Decompose and conquer: ZVP attacks on GLV curves

Vojtěch Suchánek[1], Vladimír Sedláček[2], and Marek Sýs[1]

[1] Masaryk University, Czech Republic
[2] Rutgers University, New Jersey
{vojtechsu, vlada.sedlacek, syso}@mail.muni.cz

Abstract. While many side-channel attacks on elliptic curve cryptography can be avoided by coordinate randomization, this is not the case for the zero-value point (ZVP) attack. This attack can recover a prefix of static ECDH key but requires solving an instance of the dependent coordinates problem (DCP), which is open in general. We design a new method for solving the DCP on GLV curves, including the Bitcoin secp256k1 curve, outperforming previous approaches. This leads to a new type of ZVP attack on multiscalar multiplication, recovering twice as many bits when compared to the classical ZVP attack. We demonstrate a 63% recovery of the private key for the interleaving algorithm for multiscalar multiplication. Finally, we analyze the largest database of curves and addition formulas with over 14 000 combinations and provide the first classification of their resistance against the ZVP attack.

Keywords: dependent coordinates problem · elliptic curve cryptography· GLV curve · side-channel attacks · ZVP attack

1 Introduction

Elliptic curve cryptography (ECC) provides several efficient schemes suitable for memory-constrained devices such as smartcards, but it can fall prey to side-channel attacks (SCAs) [19]. These exploit extra information leaked from the device during the execution of the protocol – e.g., timing, power consumption, or electromagnetic radiation. While most basic SCAs against ECC can be prevented by coordinate randomization, this is not the case for the refined power-analysis attack (RPA) [15], the exceptional procedure attack (EPA) [17] or the most general one – the zero-value point (ZVP) attack [1]. The ZVP attack repeatedly forces a zero intermediate value during the computations of an addition formula in the ECDH protocol to recover the upper bits of a static private key. For each bit of the private key, the attacker needs to solve an instance of the dependent coordinates problem (DCP) [24]. The difficulty of the DCP depends on the implemented addition formulas, but generally it remains an open problem to find an efficient algorithm for the DCP. If the implemented addition formula contains only hard DCP problems, only a few bits of the private key can be recovered. As a result, existing progress on the ZVP attacks focuses on addition formulas containing easy instances of DCP [1, 22, 24].

Meanwhile, Gallant, Lambert, and Vanstone [14] introduced GLV curves equipped with efficiently computable endomorphisms, which allow us to replace single-scalar multiplication with a faster multiscalar multiplication using two scalars. Extending this idea often led to record-breaking ECC implementations [13, 20, 16, 6]. A popular example of a GLV curve is `secp256k1` [9], which is used in the Bitcoin Core, or several of the popular pairing-friendly curves, such as BLS12-381, used in Zcash [7]. The ZVP attack was designed for single-scalar multiplication, and it is unclear whether multiscalar multiplication algorithms are more or less vulnerable to this attack. Additionally, while it is believed that endomorphisms do not significantly help to solve the discrete logarithm problem [5], their effect on DCP has not been studied. With this motivation, we present the following **contributions**:

– We develop new ZVP attacks on the classical multiscalar multiplication algorithms (interleaving and the Straus-Shamir trick), focusing on hard DCP problems. Our attacks use a fast endomorphism to recover twice as many bits when compared to the classical ZVP attack (Section 4). Additionally, for some implementations of the interleaving algorithm, our ZVP attack is able to recover 63% of the private key on average.
– We classify all the addition formulas from the Explicit-Formulas Database [4] with respect to the difficulty of the underlying DCP problems (Section 3). We show that the EPA attack cannot be used on the `secp256k1` curve, and the ZVP attack on all but one of the addition formulas in EFD supporting `secp256k1` must solve hard instances of DCP. We provide a semi-automatic tool for the classification of formulas with respect to other curves as well.

2 Background

Throughout the paper, $p \geq 5$ is a prime number, and $\mathbb{F}_p$ is a finite field of size p. Define an elliptic curve E over $\mathbb{F}_p$ in the short Weierstrass model by the equation

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

The set of solutions over $\mathbb{F}_p$ forms a finite abelian group $E(\mathbb{F}_p)$ with a neutral element ∞ and cardinality $n = \#E(\mathbb{F}_p) \approx p$. For any point $P \in E(\mathbb{F}_p)$, denote its coordinates by x_P, y_P. The addition on E of two affine points $R = P + Q$ is defined by the following rational functions:

$$x_R = s^2 - x_P - x_Q, \quad y_R = s(x_P - x_R) - y_P,$$

where $s = \frac{y_P - y_Q}{x_P - x_Q}$ if $P \neq Q$ and $s = \frac{3x_P^2 + a}{2y_P}$ otherwise. This is well-defined unless $P = -Q$ or $Q = P$, $2P = \infty$, in which case $R = \infty$.

We can naturally define scalar multiplication (SM) $d \cdot P$ as d-fold addition $P + \cdots + P$. There are several algorithms for computing SM; generally, they iterate over the digits[3] of the scalar d, evaluating formulas for point addition

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

($\mathsf{add}(P, Q)$) and doubling ($\mathsf{dbl}(P)$). For more efficiency, several popular SM algorithms also use differential addition formulas (i.e., $P+Q = \mathsf{dadd}(P-Q, P, Q)$) and ladder formulas (i.e., $(2P, P+Q) = \mathsf{ladd}(P-Q, P, Q)$). One of the popular SM algorithms is the double-and-add with the non-adjacent form (NAF) (Algorithm 1), in which $d^{(i)} \in \{-1, 0, 1\}$.

To avoid a costly inversion, formulas are defined in non-affine coordinate systems. For instance, the following is a projective add formula from [10]:

$$(x_3 : y_3 : z_3) = (vA : u(v^2x_1z_2 - A) - v^3y_1z_2 : v^3z_1z_2) \qquad (1)$$
$$u = y_2z_1 - y_1z_2, \quad v = x_2z_1 - x_1z_2, \quad A = u^2z_1z_2 - v^3 - 2v^2x_1z_2$$

for a projective sum $(x_3 : y_3 : z_3)$ of two points $(x_1 : y_1 : z_1)$, $(x_2 : y_2 : z_2)$. The Explicit Formulas Database (EFD) [4] is the largest public database of formulas for multiple coordinate systems and curve models ([24] provides a good overview of the database). Among the short Weierstrass and others, it supports the (Twisted) Edwards model [3] and the Montgomery model [21].

Algorithm 1: Double-and-add.

```
Input: $P \in E$, $m$-digit scalar $d$ in the NAF representation.
Output: $kP$
1 $Q \leftarrow \infty$
2 for $i = m-1$ down to $0$ do
3     $Q \leftarrow \mathsf{dbl}(Q)$
4     if $d^{(i)} = 1$ then $Q \leftarrow \mathsf{add}(Q, P)$
5     if $d^{(i)} = -1$ then $Q \leftarrow \mathsf{add}(Q, -P)$
6 return $Q$
```

2.1 GLV curves

A GLV curve [14] is an elliptic curve E equipped with an efficiently computable endomorphism ϕ for speeding up SM. An endomorphism $\phi : E \to E$ is defined by rational maps r_1, r_2: $(x, y) \mapsto (r_1(x, y), r_2(x, y))$ and satisfies $\phi(\infty) = \infty$. Coefficients of r_1, r_2 lie in $\mathbb{F}_p$, and r_1 can be expressed purely in the x-coordinate (in the Weierstrass model). If the group $E(\mathbb{F}_p)$ has a prime order n, then ϕ acts on $E(\mathbb{F}_p)$ as SM by some $\lambda \in \mathbb{Z}$. Any scalar $d \in \mathbb{Z}$ can be then decomposed using the extended Euclidean algorithm as

$$d = d_0 + d_1\lambda \pmod{n}, \quad d_0, d_1 \approx \sqrt{n}.$$

The computation of kP can then be carried out through multiscalar multiplication (MSM) as $(d_0 + d_1\lambda)P = d_0P + d_1\phi(P)$. This is usually done using the interleaving method (as in the libsecp256k1 library [29]) or the Straus-Shamir [2] trick. Both methods process the two SMs d_0P, $d_1\phi(P)$ at the same time, which eliminates half of the dbl operations when compared to simple SM dP (e.g., using Algorithm 1). The difference between the two methods is that at the beginning, the Straus-Shamir trick precomputes $P \pm \phi(P)$, while the interleaving precomputes small multiples of the points P and $\phi(P)$ separately. Details differ depending on the representation of the scalars d_0, d_1.

