

`ed-{254,256,384}-mont`	`nums`	TE	Prime	{254, 256, 384}	4
`ed-{382,510,512}-mont`	`nums`	TE	Prime	{382, 510, 512}	4
`ed-{255,383,511}-mers`	`nums`	TE	Prime	{255, 383, 511}	4
`w-{254,256,382}-mont`	`nums`	W	Prime	{254, 256, 382}	1
`w-{384,510, 512}-mont`	`nums`	W	Prime	{384, 510, 512}	1
`w-{255,383,511}-mers`	`nums`	W	Prime	{255, 383, 511}	1
`Oakley Group {3,4}`	`oakley`	W	Binary	{155, 185}	1
`SM2`	`oscca`	W	Prime	256	1
`Curve{22103,4417,67254}`	`other`	W	Prime	{221, 226, 382}	{8, 4, 4}
`Curve383187`	`other`	M	Prime	283	8
`Ed448`	`other`	TE	Prime	448	4
`Fp254BNa`	`other`	W	Prime	254	1
`Fp254n2BNa`	`other`	W	Extension	508	1.60e+76
`JubJub`	`other`	TE	Prime	255	8
`MDC201601`	`other`	E	Prime	256	4
`Ted37919`	`other`	TE	Prime	379	8
`E-3363`	`other`	E	Prime	336	8
`secp{112,128,160}r1`	`secg`	W	Prime	{112, 128, 160}	1
`secp{192,224,256}r1`	`secg`	W	Prime	{192, 224, 256}	1
`secp{384,512}r1`	`secg`	W	Prime	{384, 521}	1
`secp{112,128}r2`	`secg`	W	Prime	{112, 128}	4
`secp{160r2, 256k1}`	`secg`	W	Prime	{160, 256}	1
`secp{160,192,224}k1`	`secg`	W	Prime	{160, 192, 224}	1
`sect{113,131,163}r1`	`secg`	W	Binary	{113, 131, 163}	2
`sect{193,233,283}r1`	`secg`	W	Binary	{193, 233, 283}	2
`sect{409,571}r1`	`secg`	W	Binary	{409, 571}	2
`sect{113,131,163}r2`	`secg`	W	Binary	{113, 131, 163}	2
`sect{163k1, 193r2}`	`secg`	W	Binary	{163, 193}	2
`sect{233,239,283}k1`	`secg`	W	Binary	{233, 239, 283}	4
`sect{409,571}k1`	`secg`	W	Binary	{409, 571}	4
`wap-wsg-...wtls{1,4}`	`wtls`	W	Binary	113	2
`wap-wsg-...wtls{3,5}`	`wtls`	W	Binary	163	2
`wap-wsg-...wtls{6,8}`	`wtls`	W	Prime	112	1
`wap-wsg-...wtls{7,9}`	`wtls`	W	Prime	160	1
`prime192v{2,3}`	`x962`	W	Prime	192	1
`prime239v{1,2,3}`	`x962`	W	Prime	239	1
`c2pnb163v{1,2,3}`	`x962`	W	Binary	163	2
`c2pnb176w1`	`x962`	W	Binary	176	65390
`c2pnb208w1`	`x962`	W	Binary	208	65096
`c2tnb191v{1,2,3}`	`x962`	W	Binary	191	{2, 4, 6}
`c2tnb239v{1,2,3}`	`x962`	W	Binary	239	{4, 6, 10}
`c2pnb272w1`	`x962`	W	Binary	272	65286
`c2pnb304w1`	`x962`	W	Binary	304	65070
`c2pnb368w1`	`x962`	W	Binary	368	65392
`c2tnb359v1`	`x962`	W	Binary	359	76
`c2tnb431r1`	`x962`	W	Binary	431	10080
`c2onb191v{4,5}`	`x962`	W	Binary	191	{2, 8}
`c2onb239v{4,5}`	`x962`	W	Binary	239	{4, 6}
`ansix9p{192,256}r1`	`x962`	W	Prime	{192, 256}	1
`ansip{224,384,521}r1`	`x962`	W	Prime	{224, 384, 521}	1
`ansix9t163k1`	`x962`	W	Binary	163	2
`ansit163r2`	`x962`	W	Binary	163	2
`ansit{233,283}r1`	`x962`	W	Binary	{233, 283}	2
`ansit{409,571}r1`	`x962`	W	Binary	{409, 571}	2
`ansit{233,283}k1`	`x962`	W	Binary	{233, 283}	4
`ansit{409,571}k1`	`x962`	W	Binary	{409, 571}	4