A formula for disaster: a unified approach to elliptic curve special-point-based attacks

Vladimir Sedlacek[1,2][0000-0003-2409-661X],
Jesús-Javier Chi-Domínguez[3,4][0000-0002-9753-7263],
Jan Jancar[1][0000-0002-1864-0183], and Billy Bob Brumley[4][0000-0001-9160-0463]

[1] Masaryk University, Brno, Czech Republic
`{vlada.sedlacek, j08ny}@mail.muni.cz`
[2] Ca'Foscari University, Venice, Italy
[3] Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, United Arab Emirates
`jesus.dominguez@tii.ae`
[4] Tampere University, Tampere, Finland
`billy.brumley@tuni.fi`

Abstract. The Refined Power Analysis, Zero-Value Point, and Exceptional Procedure attacks introduced side-channel techniques against specific cases of elliptic curve cryptography. The three attacks recover bits of a static ECDH key adaptively, collecting information on whether 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 broader 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, classify all non-trivial exceptional points, and find them in new formulas. These results indicate the usefulness of our tooling, which we released publicly, for unrolling formulas and finding special points, and potentially for independent future work.

Keywords: elliptic curve cryptography · ECC · elliptic curve Diffie-Hellman · ECDH · side-channel analysis · Refined Power Analysis · RPA · Zero-Value Point attack · ZVP · Exceptional Procedure Attack · EPA · exceptional points

1 Introduction

Since the initial proposal of elliptic curve cryptography (ECC) by Koblitz [29] and Miller [32], 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 work, we unify and generalize the attacks, and systematically classify exceptional points in many widely used formulas.

Related work. In 2003, Goubin [21] introduced a new side-channel attack against implementations of ECC. Titled Refined Power Analysis (RPA), it uses a power side channel and the existence of points with a zero coordinate to steer an adaptive attack on implementations of the static elliptic curve Diffie-Hellman (ECDH) protocol. Smart [36] described effective countermeasures against RPA. Subsequently, Akishita and Takagi [1] 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 [41] modified the ZVP attack to target genus 2 curves, and Crépeau and Kazmi [15] proposed ZVP for elliptic curves over binary extension fields. Danger, Guilley, Hoogvorst, Murdica, and Naccache [16] gave new countermeasures against ZVP and RPA, while Martínez, Sadornil, Tena, Tomàs, and Valls [30] 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 [34] proposed the Same Value Analysis (SVA) attack, which tries to detect the repeated use of some finite field value via a side channel.

Izu and Takagi [27] analyzed the Brier and Joye [8] 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, without the use of fault induction. 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 [35] credit Bosma and Lenstra [7] for the only known complete formulas for prime order short Weierstrass curves, while Bernstein, Birkner, Joye, Lange, and Peters [3] and Hisil, Wong, Carter, and Dawson [25] proposed complete formulas for Twisted Edwards curves. The Explicit-Formulas Database (EFD) by Bernstein and Lange [4] 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 [2]. 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.

Contributions and outline. In this work, 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 3). Our approach leads to a new attack on windowed scalar multiplication algorithms (Section 3.5), and allows for clearer analysis of the attacks. 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 4). We then survey widely deployed software libraries, gaining insight into the practical implications of our analysis (Section 5). Finally, we draw our concluding remarks in Section 6. We released our code and data under an open-source license at github.com/crocs-muni/formula-for-disaster.

2 Background

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

$$E/\mathbb{F}_p\colon y^2 = x^3 + ax + b, \quad a, b \in \mathbb{F}_p, \quad 4a^3 + 27b^2 \neq 0. \tag{1}$$

The group $E(\mathbb{F}_p)$ consists of affine points $(x, y) \in \mathbb{F}_p^2$ satisfying (1) together with the neutral element $\mathcal{O}$, corresponding to 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 $[-k]P$ by $-[k]P$. The *order of a point* $P \in E(\mathbb{F}_p)$ is defined as the smallest positive integer k such that $[k]P = \mathcal{O}$. We refer to points of order dividing k as the k-torsion points. For typical cryptographic applications, $E(\mathbb{F}_p)$ has cardinality $n = h \cdot q$, where q is prime and $h \in \{1, 2, 4, 8\}$; h is called the *cofactor*.