Algorithm 2 is the interleaving method combined with the regular w-NAF representation as designed in [18] and used in libsecp256k1 for $w = 5$ (also proposed by Brumley [8] for OpenSSL). In the regular w-NAF representation, the digits of the scalar take the odd values from $\{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$. Since none of the digits of the private key are ever zero in this representation, the number of add and dbl operations is fixed, providing a SCA protection (e.g., against cache timing attacks). Note that in each iteration of Algorithm 2, computing $Q + d_0^{(i)}P + d_1^{(i)}\phi(P)$ requires two additions. There are two natural options to order them, either $(Q + d_0^{(i)}P) + d_1^{(i)}\phi(P)$ or $Q + (d_0^{(i)}P + d_1^{(i)}\phi(P))$. The second option requires temporary space to store $d_0^{(i)}P + d_1^{(i)}\phi(P)$, but it can use faster formulas for addition since both the precomputed points $d_0^{(i)}P, d_1^{(i)}P$ can be in the affine form. For instance, this approach with the modified `add-2009-bl` addition formula can save one finite field squaring compared to the first option [4]. The first variant is used in libsecp256k1, but we will consider both variants of the interleaving algorithm, as their distinction will be relevant for our attacks.

Algorithm 2: Interleaving Multiscalar Multiplication.

```
  Input: $P \in E$, endomorphism $\phi$, $l$-digit regular $w$-NAF scalars $d_0, d_1$
  Output: $d_0 + d_1\phi(P)$
1 Store $kP$, $k\phi(P)$ for $k \in \{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$
2 $Q \leftarrow \infty$
3 for $i = l - 1$ down to $0$ do
4     for $j = 0$ to $w - 1$ do
5         $Q \leftarrow \mathsf{dbl}(Q)$
6     $Q \leftarrow \mathsf{add}(Q, d_0^{(i)}P)$          /* Alternatively: $R \leftarrow \mathsf{add}(d_0^{(i)}P, d_1^{(i)}\phi(P))$ */
7     $Q \leftarrow \mathsf{add}(Q, d_1^{(i)}\phi(P))$     /*                $Q \leftarrow \mathsf{add}(Q, R)$                */
8 return Q
```

Algorithm 3 is the Straus-Shamir trick combined with the GLV-SAC representation. Faz-Hernández, Longa, and Sánchez [12] designed this representation to achieve a constant number of dbl and add operations for SCA protection. The GLV-SAC representation of two scalars d_0, d_1 is a signed representation satisfying $(d_0^{(i)}, d_1^{(i)}) \in \{(1,0), (-1,0), (1,1), (-1,-1)\}$. In particular, $(d_0^{(i)}, d_1^{(i)}) \neq (0,0)$, which means that add is computed in each iteration.

The most prominent example of a GLV curve is the secp256k1 curve used in the Bitcoin core:

$$E/\mathbb{F}_p : y^2 = x^3 + 7, \quad p = 2^{256} - 2^{32} - 977,$$

Algorithm 3: Straus-Shamir trick.

```
   Input: $P \in E$, endomorphism $\phi$, $l$-digit GLV-SAC scalars $d_0, d_1$
   Output: $d_0 + d_1\phi(P)$
1  Store $g_0P + g_1\phi(P)$ for $(g_0, g_1) \in \{(1,1), (-1,-1), (1,0), (-1,0)\}$
2  $Q \leftarrow \infty$
3  for $i = l-1$ down to $0$ do
4      $Q \leftarrow \mathsf{dbl}(Q)$
5      $Q \leftarrow \mathsf{add}(Q, d_0^{(i)}P + d_1^{(i)}\phi(P))$
6  return $Q$
```

which is a member of the family of short Weierstrass curves with $a = 0$ and $p = 1 \pmod 3$. Such curves admit an endomorphism $\phi : (x, y) \mapsto (\beta x, y)$, where $\beta \in \mathbb{F}_p^\times$ is an element of order 3. The endomorphism can be computed using only one multiplication (by β) in $\mathbb{F}_p$. Moreover, it acts on $E(\mathbb{F}_p)$ as λ, which satisfies the quadratic equation $\lambda^2 + \lambda + 1 = 0$. This family of GLV curves, including the `secp256k1` curve, is the focus of this work.

2.2 ZVP attack

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

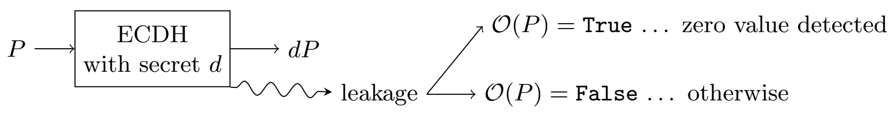

P
ECDH
with secret d
dP
leakage
O(P) = True . . . zero value detected
O(P) = False . . . otherwise

Fig. 1: Construction of the ZVP oracle $\mathcal{O}$.

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

It remains to explain how to construct such points P that invoke the zero intermediate value during add. The attacker uses the fact that in a formula for add, any intermediate value can be expressed as a polynomial f in the coordinates of the input points Q and P (or $-P$) to the add – one can see that in the add formula in (1). The goal is, therefore, to find P such that the coordinates of $\pm P$ and Q are the roots of f, where Q is a known multiple of P. This problem is an instance of the so-called *dependent coordinates problem* (DCP):

Definition 1. *Let $E/\mathbb{F}_p : y^2 = x^3 + ax + b$ be a curve. Given a polynomial $f \in \mathbb{F}_p[x_1, x_2, y_1, y_2]$ and scalars $k, g \in \mathbb{Z}$, find a point $P \in E(\mathbb{F}_p)$ such that $f(x_Q, x_G, y_Q, y_G) = 0$, where $Q = kP$, $G = gP$. We denote this problem*[4] *as $DCP_f(k, g)$. An instance with $g = 1$, $G = P$ is denoted as $DCP_f(k)$.*

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

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

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

There are two attacks that can be considered special cases of the ZVP attack called Refined Power Analysis (RPA) attack [15] and Exceptional Procedure attack (EPA) [17]. The RPA attack forces a coordinate of a point to be zero during the computation of the scalar multiplication. The DCP in this case is $\mathrm{DCP}_f(k)$ for either $f = y_1$ or $f = x_1$. The RPA attack is independent of the choice of the formula for add or dbl and relies purely on the existence of points with zero coordinates. All points with zero coordinates in the usual curve models have low order (≤ 4) and should not appear during the scalar multiplication – with the exception of points $(0, y)$ in the Weierstrass model [24]. The EPA attack takes a similar approach as RPA but also assumes that an incorrect computation

[4] The original formulation of DCP in [24] is with $g = 1$ for the simple binary representation. This is a generalization for more SM algorithms and scalar representations.

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

All ZVP attacks can be thwarted using scalar randomization, as the attacker cannot solve the DCP for an unknown randomized scalar and cannot prepare the input point. However, the RPA attack is resistant to projective coordinate randomization as $(0 : y : z) = (0 : ry : rz)$ for any $r \in \mathbb{F}_p$. ZVP and EPA attacks are resistant to coordinate randomization on the condition that the roots of the DCP polynomial f do not change under coordinate randomization. This is almost always the case for addition formulas in EFD (see Section 3).

2.3 Solving DCP

Given a $\mathrm{DCP}_f(k, g)$ problem, one can replace the polynomial $f \in \mathbb{F}_p[x_1, x_2, y_1, y_2]$ with $f' \in \mathbb{F}_p[x_1, x_2]$, by removing any occurrences of y_1, y_2 using the curve equation [24]. We will, therefore, assume that already $f \in \mathbb{F}_p[x_1, x_2]$. In general, the only known approach [24, 1] for solving the $\mathrm{DCP}_f(k, g)$ is to express the maps for multiplication by k and g using rational functions $u_k, u_g \in \mathbb{F}_p(x)$ and substitute into f:

$$f\left(u_k(x), u_g(x)\right) = 0. \tag{2}$$

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

Remark 2. All complexities are stated in an indeterminate $M(m)$ for the complexity of multiplying two degree-m polynomials. The complexity of root-finding of a degree-m polynomial is $O(M(m) \log m \log mp)$ operations in $\mathbb{F}_p$ [27].

Regarding the complexity of the just described method, the main bottleneck is the construction and root finding of $\overline{f}$, i.e., the numerator of (2). The rational function u_k can be computed using division polynomials in $O(M(k^2) \log k)$ time/space and satisfies $\deg(u_k) = k^2$ [28]. Similarly, $\deg(u_g) = g^2$ and so $\deg(\overline{f}) \leq 2 \max(k, g)^2 \deg f$. Taking the Karasuba complexity $m^{1.6}$ for $M(m)$, the overall time and space complexity is:

$$O(M(\delta^2) \log \delta \log \delta^2 p) \subseteq O(\delta^{3.2} \log \delta \log \delta^2 p), \quad \delta = 2 \max(k, g)^2 \deg f. \tag{3}$$

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

Instances of DCP that have been solved in the literature for large k focused on special types of polynomials f. The authors of [1, 22, 24] target $\mathrm{DCP}_f(k)$ in which the polynomial f depends on the coordinates of only one input point. For instance, if $f = x_1 + 1$, we can first find a point Q that satisfies $x_Q + 1 = 0$ and then compute $P = k^{-1}Q$. As our analysis in the following section shows, most addition formulas do not contain such polynomials, and the attacker is left with the general (hard) DCP problem. Hard DCP problems on the GLV curve, including the `secp256k1`, are the focus of this work.

3 Classification of formulas

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

3.1 Methodology

Our source of formulas was the updated export[5] of the EFD from [24]. For curves, we have used the database of standard prime field curves from [25], including 144 Weierstrass curves, 15 Edwards curves, 5 Montgomery curves, and 31 Twisted Edwards curves.

Example 1. The following is part of the three-operand code for the projective `add` formula (1): `t0 = Y1*Z2, t1 = Y2*Z1, u = t1-t0, t2 = X1*Z2, t3 = X2*Z1, v = t3-t2, t4 = u^2, t5 = v^3, t6 = v^2,..., A = t12-t9, X3 = v*A, Y3 = t20-t19, Z3 = t21*t22`

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

$$IV = f(X_1, Y_1, Z_1, X_2, Y_2, Z_2), \quad f \in \mathbb{Z}[X_1, Y_1, Z_1, X_2, Y_2, Z_2], \tag{4}$$

For instance, for the third IV in Example 1, we have `u=(Y2*Z1-Y1*Z2)`. For each formula F and a curve E, we consider the set of polynomials $\mathcal{I}_{F,E}$ containing the factors of the polynomials of every IV in F. Any ZVP attack on the formula must then use one of the polynomials in $\mathcal{I}_{F,E}$ to force a zero IV.

We label any polynomial $f \in \mathcal{I}_{F,E}$ as **easy** if it depends on the coordinates of only one point (either X_1, Y_1 or X_2, Y_2). We have seen in Section 2 that the DCP is easily solvable in such cases. Any other polynomial in $\mathcal{I}_{F,E}$ is labeled **hard**. Moreover, in the context of ZVP attacks, we exclude the following polynomials f from $\mathcal{I}_{F,E}$:

- easy polynomials f that do not have a solution corresponding to a point on the curve E, including constant polynomials that are trivially without roots.

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

- the few f for which $f = 0$ depends on the choice of the projective representation of the input point(s). This choice is based on the assumption that a projective randomization of points is in place. It also allows us to transform all polynomials in $\mathcal{I}_{F,E}$ to an affine form by putting $Z_i = 1$.
- f s.t. $f = 0$ implies that one of the coordinates of the input or output points (e.g., `X3` in Example 1) is 0. Recall that forcing zero coordinates of points is the domain of RPA and EPA attacks (Section 2). We exclude RPA attacks in our analysis of formulas since they do not depend on the choice of formula. We also exclude the general EPA attacks (with the exception of EPA on secp256k1 in Section 3.3) as their analysis has been done in [24].
- f s.t. $f = 0$ implies that the input or output points are of small order or satisfy some trivial condition, i.e., $(X_1, Y_1) = -(X_2, Y_2)$. Such points should not appear in the SM and can often be used for easier attacks than ZVP. Nevertheless, this assumption depends on the context, so we also recomputed the analysis without it.

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

3.2 Results

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

Table 1 shows the results for dbl formulas. For clarity, formulas are grouped by their coordinate systems with the number of formulas in brackets. The coordinate systems are abbreviated: `jac` for jacobian, `proj` for projective, `mod` for modified jacobian, `xz` for xz, `xyz` for xyzz. The formulas are divided into either resistant or vulnerable, as per our definition above. The rows further divide the formulas per the number of easy IV polynomials (e.g., resistant formulas have none). For example, for P-256, there are two vulnerable projective formulas with two easy IV polynomials and twelve resistant jacobian formulas. Clearly, the resistance of formulas depends on the choice of the curve. While most formulas are vulnerable for P-521, the opposite is true for P-256. Surprisingly, none of them are vulnerable for secp256k1. This means that the ZVP attack cannot target any dbl formula from EFD when the secp256k1 curve is used.

Similarly, Table 2 shows the results for add formulas. It turned out that there are no resistant Weierstrass add formulas, and so formulas are only divided into

	#IV	P-256	secp256k1	P-384	P-521

resistant	0	jac(12)
		mod(3),xyz(6)
		proj(9)
			jac(12)
			mod(3), xyz(2)
			proj(5), xz(5)
				jac(2)
				proj(2)
				xyz(3)
					jac(2)
					proj(2)
					xyz(3)

vulnerable	1	proj
		xz(5)
				jac(10)
				mod(3), xyz(3)
				proj(7), xz(3)
					jac(4), mod(3)
					proj(3)
					xyz(1), xz(3)

	2	proj(2)		proj(2)	jac(6), proj(4)
					xyz(2), xz(2)

	3			proj, xz(2)	
	5				proj
	6				proj(2)

Table 1: Distribution of IV polynomials for dbl formulas and four popular curves. Formulas are grouped by their coordinate system with their amount in brackets. Each formula is either resistant or vulnerable with the number of easy IV polynomials in rows.

semi-resistant or vulnerable. This means that each formula either contains an easy IV polynomial (vulnerable formula) or no easy ones and at least one hard IV polynomial (semi-resistant formula). The number of these polynomials is again in the rows. Since the semi-resistance of formulas did not change significantly between curves, we only show the results for P-256 and secp256k1. Focusing on secp256k1, we found the following eleven hard IV polynomials shared between the formulas, which we will use for our attacks (Section 4).