The scalar point multiplication mapping $P \mapsto [k]P$ can also be computed by using the *division polynomial* ψ_k [39]: 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

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

All of these polynomials are considered modulo the curve equation (1). For simplicity, we denote $m_k(x) := \frac{\phi_k(x)}{\psi_k^2(x)}$, then for all $k_1, k_2, i \in \mathbb{Z}$, we have $(m_{k_1} \circ m_{k_2})(x) = m_{k_1 \cdot k_2}(x)$, $m_{k_1}(x) = m_{-k_1}(x)$ and $m_{k_1}(x) = m_{\pm k_1 + in}(x)$.

2.1 Curve models and their zero-coordinate points

For a short Weierstrass curve E_W over $\mathbb{F}_p$ given by Equation (1), 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 $(0, \pm\sqrt{b}) \in E_W/\mathbb{F}_p$ [22]. Any elliptic curve can be converted to the short Weierstrass model.

Montgomery. The Montgomery model of an elliptic curve [33, 14] is

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

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 [3] is

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

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 [18, 5] is

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

When using `yz` or `yzsquared` 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 $a_T = 1$).

2.2 Point coordinates and addition formulas

In practice, we mostly work with non-affine coordinates[5], 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 [33]) or y-only (Edwards models [10]) coordinates, assuming the difference of the input points is known. Table 1 lists the non-affine coordinates present in EFD.

[5] 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 [12, 9, 4, 35]	$(xZ: yZ: Z)$	$(0: 1: 0)$	21
	jacobian [11, 12, 23, 22, 31]	$(xZ^2: yZ^3: Z)$	$(1: 1: 0)$	36
	modified [12, 4]	$(xZ^2: yZ^3: Z: aZ^4)$	$(1: 1: 0: 0)$	4
	w12 with $b = 0$ [13]	$(xZ: yZ^2: Z)$	$(1: 0: 0)$	2
	xyzz	$(xZ^2: yZ^3: Z^2: Z^3)$	$(1: 1: 0: 0)$	6
	xz [9, 26]	$(xZ: Z)$	$(1: 0)$	22
E_M	xz [33]	$(xZ: Z)$	$(1: 0)$	8
E_T	projective [3]	$(xZ: yZ: Z)$	$(0: 1: 1)$	3
	extended [24]	$(xZ: yZ: xyZ: Z)$	$(0: 1: 0: 1)$	18
	inverted [3, 25]	$\left(\frac{Z}{x}: \frac{Z}{y}: Z\right)$	none	3
E_E	projective [5, 24, 25]	$(xZ: yZ: Z)$	$(0: c: 1)$	12
	inverted [6, 25]	$\left(\frac{Z}{x}: \frac{Z}{y}: Z\right)$	none	6
	yz [20]	$\left(yZ\sqrt{d}: Z\right)$	$\left(\sqrt{d}: 1\right)$	6
	yzsquared [20]	$\left(y^2Z\sqrt{d}: Z\right)$	$\left(\sqrt{d}: 1\right)$	6

Table 1. Non-affine coordinates analyzed in this work, and the quantity of corresponding EFD formulas. Note that the conversion from `xz`, `yz`, and `yzsquared` coordinates to affine is not unique, and that both `yz` and `yzsquared` assume $c = 1$.

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 curve satisfying the assumptions of the formula. Unified and complete 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 [35].

2.3 Explicit-Formulas Database

The Explicit-Formulas Database (EFD) by Bernstein and Lange [4] 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[6] with

[6] https://github.com/crocs-muni/efd

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, 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 SageMath, 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 4.

2.4 Scalar multiplication algorithms

During an ECDH key exchange, all scalar multiplications use a single scalar and the multiplied point is the public key of the other party, which is unknown before the computation. This excludes the use of heavy pre-computations like comb-based methods. Following Jancar [28], 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 [22]. 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 [33] 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, the window is called *sliding* [22]. 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 work, 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.

2.5 Side-channel attack countermeasures

To mitigate side-channel attacks on ECC, including those discussed in this work, several countermeasures were developed. Here we show those relevant to 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) [17]. 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 select 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 [17]. 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. Such randomization uses either an isomorphic or an isogenous curve [17, 36].

2.6 The Refined Power Analysis and Zero-Value Point attacks

Goubin's Refined Power Analysis (RPA) [21] 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[7] 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

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

discussed the existence of zero coordinate points in Section 2.1. 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 [16]) 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 countermeasures successfully thwart the RPA attack, as well as Smart's curve randomization via isogenies [36], while point randomization or curve randomization via isomorphisms do not, since the zero point coordinate does not get randomized. Unlike [19], the attack does not require fault injection.