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

Out of the 36 semi-resistant add formulas: 32 formulas contained only one IV polynomial (f_1, f_2, f_4 or f_5), two formulas contained two IV polynomials (both f_1, f_3), one formula contained f_4, f_6, f_7 and one formula contained $f_1, f_8, f_9, f_{10}, f_{11}$. The overall conclusion for secp256k1 is that the ZVP attack on any classical scalar multiplication algorithm using dbl and add must focus on the add with all but two formulas being semi-resistant with hard DCPs (the exceptions are two variants of the formula by Renes, Costello, and Batina [23]). Furthermore, the amount of possible IVs is limited (only one for most formulas). We have to emphasize that we do not make any claims for formulas that are not in the EFD database.

Curve25519. To illustrate our results on other operations and curve models, we will summarize our findings for the popular Curve25519 in the Montgomery form. All four formulas for the differential addition are resistant (contain no

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

Table 2: Distribution of IV polynomials for add formulas and for P-256 and secp256k1. Formulas are grouped by their coordinate system with their amount in brackets. Each formula is either semi-resistant or vulnerable with the number of hard (for resistant) or easy (for vulnerable) IV polynomials in rows. There are no resistant Short Weierstrass add formulas in EFD.

IV polynomials usable for ZVP). Similar holds for ladder and doubling formulas with the exceptions of `xz:ladd-1987-m` and `xz:dbl-1987-m`. Both of these formulas contain an easy IV polynomial $X_1 - a$, where a is the Montgomery model parameter. Surprisingly, all of the considered Montgomery curves have a solution point for this polynomial (M-221, M-511, Curve383187 or M-383) except for Curve25519. This means that all of the formulas supporting Curve25519 are resistant against ZVP with this curve.

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

3.3 Resistance of secp256k1 to EPA and RPA

We excluded the EPA attack (a special case of ZPV) in our analysis of formulas as it was already done by Sedlacek et al. [24]. The EPA targets a polynomial f, for which $f = 0$ causes an error during the scalar multiplication. As shown in [24], all known add formulas in the EFD database for short Weierstrass curves offer only one f that could be zero: $y_1 + y_2$. The authors focused on the general case and did not consider how the situation changes for particular curves. We will now show that for the secp256k1 (and the general GLV curves) that $\mathrm{DCP}_{y_1+y_2}(k, g)$

has a solution only for three values of $\frac{k}{g}$. Since k corresponds to a prefix of a random secret scalar (target of the EPA attack), the probability of encountering such k and g is negligible, making the `secp256k1` curve resistant to EPA.

The GLV curves of the form $y^2 = x^3 + b$ are acted upon by an endomorphism $\lambda(x, y) = (\beta x, y)$, where $\beta \in \mathbb{F}_p$ is a (nontrivial) third root of 1. If P is a solution to $\mathrm{DCP}_{y_1+y_2}(k, g)$, then $Q = kP$, $G = gP$ and $y_Q + y_G = 0$. Squaring the equation and replacing y_Q^2 and y_G^2 using the curve equation, we get $x_Q^3 = x_G^3$, which means that either $x_Q = x_G$, $x_Q = \beta x_G$ or $x_Q = \beta^2 x_G$. These three cases imply, respectively, that either $Q = -G$, $Q = -\lambda G$, or $Q = -\lambda^2 G$. Consequently, $\frac{k}{g} \in \{-1, -\lambda, -\lambda^2\}$ as claimed.

Similarly for `ladd` formulas, the only potential zero polynomial is $f = x_2$ [24]. This means that as long as the curve does not contain points with zero x-coordinate, the curve is resistant to EPA attack. This is the case for `secp256k1` as the curve equation is $y^2 = x^3 + b$ with nonsquare b. For the same reason, `secp256k1` is resistant to RPA attack, for which a zero coordinate point is necessary. Thus, if the target is `secp256k1`, only the general ZVP attack is relevant and is also the focus of the next section.

4 ZVP-GLV attack

During the classical ZVP attack as described in Section 2, the attacker needs to find a solution to a $\mathrm{DCP}_f(k, g)$ where k is a small prefix of the private key d and g is the guess on the next digit. The situation changes when the implementation uses the GLV decomposition $d = d_0 + d_1\lambda \pmod{n}$. The multiscalar multiplication algorithms gradually go through the prefixes of both the scalars d_0 and d_1 at the same time. The attacker would now need to solve $\mathrm{DCP}_f(k_0 + k_1\lambda, g)$, where both k_0, k_1 are small but $k_0 + k_1\lambda$ is not. Since the state-of-the-art approach for solving DCP works only for small k, it seems that the ZVP attack cannot be used for implementations using the GLV decomposition. This section shows that the opposite is true:

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

Firstly, Section 4.1 describes a method for solving $\mathrm{DCP}_f(k_0 + k_1\lambda, g)$. In Section 4.2, 4.3 and 4.4, we use the method to attack the Straus-Shamir trick (Algorithm 3) and the interleaving algorithm (Algorithm 2). We also show that one variant of the interleaving algorithm is especially vulnerable to ZVP attack, allowing the attacker to recover the majority of the digits of d (Section 4.5). All of the sections contain experimental results on the `secp256k1` curve as well.

Attacker model. The target is a device that performs MSM $P \mapsto dP$ on a GLV curve $E/\mathbb{F}_p : y^2 = x^3 + b$ using a fixed secret scalar $d = d_0 + d_1\lambda \pmod{n}$. Here, n is the order of $E(\mathbb{F}_p)$, λ is the action of an endomorphism ϕ on $E(\mathbb{F}_p)$, where $\lambda^2+\lambda+1 = 0 \pmod{n}$. The MSM is done using either Algorithm 3 or Algorithm 2. The scalars d_0 and d_1 have l digits and are encoded in the appropriate representations (GLV-SAC or regular w-NAF). The attacker controls the input point P and can detect (using a suitable side channel) whether a zero appears during the computation of add, i.e., they have an oracle $\mathcal{O}$ described in Section 2.2 (we extend this model in Section 4.5). The attacker's goal is to recover as many digits of d as possible using $\mathcal{O}$. All of our experiments are simulated, i.e., we **do not** perform any side-channel measurements and treat $\mathcal{O}$ as a black-box oracle.

4.1 DCP on GLV curves

The state-of-the-art approach to solving $\mathrm{DCP}_f(k, g)$, as described in Section 2.3, is to find a root of a univariate polynomial $\overline{f}$ of degree $\approx 2\max(k, g)^2 \deg f$. As we show in Lemma 1, we can extend the approach to $\mathrm{DCP}_f(k_0 + k_1\lambda, g)$ where both k_0, k_1 are small. We construct a polynomial F of degree $\approx 16\max(k_0, k_1, g)^2 \deg f$ and then look for roots of F. The key idea behind the construction of F is that the multiplication-by-k map can be replaced by two multiplication-by-k_i maps while the degree of the map for λ is one, bringing no additional cost.