More generally, Akishita's and Takagi's Zero-Value Point (ZVP) attack [1] 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 1 for an example of the intermediate values and Figure 4 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 [30], 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.

2.7 Exceptional procedure attacks

In practice, scalar multiplication uses non-affine point representations (shown in Table 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. Exceptional Procedure Attacks (EPA) are based on finding a pair of points P and Q such that the final conversion of $P + Q = \textbf{add}(P, Q)$ fails, because the expression being inverted is zero. The implementation then either throws an error, or produces an obviously detectable output [27]. Such points are called exceptional w.r.t. a given formula; see Section 4 for a more precise definition and classification of all non-trivial exceptional points for EFD formulas.

3 A unified approach to the attacks

The attacks introduced in Section 2.6 and Section 2.7 have a lot in common. In this section, we build a common framework that captures them as special cases.

3.1 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 2.4 and displayed in Figure 1, the formulas take as input some multiples of P, which depend on k and S.

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

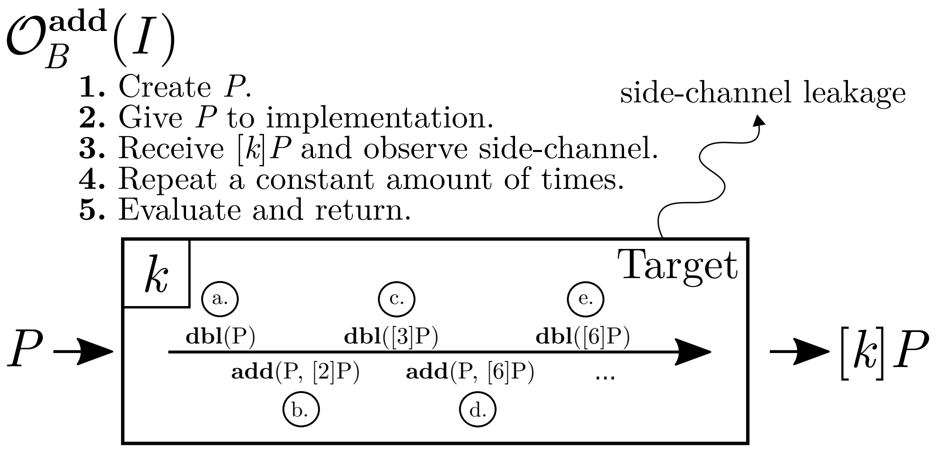
$$\mathcal{O}^{\mathcal{F}}_{B,U}(I) := \begin{cases} 1 \text{ if } I \text{ was input into } \mathcal{F} \text{ during the } S(k,P) \text{ computation;} \\ 0 \text{ if } I \text{ was not input into } \mathcal{F}; \\ \perp \text{ if the oracle could not determine the result,} \end{cases}$$

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 also 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 also 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 3.4 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.



$\mathcal{O}_B^{\mathbf{add}}(I)$
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.
dbl(P)
dbl([3]P)
dbl([6]P)
add(P, [2]P)
add(P, [6]P)
...
b.
d.
P
$[k]P$

Fig. 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.

3.2 The dependent coordinates problem

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

Following the notation introduced in Section 2, for the rest of this section we fix a prime $p \geq 3$ and an elliptic curve $E/\mathbb{F}_p$ given[8] 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 $E(\mathbb{F}_p)$ with prime order q. Recall that m_k is the x-coordinate of the rational multiplication-by-k function on E. Furthermore, let

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

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 1 (DCP: the 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.

[8] 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.

Solving DCP via xDCP. The following lemma cancels the occurrences of Y_1 and Y_2 (if any) thanks to squarings and reductions modulo the curve equation.

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 $f_0 + f_1Y_1 + f_2Y_2 + f_{12}Y_1Y_2$ 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

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

□

Indeed, Lemma 1 allows us to only consider xDCP instead of DCP for the remainder of the paper. 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. 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 [1], 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.

3.3 Solving xDCP

The basic strategy to solve xDCP described in [1] 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 $\left(\begin{smallmatrix} 1 & k \\ 0 & q \end{smallmatrix}\right)$ using the Lagrange-Gauss algorithm, and take l as its first coordinate. Indeed, 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.

3.4 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 3.1, 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 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 an $\mathcal{O}^{\mathbf{add}}_{T,([e],\_)}$ 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 [1]. 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[9]. 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.

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 [21]. This leads to an easy case of the (x)DCP, discussed in Section 3.2, 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 2.7, 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[10] into the scalar multiplication, which will fail if it computes $P + [k]P$, enabling us to construct an $\mathcal{O}^{\mathbf{add}}_{B,([1],[k])}$. Note that this is a boolean oracle, as with the error side channel we can only detect that the mapping back to affine coordinates failed, and not during which iteration the zero was introduced.

3.5 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.

The attack requires that the addition formula in question has an intermediate value which depends only on one of the operands, so that zeroing it out leads

[9] 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.
[10] The homogeneity of f allows us to only consider affine representations.

to an easy case of the DCP as mentioned in Section 3.2. Together with an appropriate side channel, this allows the attacker to construct an $\mathcal{O}^{\mathbf{add}}_{T,([e],\_)}$ 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 iterations where the e-th multiple was added, the attacker varies over all values e in the set of precomputed multiples, based on the algorithm. In this way, the attacker recovers the window digits and thus 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.

4 Classifying the exceptional points

While many EFD formulas [4] 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** [28] (**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 (e.g., over 5-bit fields) of all relevant types, trying all input point pairs for all the analyzed addition formulas, comparing the result to the correct affine output. 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 to large ones (though they can reveal patterns for manual analysis).

Manual analysis. To find the sufficient and necessary conditions that classify all the exceptional points, we resort to manual inspection. Compared to breadth-focused fuzzing, it dives deeper, taking 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.

We carefully went through all 111 addition formulas and 42 differential addition/ladder formulas for the E_W, E_M, E_T, E_E models in the EFD[11], and investigated when the expressions by which we divide during the conversion to affine coordinates could be zero[12]. Namely, for addition this amounted to studying the conditions $X_3 = 0$ or $Y_3 = 0$ for (twisted) inverted Edwards coordinates,

[11] Some of the formulas are just adaptations for specific coefficients (e.g. $a = -3$ for E_W), mixed additions, etc.

[12] Occasionally omitting the cases where the result is the neutral element.

$ZZ_3 = 0$ or $ZZZ_3 = 0$ for short Weierstrass `xyzz` coordinates[13], 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 instead study when the outputs Z_4 and Z_5 were equal to 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. The expressions we refer to are present in our data release (see Section 5.3), though we also provide example expressions that illustrate the process in Table 2.

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 2. Jacobian coordinate outputs on short Weierstrass curves.

4.1 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 moreover $P \neq \pm Q$, 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 coords. For short Weierstrass curves, non-triviality implies $x_1 \neq x_2$. Moreover, we do not need to consider the expression corresponding to Z_3 in the formulas by Renes et al. [35], as Bosma and Lenstra [7] 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, which implies $x_i = X_i, y_i = Y_i$. With this in mind, there is only a single factor that could possibly be zero in the

[13] ZZ_i and ZZZ_i are variables whose values equal Z_i^2 and Z_i^3 throughout the computation, respectively.

studied Z_3 expressions, namely $(y_1 + y_2)^3$. This factor is present in all variants of the Brier-Joye [8] formulas (`add-2002-bj`) and Bernstein-Lange [4] formulas (`add-2007-bl`), illustrated in Algorithm 1. Note that $(y_1+y_2)^3 = 0$ is equivalent to $y_1 = -y_2$, which implies

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

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 [27] previously identified this family for the `add-2002-bj` case, but not for the `add-2007-bl` one.