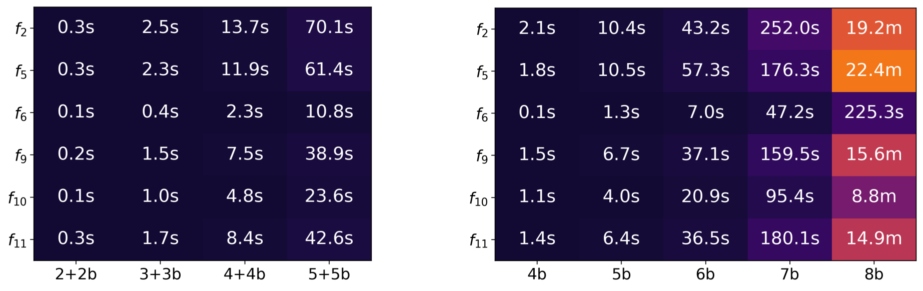
Lemma 1. *Let $E/\mathbb{F}_p$ be a GLV curve with an endomorphism ϕ and an action λ. For $k_0, k_1, g \in \mathbb{Z}$ and $f \in \mathbb{F}_p[x_1, x_2]$, consider the $DCP_f(k_0 + k_1\lambda, g)$. There is a polynomial $h \in \mathbb{F}_p[x]$ of degree $\leq \delta = 16\max(k_0, k_1, g)^2 \deg f$, such that if $P \in E$ is a solution to the DCP then x_P is a root of h. The polynomial h can be found with the following time and space complexity:*

$$O(M(\delta^2) \log\delta \log \delta^2 p).$$

Proof. Denote $u_0, u_1, u_g \in \mathbb{F}_p(x)$ the rational functions for the x-coordinates of the multiplication maps by k_0, $k_1\phi$ and g, respectively. To construct h, we will substitute u_0, u_1, g, f into the *Semaev* polynomial S_3 [26]. The polynomial $S_3 \in \mathbb{F}_p[X_1, X_2, X_3]$ satisfies for any points $P_0+P_1 = P_2$ that $S_3(x_{P_0}, x_{P_1}, x_{P_2}) = 0$. In particular, for any $P \in E$ and $Q = k_0P + k_1\lambda P$, we have $S_3(x_{k_0P}, x_{k_1\lambda P}, x_Q) = 0$. Since $u_0(x_P) = x_{k_0P}$, $u_1(x_P) = x_{k_1\lambda P}$, we can substitute u_0, u_1 into S_3: $S_3(u_0(x_P), u_1(x_P), x_Q) = 0$. If P is a solution to the DCP, then $f(x_Q, x_G) = 0$ where $G = gP$. We will now assume that the polynomial $f(x_1, x_2) = 0$ can be expressed as a $x_2 = f'(x_1)$ for a rational function $f' \in \mathbb{F}_p(x_1)$ and leave the proof of the general case for Appendix 6.1. Substituting f' and g into S_3, we get $S_3(u_0(x_P), u_1(x_P), f'(g(x_P))) = 0$. As a result, if h is the denominator of the rational function $S_3(u_0(x), u_1(x), f'(g(x))) \in \mathbb{F}_p(x)$ then x_P is its root as claimed. The degree of h is given by the degree of f' (which is $\deg f$), the maximum of degrees of u_0, u_1, u_g and the degree of S_3 (which is 2 in each variable). The complexity of constructing h is dominated by the construction of the maps u_0, u_1, u_g, and their substitution into S_3 with the complexity discussed in Section 2.3. □

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

Figure 2a shows the timings[6] of solving $\mathrm{DCP}_f(k_0 + k_1\lambda)$ for scalars k_0, k_1 of small bit-lengths (2-5 bits) and a selection of IV polynomials that we found in our analysis of formulas for `secp256k1` in Section 3. For comparison, Figure 2b shows the timings of solving $\mathrm{DCP}_f(k)$ classically for small k. While solving $\mathrm{DCP}_f(k)$ for 8-bit k can take more than 18 minutes (for f_2), solving $\mathrm{DCP}_f(k_0 + k_1\lambda)$ for two 4-bit scalars k_0, k_1 takes less than 14 seconds. In Section 4.2, we will show that this makes the decomposed private key $d = d_0 + d_1\lambda$ more vulnerable.



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

Fig. 2: Timings for solving $\mathrm{DCP}_f(k_0 + k_1\lambda)$ and $\mathrm{DCP}_f(k)$ for different polynomials f_i and scalar bit-lengths on the `secp256k1` curve.

Using Lemma 1, we can efficiently solve $\mathrm{DCP}_f(k_0 + k_1\lambda)$ for small k_0, k_1 *assuming* that there is at least one solution. To empirically estimate the probability that a DCP has a solution, we have solved $\mathrm{DCP}_f(k_0 + k_1\lambda)$ for all positive $2 - 5$ bit scalars k_0, k_1 and all the IV polynomials $\{f_1, \ldots, f_{11}\}$. For each such DCP instance, we have recorded whether it has at least one solution. Additionally, as we saw in Section 3, some formulas have up to five IV polynomials. Hence, we have combined the results for individual IV polynomials to see how the success rate of finding a solution changes if it is enough to have at least one solution for at least one polynomial from a given set of IV polynomials. Results are in Figure 3 with the number of IV polynomials on the y-axis and the success rate (in percentages) on the x-axis. Each dot represents a set of IV polynomials of size $1, 2, 3, 4$ or 5. Surprisingly, the DCP success rate for single polynomials is very low (below 25% for all except for f_6, f_7, f_{11}), which limits the ZVP attack

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

on formulas with single IVs. With at least three IV polynomials, the average rate is over 50%, and with five IV polynomials, the rate reaches 87%.

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

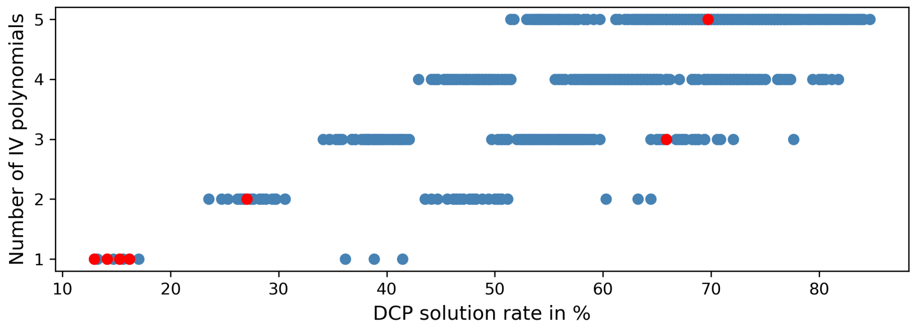


Fig. 3: Each dot represents a subset of IV polynomials from $\{f_1, \ldots, f_{11}\}$ with their amount on y-axis and the rate of at least one solvable $\mathrm{DCP}_{f_i}(k_0 + k_1\lambda)$ problem on the x-axis. The red dots represent EFD `add` formulas.

4.2 Description of the ZVP-GLV attack

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

The general idea of the ZVP-GLV attack follows the idea of the classical ZVP attack: we iteratively recover the upper digits of d_0, d_1, where $d = d_0 + d_1\lambda$ is the decomposition of the private key. In each iteration, we take a guess of the digits $d_0^{(i)}$, $d_1^{(i)}$. To verify our guess, we attempt to invoke a zero during the computation of `add` with a special input point found as a solution to DCP using Lemma 1. If our guess is correct, then we detect a zero. It might happen that the DCP does not have a solution. As we saw in Figure 3, the probability of that event is relatively high (as high as 88%) unless multiple intermediate polynomials are used. This means that the individual guesses for $d_0^{(i)}, d_1^{(i)}$ might not be verified and we must keep track of all possibilities for the values of $d_0^{(i)}, d_1^{(i)}$.

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

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

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

- For each candidate pair $(\kappa_0, \kappa_1) \in K_i$, we make a guess g_0, g_1 for the following digits $d_0^{(i-1)}, d_1^{(i-1)}$. The set of options for g_0, g_1 depends on the representation (GLV-SAC for Algorithm 3 or regular w-NAF for Algorithm 2).
- For each candidate pair κ_0, κ_1 and a guess g_0, g_1 we create an instance of the DCP problem and attempt to find a solution point P using Lemma 1. If we have multiple intermediate polynomials available, we create and solve a DCP for each one of them until a solution is found or we have no more polynomials. Note that some candidate pairs and guesses might have a common solution point (see Remark 1).
- Using all found points P, we query the zero detection oracle $\mathcal{O}(P)$. We reject all candidate pairs and guesses for which a zero was not detected, as they have to be incorrect.
- Any of the candidate pairs κ_0, κ_1 and guesses g_0, g_1, for which a zero was detected on the corresponding point P, might be correct as some of them might share the point P. We put $(\kappa_0 | g_0, \kappa_1 | g_1) \in K_i$ for all of these (where $|$ stands for concatenation).
- If no zeros are detected (and so far $K_i = \{\}$) for any point P, then the correct candidate pair κ_0, κ_1 and guess g_0, g_1 must be one of the ones for which the DCP had no solution point, and so we put all of these in K_i. In principle, we should keep the guesses without solutions even if a zero was detected, as we might lose the correct guess g_0, g_1 due to accidental zeros. We never found this to be the case in our experiments and so we use this heuristic for more efficiency.

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

The following subsections describe the details specific to Algorithm 3 and Algorithm 2 and the experimental results. The pseudocodes for the attacks can be found in Appendix 6.2 (Algorithm 4, Algorithm 5 and Algorithm 6).

4.3 ZVP-GLV attack on Straus-Shamir trick

The ZVP-GLV attack on the Straus-Shamir trick (Algorithm 3) iteratively uncovers the GLV-SAC representation of the upper bits. This means that (g_0, g_1)

$\in \{(1,0),(-1,0),(1,1),(-1,-1)\}$, using the general notation from above. We will now explain how to construct the DCP problem for GLV-SAC. The rest of the attack follows the general description above.

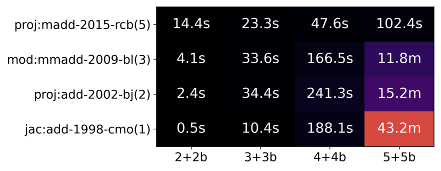
A guess (g_0, g_1) for $d_0^{(i)}, d_1^{(i)}$ corresponds to a guess of the input point $(g_0 + g_1\lambda)P \in \{P, -P, P+\lambda P, -P-\lambda P\}$. Given a prefix candidate (κ_0, κ_1), we verify the guess by solving $\mathrm{DCP}_f(\kappa_0+\kappa_1\lambda, g_0+g_1\lambda)$, i.e., find P such that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

The experimental results for this attack are in Figure 4a and Figure 4b. For comparison, we also simulated the classical ZVP attack on the regular 2-NAF double-and-add with the results in Figure 4c and Figure 4b. We ran the experiments for four addition EFD formulas with a different number of IV polynomials. The target bits were 4, 6, 8 and 10 for ZVP-GLV and only 4,5,6,7 for the classical ZVP, as more than 7 bits was too time-consuming. Note that these formulas represent the four red clusters in Figure 3 with their DCP solution rate 70%, 66%, 27%, and 16%, respectively. The duration and the recovery rate of both the classical ZVP and our ZVP-GLV attack improve with more IV polynomials. A lower number of IV polynomials causes a lower DCP solution rate, which increases the set of possible candidates K_i and the number of DCP problems in each iteration.

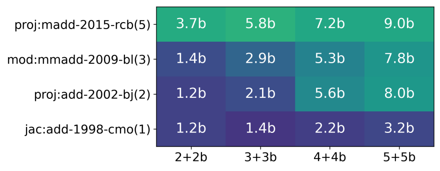
Overall, our ZVP-GLV attack was able to recover significantly more bits in less time than the classical ZVP attack. For instance, while the classical ZVP attack took 28 minutes to recover 6.3 bits (with `proj:madd-2015-rcb`), our ZVP-GLV attack recovered 7.2 bits in 48 seconds. For `proj:add-2009-bl`, the ZVP attack took 64 minutes to recover 4.9 bits, and the ZVP-GLV attack managed to recover 5.6 bits in 240 seconds. We also targeted higher bit-lengths for `proj:madd-2015-rcb` to see the advantage of ZVP-GLV. We were able to recover 16 bits with the ZVP-GLV attack in 230 minutes – the same time it took the classical ZVP attack to recover 7 bits.

4.4 ZVP-GLV attack on window interleaving

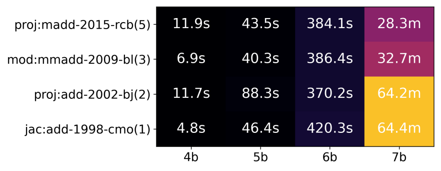
The interleaving algorithm (Algorithm 2) performs two `add` operations in each iteration, one for each scalar d_0, d_1. Hence, our ZVP-GLV attack targets both additions separately in a similar spirit. We first make a guess g_0 for $d_0^{(i)}$, verify it and then repeat this for g_1 and $d_1^{(i)}$. To verify a prefix candidate $(\kappa_0, \kappa_1) \in K_i$ and a guess g_0 for $d_0^{(i)}$, we need to find point P such that $f(x_G, x_Q) = 0$ where $G = g_0 P$ and $Q = (2\kappa_0 + 2\kappa_1\lambda)P$. This means solving $\mathrm{DCP}_f(\kappa_0 + \kappa_1\lambda, g_0)$. For the guess g_1, the DCP instance is $\mathrm{DCP}_f(\kappa_0 + \kappa_1\lambda, g_1\lambda)$. The ZVP-GLV attack, in this case, uncovers the regular w-NAF representation, which means that it



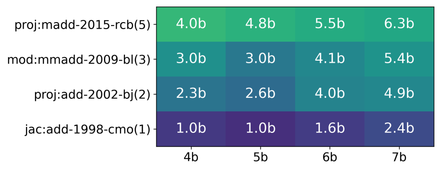
(a) ZVP-GLV timings



(b) ZVP-GLV recovery



(c) ZVP timings



(d) ZVP recovery

Fig. 4: Experimental results for the ZVP-GLV attack on the Straus-Shamir trick and the classical ZVP attack on the double-and-add (using `secp256k1`). Figure 4a and Figure 4c show the timings for four formulas (number of IV polynomials in brackets) and four target bits. Figure 4b and Figure 4d shows the average number of recovered bits.

can only target multiples of the window size. Figure 6 in Appendix 6.2 shows the experimental results of the attack on the first window for $w = 3, 4$ and 5. The recovery rate is lower than that of the Straus-Shamir trick. Still, the attack recovered 5.7 bits (for `mod:mmadd-2009-bl`) in 12 seconds, while it took the ZVP attack 32 minutes to recover 5.4 bits.

4.5 ZVP-GLV attack on window interleaving (alternative)

This section describes a ZVP-GLV attack on the alternative version of the interleaving Algorithm 2 in which $d_0^{(i)}P + d_1^{(i)}\lambda P$ is computed in each iteration.

For any IV polynomial f, the DCP instance corresponding to the addition $d_0^{(i)}P + d_1^{(i)}\lambda P$ is $\mathrm{DCP}_f(d_0^{(i)}, d_1^{(i)}\lambda)$. Both $d_0^{(i)}, d_1^{(i)}$ are small integers from $\{\pm 1, \pm 3, \dots, \pm(2^w - 1)\}$. On the one hand, this means that no large scalars are involved, and we have an easy instance of DCP. On the other hand, the DCP no longer depends on the accumulated scalar multiple Q, and so if a zero is detected using the simple binary oracle, we have no way of knowing in which iteration. Additionally, the zero might appear in multiple iterations. We will extend our side-channel model and assume that the position of the zero is visible in the traces. This is a reasonable assumption since the number of dbl and add operations is constant. More formally, we will consider a side-channel oracle $\mathcal{O}^*(P) \in \{0, 1\}^l$, which outputs a vector of 1s and 0s for each iteration, indicating whether a zero appeared in the iteration or not.