Algorithm 1 Point addition formula `add-2007-bl` in projective coordinates

```
              $E/\mathbb{F}_p\colon y^2 = x^3 + ax + b$,
Require:      $P = (X_1 : Y_1 : Z_1)$,
              $Q = (X_2 : Y_2 : Z_2)$
Ensure: $(X_3 : Y_3 : Z_3) = P + Q$
 1: $U_1 = X_1 \cdot Z_2$
 2: $U_2 = X_2 \cdot Z_1$
 3: $S_1 = Y_1 \cdot Z_2$
 4: $S_2 = Y_2 \cdot Z_1$
 5: $ZZ = Z_1 \cdot Z_2$
 6: $T = U_1 + U_2$
 7: $TT = T^2$
 8: $M = S_1 + S_2$
 9: $t_0 = ZZ^2$
10: $t_1 = a \cdot t_0$
11: $t_2 = U_1 \cdot U_2$
12: $t_3 = TT - t_2$
13: $R = t_3 + t_1$
14: $F = ZZ \cdot M$
15: $L = M \cdot F$
16: $LL = L^2$
17: $t_4 = T + L$
18: $t_5 = t_4^2$
19: $t_6 = t_5 - TT$
20: $G = t_6 - LL$
21: $t_7 = R^2$
22: $t_8 = 2 \cdot t_7$
23: $W = t_8 - G$
24: $t_9 = F \cdot W$
25: $X_3 = 2 \cdot t_9$
26: $t_{10} = 2 \cdot W$
27: $t_{11} = G - t_{10}$
28: $t_{12} = 2 \cdot LL$
29: $t_{13} = R \cdot t_{11}$
30: $Y_3 = t_{13} - t_{12}$
31: $t_{14} = F^2$
32: $t_{15} = F \cdot t_{14}$
33: $Z_3 = 4 \cdot t_{15}$
```

(Twisted) Edwards: projective, extended, inverted coords. Let $E_{a,d}$: $ax^2 + y^2 = 1 + dx^2y^2$ be a (twisted) Edwards curve[14] (cf. Section 2; 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.

[14] 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:*

$$\begin{aligned} x_1x_2y_1y_2 &\neq 0, & (2)\\ dx_1x_2y_1y_2 &\neq \pm 1, & (3)\\ y_1y_2 &\neq -ax_1x_2, & (4)\\ x_1y_2 &\neq x_2y_1, & (5)\\ x_1y_2 &\neq -x_2y_1, & (6)\\ y_1y_2 &\neq ax_1x_2. & (7)\\ x_1y_1 &\neq \pm x_2y_2. & (8) \end{aligned}$$

Proof. (2) follows from the fact that neither P nor Q are 4-torsion. Hisil et al. [25] (Theorem 1, Corollary 1) prove (3), (4) and (5).

Now consider the addition law from [3]:

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

Assume that either (6) or (7) is false. Since the denominators are nonzero by (3), 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 [25]:

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

Assume that (8) is false. Since the denominators are nonzero by (4) and (5), 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.

4.2 Exceptional points for differential addition and ladders

Recall from Section 2.3 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).

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 3. Formulas [4, 26, 9] with non-trivial exceptional points. The projective coordinates apply to all formula versions: $a = -1$, $a = -3$ and general a.

Short Weierstrass: xz coords. 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 $Z_1 = Z_2 = Z_3 = 1$ 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 $x_2 = x_3$, $x_2^3 + ax_2 + b = 0$, and $x_1 = 0$. Only the latter corresponds[15] 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`, `{d/l}add-2002-it-3`, and their mixed variants, as well as `mdadd-2002-bj`, `m{l/d}add-2002-bj-2`, and `mladd-2002-bj-3`.

Montgomery: xz coords. 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 2 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, yzsquared coords. 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 `yzsquared` coordinates,

[15] Note that x_1 does not directly affect X_4 nor X_5.

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

$$\begin{aligned}(r-1)(1 + ry_2^2 + ry_3^2 + r^2y_2^2y_3^2) &= (r+1)(1 - ry_2^2 - ry_3^2 + r^2y_2^2y_3^2)\\ -2 + 2r^2y_2^2 + 2r^2y_3^2 - 2r^2y_2^2y_3^2 &= 0\\ r^2y_3^2(1 - y_2^2) &= 1 - r^2y_2^2. \end{aligned} \tag{9}$$

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

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

and since (9) is symmetric, analogical arguments yield

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

Thus the only case left to consider is $x_2^2y_3^2 = x_3^2y_2^2 = \frac{1}{r^2}$. But then we have $(1 + dx_2x_3y_2y_3)(1 - dx_2x_3y_2y_3) = 1 - r^4x_2^2x_3^2y_2^2y_3^2 = 0$, which is impossible for non-trivial exceptional points by (3) in Lemma 2.

5 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.

5.1 Impact on cryptographic libraries

We examined the EC arithmetic implementations in 15 popular open-source cryptographic libraries. Table 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, which we did not examine.