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

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

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

Figure 5 shows the results of the attack on the `secp256k1` curve with window size $w = 4$ (results for $w = 3, 5$ are in Appendix 6.2). We ran the attack with 100 scalars for four selected addition formulas (the number of IV polynomials is in the parentheses) and one fictitious formula containing IV polynomials with a high DCP success rate. The resulting figure is a histogram of the size of the space (in bits) given by the non-recovered $d_0^{(i)}, d_1^{(i)}$. The average bit-sizes are 98, 94, 162, 210 and 51 for `madd-2015-rcb`, `mmadd-2009-bl`, `add-2002-bj`, `add-1998-cmo`, $\{f_1, f_{11}, f_6, f_7, f_9\}$, respectively. In particular, for `mmadd-2009-bl` with the 94-bit space, this means a reduction of the original 256-bit space by 63% with 47-bit complexity for the baby-step giant-step to find the remaining bits $d_0^{(i)}, d_1^{(i)}$. For 3% of the private keys, the bit-size of the space was below 70 bits, which might be relevant for attacks on multiple private keys. The space for the fictitious formula has 51 bits. This shows that for any formula with IV polynomials with a high DCP success rate, the private key can be fully recovered. While we have not found any such formula in EFD supporting `secp256k1`, it is an important takeaway for standards and developers proposing new formulas to avoid complex formulas with a large number of IV polynomials.

Finding the private key. We will now sketch how to find the private key d in the reduced space. After the ZVP-GLV attack on the interleaving algorithm, we know certain digits of the decomposed parts d_0, d_1. We can assume that we know P, Q such that $dP = d_0P + d_1\lambda P = Q$. We can now cut d_0, d_1 in halves into upper and lower digits, i.e. $d_0 = 2^{e_0}u_0 + l_0$, $d_1 = 2^{e_1}u_1 + l_1$. We iterate through the unknown parts of u_0, u_1 and store the possible values for $2^{e_0}u_0P + 2^{e_1}u_1\lambda P$. Then, we iterate through the unknown parts of l_0, l_1, computing $Q - l_0P - l_1\lambda P$ and comparing it with the stored candidates. While this baby-step giant-step approach gives us 50-bit time complexity to recover the unknown 100 bits in Figure 5, it requires to store 2^{50} points. Solving this type of problem in constant space is an open problem, e.g., using a variant of Pollard's rho algorithm [11].

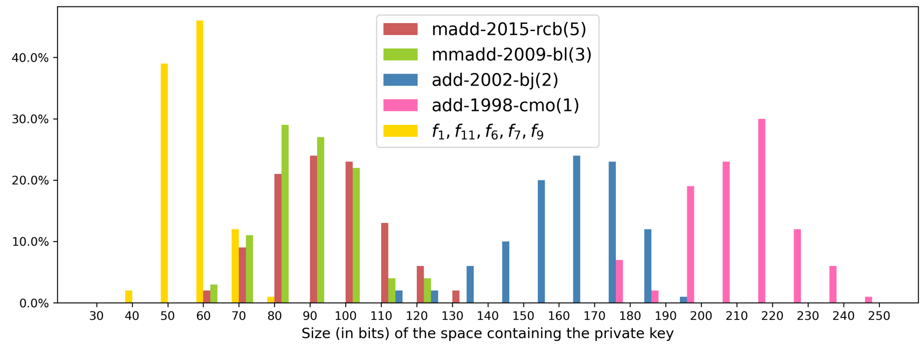


Fig. 5: Histogram of the number of unknown bits after the ZVP-GLV attack on the regular 4-NAF interleaving algorithm. The attack was performed on 100 scalars for four different addition formulas and one fictitious formula containing IV polynomials with a high success rate.

5 Conclusion

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

Our ZVP classification of formula-curve combinations (Section 3) gave us several important takeaways for standards and developers proposing new formulas for SCA protected implementations. Less complicated formulas with fewer IV polynomials are less vulnerable to ZVP attacks. Our ZVP-GLV attacks then demonstrate that the choice of formula can mean the difference between a full key recovery and a complete ZVP resistance.

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

Acknowledgements

We thank Sorina Ionica for the initial idea of exploiting small endomorphisms. This project has been made possible in part by a grant from the Cisco University

Research Program Fund, an advised fund of Silicon Valley Community Foundation. V. Suchanek was also supported by the Ph.D. Talent Scholarship – funded by the Brno City Municipality. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development, and Innovations Infrastructures.

References

[1] T. Akishita and T. Takagi. "Zero-Value Point Attacks on Elliptic Curve Cryptosystem". In: *Information Security*. Ed. by C. Boyd and W. Mao. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 218–233. ISBN: 978-3-540-39981-0.

[2] R. Bellman and E. Straus. "Addition chains of vectors (problem 5125)". In: *The American Mathematical Monthly* 71.7 (1964), pp. 806–808.

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

[4] D. J. Bernstein and T. Lange. *Explicit-formulas database*. 2007. URL: http://hyperelliptic.org/EFD.

[5] D. J. Bernstein and T. Lange. *SafeCurves: choosing safe curves for elliptic-curve cryptography*. 2017. URL: https://safecurves.cr.yp.to/.

[6] J. W. Bos, C. Costello, H. Hisil, and K. Lauter. "High-performance scalar multiplication using 8-dimensional GLV/GLS decomposition". In: *Cryptographic Hardware and Embedded Systems-CHES 2013: 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings 15*. Springer. 2013, pp. 331–348.

[7] S. Bowe. *BLS12-381: New zk-SNARK Elliptic Curve Construction*. https://electriccoin.co/blog/new-snark-curve/. 2017. (Visited on 01/16/2024).

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

[9] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0*. 2010. URL: https://secg.org/.

[10] H. Cohen, A. Miyaji, and T. Ono. "Efficient elliptic curve exponentiation using mixed coordinates". In: *Advances in Cryptology—ASIACRYPT'98: International Conference on the Theory and Application of Cryptology and Information Security Beijing, China, October 18–22, 1998 Proceedings*. Springer. 1998, pp. 51–65.

[11] G. De Micheli and N. Heninger. "Recovering cryptographic keys from partial information, by example". In: *Cryptology ePrint Archive* (2020).

[12] A. Faz-Hernández, P. Longa, and A. H. Sánchez. "Efficient and Secure Algorithms for GLV-Based Scalar Multiplication and Their Implementation on GLV-GLS Curves". In: *Topics in Cryptology – CT-RSA 2014*. Ed. by J.

Benaloh. Cham: Springer International Publishing, 2014, pp. 1–27. ISBN: 978-3-319-04852-9.

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

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

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

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

[17] T. Izu and T. Takagi. "Exceptional Procedure Attack on Elliptic Curve Cryptosystems". In: *Public Key Cryptography — PKC 2003*. Ed. by Y. G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 224–239. ISBN: 978-3-540-36288-3.

[18] M. Joye and M. Tunstall. "Exponent Recoding and Regular Exponentiation Algorithms". In: *Progress in Cryptology – AFRICACRYPT 2009*. Ed. by B. Preneel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 334–349. ISBN: 978-3-642-02384-2.

[19] P. Kocher, J. Jaffe, and B. Jun. "Differential power analysis". In: *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer. 1999, pp. 388–397.