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. There is thus nothing stopping the developer from misusing the formulas, as the formula papers or the EFD give no warning. 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[16] 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, Sullivan, Sanso, and Heninger [38, Section 7.2] uncovered the analogous flaw in this code. There were no corresponding flaws in point doubling. Mozilla issued CVE-2017-7781[17] 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. [37] 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 4.2). One month passed between merging the broken implementation and the fix[18], switching to `ladd-2002-it-4`. The discovery[19] was mostly luck – during standardization, GOST curves utilized generators with the smallest possible x-coordinate.

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. [40] 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)
```

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

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 is not constant-time. 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[20] 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[21], 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 4 methodology, we discovered[22] 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.

5.2 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 3.5 allows the attacker to recover the full scalar. Thus, this shows that windowed methods of scalar multiplication are somewhat more vulnerable

[20] https://boringssl.googlesource.com/boringssl/+/12d9ed670da3edd64ce8175c
[21] https://pypi.org/project/fastecdsa/
[22] https://github.com/AntonKueltz/fastecdsa/pull/58

Library	**Operation**	**Scalar multiplier**	**Coordinates**	**Formulas**
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 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.

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 EM side channel, but the toolkit simulates the computation down to individual finite field operations, and produces the side-channel output (i.e., whether a zero occurred during computation). Appendix A shows the attack code 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.

5.3 Tooling

We released all of our code and data under an open-source license, as an extension to the **pyecsca** project[23]. This includes tooling for unrolling EFD formulas, helping analyze exceptional cases, and automatically construct ZVP points (note that Akishita and Takagi [1] construct them manually), as well as improvements to (x)DCP solving (Section 3.3). These tools can be used proactively in the future, analyzing formulas about to be used in new implementations, rather than analyzing existing implementations and finding vulnerabilities.

5.4 Reverse engineering

Another application of our techniques is in reverse engineering black-box implementations of ECC, as suggested in [28]. Many side-channel attacks critically depend on the attacker having detailed knowledge of the target's 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 using 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.

6 Conclusion

In this work, 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 3.5). 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

[23] https://github.com/crocs-muni/formula-for-disaster

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 5 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 4 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 3.

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.

Acknowledgments. The authors would like thank Marek Sys for helpful consultations. This project has initially received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 804476), and it has been made possible in part by a grant from the Cisco University Research Program Fund, an advised fund of Silicon Valley Community Foundation. This project started when J. J. Chi-Domínguez was a postdoctoral researcher at Tampere University. V. Sedlacek and J. Jancar were supported by Czech Science Foundation project GA20-03426S. J. Jancar was also supported by Red Hat Czech and V. Sedlacek by the Ph.D. Talent Scholarship - funded by the Brno City Municipality.

A 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 demonstrate the attack on NIST's P-224 curve, which has no points suitable for RPA. Figure 2 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.

B 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 4 gives an excerpt of the unrolled `add-2007-bl` formula in projective coordinates on short Weierstrass curves.

```
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():
            if 0 in [j.value for j in intermediate]:
                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
```

Fig. 2. 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
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
```

Fig. 3. ZVP attack demonstration on window NAF scalar multiplication algorithm.

```
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
...
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)
...
```

Fig. 4. An excerpt of an unrolled formula, `add-2007-bl` in projective coordinates on short Weierstrass curves.

References

1. Akishita, T., Takagi, T.: Zero-value point attacks on elliptic curve cryptosystem. In: Boyd, C., Mao, W. (eds.) Information Security, 6th International Conference, ISC 2003, Bristol, UK, October 1-3, 2003, Proceedings. LNCS, vol. 2851, pp. 218–233. Springer (2003), https://doi.org/10.1007/10958513_17
2. Belyavsky, D., Brumley, B.B., Chi-Domínguez, J., Rivera-Zamarripa, L., Ustinov, I.: Set it and forget it! Turnkey ECC for instant integration. In: ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020. pp. 760–771. ACM (2020), https://doi.org/10.1145/3427228.3427291
3. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: Vaudenay, S. (ed.) Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings. LNCS, vol. 5023, pp. 389–405. Springer (2008), https://doi.org/10.1007/978-3-540-68164-9_26
4. Bernstein, D.J., Lange, T.: Explicit-Formulas database (EFD) (2007), https://www.hyperelliptic.org/EFD/
5. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Kurosawa, K. (ed.) Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings. LNCS, vol. 4833, pp. 29–50. Springer (2007), https://doi.org/10.1007/978-3-540-76900-2_3
6. Bernstein, D.J., Lange, T.: Inverted Edwards coordinates. In: Boztas, S., Lu, H. (eds.) Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 17th In-

ternational Symposium, AAECC-17, Bangalore, India, December 16-20, 2007, Proceedings. LNCS, vol. 4851, pp. 20–27. Springer (2007), https://doi.org/10.1007/978-3-540-77224-8_4
7. Bosma, W., Lenstra, Jr., H.W.: Complete systems of two addition laws for elliptic curves. J. Number Theory 53(2), 229–240 (1995), https://doi.org/10.1006/jnth.1995.1088
8. Brier, E., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings. LNCS, vol. 2274, pp. 335–345. Springer (2002), https://doi.org/10.1007/3-540-45664-3_24
9. Brier, E., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings. LNCS, vol. 2274, pp. 335–345. Springer (2002), https://doi.org/10.1007/3-540-45664-3_24
10. Castryck, W., Galbraith, S.D., Farashahi, R.R.: Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. IACR Cryptol. ePrint Arch. 2008(218) (2008), http://eprint.iacr.org/2008/218
11. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Adv. in Appl. Math. 7(4), 385–434 (1986), https://doi.org/10.1016/0196-8858(86)90023-0
12. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In: Ohta, K., Pei, D. (eds.) Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings. LNCS, vol. 1514, pp. 51–65. Springer (1998), https://doi.org/10.1007/3-540-49649-1_6
13. Costello, C., Lange, T., Naehrig, M.: Faster pairing computations on curves with high-degree twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings. LNCS, vol. 6056, pp. 224–242. Springer (2010), https://doi.org/10.1007/978-3-642-13013-7_14
14. Costello, C., Smith, B.: Montgomery curves and their arithmetic - the case of large characteristic fields. J. Cryptographic Engineering 8(3), 227–240 (2018), https://doi.org/10.1007/s13389-017-0157-6
15. Crépeau, C., Kazmi, R.A.: An analysis of ZVP-attack on ECC cryptosystems. IACR Cryptol. ePrint Arch. 2012(329) (2012), https://eprint.iacr.org/2012/329
16. Danger, J., Guilley, S., Hoogvorst, P., Murdica, C., Naccache, D.: Dynamic countermeasure against the zero power analysis. In: IEEE International Symposium on Signal Processing and Information Technology, Athens, Greece, December 12-15, 2013. pp. 140–147. IEEE Computer Society (2013), https://doi.org/10.1109/ISSPIT.2013.6781869
17. Danger, J., Guilley, S., Hoogvorst, P., Murdica, C., Naccache, D.: A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. J. Cryptographic Engineering 3(4), 241–265 (2013), https://doi.org/10.1007/s13389-013-0062-6
18. Edwards, H.M.: A normal form for elliptic curves. Bull. Amer. Math. Soc. (N.S.) 44(3), 393–422 (2007), https://doi.org/10.1090/S0273-0979-07-01153-6
19. Fan, J., Gierlichs, B., Vercauteren, F.: To infinity and beyond: Combined attack on ECC using points of low order. In: Preneel, B., Takagi, T. (eds.) Cryptographic

Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. LNCS, vol. 6917, pp. 143–159. Springer (2011), https://doi.org/10.1007/978-3-642-23951-9_10
20. Gaudry, P.: Variants of the Montgomery form based on Theta functions. Toronto (November 2006) (2006)
21. Goubin, L.: A refined power-analysis attack on elliptic curve cryptosystems. In: Desmedt, Y. (ed.) Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings. LNCS, vol. 2567, pp. 199–210. Springer (2003), https://doi.org/10.1007/3-540-36288-6_15
22. Hankerson, D., Menezes, A., Vanstone, S.: Guide to elliptic curve cryptography. Springer Professional Computing, Springer (2004), https://doi.org/10.1016/s0012-365x(04)00102-5
23. Hasegawa, T., Nakajima, J., Matsui, M.: A practical implementation of elliptic curve cryptosystems over GF(p) on a 16-bit microcomputer. In: Imai, H., Zheng, Y. (eds.) Public Key Cryptography, First International Workshop on Practice and Theory in Public Key Cryptography, PKC '98, Pacifico Yokohama, Japan, February 5-6, 1998, Proceedings. LNCS, vol. 1431, pp. 182–194. Springer (1998), https://doi.org/10.1007/BFb0054024
24. Hisil, H., Carter, G., Dawson, E.: New formulae for efficient elliptic curve arithmetic. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings. LNCS, vol. 4859, pp. 138–151. Springer (2007), https://doi.org/10.1007/978-3-540-77026-8_11
25. Hisil, H., Wong, K.K., Carter, G., Dawson, E.: Twisted Edwards curves revisited. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings. LNCS, vol. 5350, pp. 326–343. Springer (2008), https://doi.org/10.1007/978-3-540-89255-7_20
26. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. In: Naccache, D., Paillier, P. (eds.) Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings. LNCS, vol. 2274, pp. 280–296. Springer (2002), https://doi.org/10.1007/3-540-45664-3_20
27. Izu, T., Takagi, T.: Exceptional procedure attack on elliptic curve cryptosystems. In: Desmedt, Y. (ed.) Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings. LNCS, vol. 2567, pp. 224–239. Springer (2003), https://doi.org/10.1007/3-540-36288-6_17
28. Jancar, J.: PYECSCA: Reverse-engineering black-box Elliptic Curve Cryptography implementations via side-channels. Master's thesis, Masaryk University, Brno, Czechia (2020), https://is.muni.cz/th/fjgay/
29. Koblitz, N.: Elliptic curve cryptosystems. Math. Comp. 48(177), 203–209 (1987), https://doi.org/10.2307/2007884
30. Martínez, S., Sadornil, D., Tena, J., Tomàs, R., Valls, M.: On Edwards curves and ZVP-attacks. Appl. Algebra Eng. Commun. Comput. 24(6), 507–517 (2013), https://doi.org/10.1007/s00200-013-0211-2
31. Meloni, N.: New point addition formulae for ECC applications. In: Carlet, C., Sunar, B. (eds.) Arithmetic of Finite Fields, First International Workshop, WAIFI

2007, Madrid, Spain, June 21-22, 2007, Proceedings. LNCS, vol. 4547, pp. 189–201. Springer (2007), https://doi.org/10.1007/978-3-540-73074-3_15

32. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings. LNCS, vol. 218, pp. 417–426. Springer (1985), https://doi.org/10.1007/3-540-39799-X_31
33. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comp. 48(177), 243–264 (1987), https://doi.org/10.2307/2007888
34. Murdica, C., Guilley, S., Danger, J., Hoogvorst, P., Naccache, D.: Same values power analysis using special points on elliptic curves. In: Schindler, W., Huss, S.A. (eds.) Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings. LNCS, vol. 7275, pp. 183–198. Springer (2012), https://doi.org/10.1007/978-3-642-29912-4_14
35. Renes, J., Costello, C., Batina, L.: Complete addition formulas for prime order elliptic curves. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I. LNCS, vol. 9665, pp. 403–428. Springer (2016), https://doi.org/10.1007/978-3-662-49890-3_16
36. Smart, N.P.: An analysis of Goubin's refined power analysis attack. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings. LNCS, vol. 2779, pp. 281–290. Springer (2003), https://doi.org/10.1007/978-3-540-45238-6_23
37. Tuveri, N., ul Hassan, S., Pereida García, C., Brumley, B.B.: Side-channel analysis of SM2: A late-stage featurization case study. In: Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018. pp. 147–160. ACM (2018), https://doi.org/10.1145/3274694.3274725
38. Valenta, L., Sullivan, N., Sanso, A., Heninger, N.: In search of CurveSwap: Measuring elliptic curve implementations in the wild. In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018. pp. 384–398. IEEE (2018), https://doi.org/10.1109/EuroSP.2018.00034
39. Washington, L.C.: Elliptic curves: number theory and cryptography. Discrete Mathematics and its Applications, Chapman and Hall/CRC, second edn. (2008), https://doi.org/10.1201/9781420071474
40. Weiser, S., Schrammel, D., Bodner, L., Spreitzer, R.: Big numbers - big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations. In: Capkun, S., Roesner, F. (eds.) 29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020. pp. 1767–1784. USENIX Association (2020), https://www.usenix.org/conference/usenixsecurity20/presentation/weiser
41. Zhang, F., Lin, Q., Liu, S.: Zero-value point attacks on Kummer-based cryptosystem. In: Bao, F., Samarati, P., Zhou, J. (eds.) Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings. LNCS, vol. 7341, pp. 293–310. Springer (2012), https://doi.org/10.1007/978-3-642-31284-7_18