[20] P. Longa and F. Sica. "Four-dimensional Gallant-Lambert-Vanstone scalar multiplication". In: *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*. Springer. 2012, pp. 718–739.

[21] P. L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of computation* 48.177 (1987), pp. 243–264.

[22] C. Murdica, S. Guilley, J.-L. Danger, P. Hoogvorst, and D. Naccache. "Same Values Power Analysis Using Special Points on Elliptic Curves". In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by W. Schindler and S. A. Huss. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 183–198. ISBN: 978-3-642-29912-4.

[23] J. Renes, C. Costello, and L. Batina. "Complete Addition Formulas for Prime Order Elliptic Curves". In: *Proceedings, Part I, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT*

2016 - Volume 9665. Berlin, Heidelberg: Springer-Verlag, 2016, 403–428. ISBN: 9783662498897.

[24] V. Sedlacek, J.-J. Chi-Domínguez, J. Jancar, and B. B. Brumley. “A formula for disaster: a unified approach to elliptic curve special-point-based attacks”. In: *Advances in Cryptology – ASIACRYPT 2021*. Springer, 2021. ISBN: 978-3-030-64837-4.

[25] V. Sedláček, V. Suchánek, and A. Dufka. *DiSSECT:Distinguisher of Standard and Simulated Elliptic Curves via Traits*. https://dissect.crocs.fi.muni.cz/.

[26] I. Semaev. *Summation polynomials and the discrete logarithm problem on elliptic curves*. Cryptology ePrint Archive, Paper 2004/031. https://eprint.iacr.org/2004/031. 2004. URL: https://eprint.iacr.org/2004/031.

[27] J. Von Zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[28] L. C. Washington. *Elliptic curves: number theory and cryptography*. CRC press, 2008.

[29] P. Wuille. *libsecp256k1*. https://github.com/bitcoin-core/secp256k1. 2013.

6 Appendix

6.1 Full proof of Lemma 1

Denote $u_0, u_1, u_g \in \mathbb{F}_p(x)$ the rational functions for x-coordinates of multiplication maps by k_0, $k_1\phi$ and g, respectively. Let $S_3 \in \mathbb{F}_p[X_1, X_2, X_3]$ be the Semaev polynomial. For a solution P of the DCP, we have

$$S_3(x_Q, x_{k_0P}, x_{k_1\lambda P}) = S_3(x_Q, u_0(x_P), u_1(x_P)) = 0,$$

where $Q = k_0P + k_1\lambda P$ and $f(x_G, x_Q) = 0$, $G = gP$. We can substitute u_g into f and get $f(u_g(x_P), x_Q) = 0$. Consider the rational function $S_3(x_2, u_0(x), u_1(x))$ as a quadratic polynomial $S_3'(x_2)$ in x_2 with coefficients in $\mathbb{F}_p(x)$, i.e. $S_3' \in \mathbb{F}_p(x)[x_2]$. Denote r_1, r_2 the roots of the S_3'. Based on Vieta's formulas, any polynomial expression that is symmetric in r_1, r_2 can be expressed using the coefficients of S_3' – rational functions in x. In particular:

$$h = f(u_g(x), r_1)f(u_g(x), r_2) \in \mathbb{F}_p(x).$$

Since $f(u_g(x_P), x_Q) = 0$ then $F(x_P) = 0$. Note that if h vanishes then either $f(x_1, r_1) = 0$ or $f(x_1, r_2) = 0$, in which case any point P is a solution to the DCP. The degree of h is $2(g^2 + \deg r_1)\deg f = 2(g^2 + \deg(2k_0^2 + 2k_1^2))\deg f$ since the Semaev polynomial S_3 has degree 2 in each variable. This can be bounded by $\delta = 16m^2 \deg f$ where $m = \max(k_0, k_1, g)$. The complexity of constructing h is dominated by the construction of the maps u_0, u_1, u_g, and their substitution into S_3 with the complexity discussed in Section 2.3.

6.2 Algorithms and graphs referenced in the text

Algorithm 4: ZVP-GLV attack on the Straus-Shamir trick.

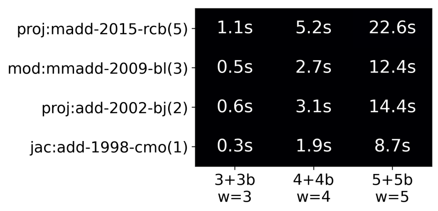
```
   input  : A set $K_{i+1}$ of candidates pairs for $d_0^{\geq i+1}, d_1^{\geq i+1}$
   output: A set $K_i$ of candidate pairs for $d_0^{\geq i}, d_1^{\geq i}$
1  $K_i, U_i \leftarrow \{\}, \{\}$
2  foreach $(\kappa_0, \kappa_1) \in K_{i+1}$ do
3      for $(g_0, g_1) \in \{(1,0), (-1,0), (1,1), (-1,-1)\}$ do
4          $D \leftarrow \mathrm{DCP}_f(g_0\kappa_0 + \lambda(g_0\kappa_1 - g_1\kappa_0))$
5          Find a solution point $P$ to $D$ using Lemma 1
6          if no solution then add $(2\kappa_0 + g_0, 2\kappa_1 + g_1)$ to $U_i$
7          else if $\mathcal{O}(P) = 1$ then add $(2\kappa_0 + g_0, 2\kappa_1 + g_1)$ to $K_i$
8  if $K_i$ is empty then $K_i \leftarrow U_i$
9  return $K_i$
```

Algorithm 5: ZVP-GLV attack on the regular w-NAF interleaving.

```
   input  : A set $K_{i+1}$ of candidates pairs for $d_0^{\geq i+1}, d_1^{\geq i+1}$
   output: A set $K_i$ of candidate pairs for $d_0^{\geq i}, d_1^{\geq i}$
1  $K_i, U_i \leftarrow \{\}, \{\}$
2  foreach $(\kappa_0, \kappa_1) \in K_{i+1}$ do
3      for $g_0 \in \{\pm 1, \dots, \pm(2^w - 1)\}$ do
4          $D \leftarrow \mathrm{DCP}_f(2^w\kappa_0 + 2^w\lambda\kappa_1, g_0)$
5          Find a solution point $P$ to $D$ using Lemma 1
6          if no solution then add $(2^w\kappa_0 + g_0, 2^w\kappa_1)$ to $U_i$
7          else if $\mathcal{O}(P) = 1$ then add $(2^w\kappa_0 + g_0, 2^w\kappa_1)$ to $K_i$
8  if $K_i$ is empty then $K_i \leftarrow U_i$
9  return $K_i$
```

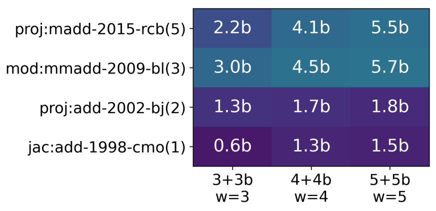
Algorithm 6: ZVP-GLV attack on regular w-NAF interleaving.

```
    input  : Oracle $\mathcal{O}^*$, integer $t$, polynomial $f$
    output: A set $C_i$ of candidates pairs of $w$-NAF bits $d_0^{(i)}, d_1^{(i)}$ for each $i$
1   $\mathcal{G} \leftarrow \{\pm 1, \dots, \pm(2^w - 1)\}^2$
2   $\mathcal{P} \leftarrow \{\}$
3   for $(g_0, g_1) \in \mathcal{G}$ do
4       Attempt to find a solution point $P$ to $\mathrm{DCP}_f(g_0, g_1)$ and store in $\mathcal{P}$
5   for $P \in \mathcal{P}$ do
6       Store in $\mathcal{T}_P$ all $(g_0, g_1) \in \mathcal{G}$ for which $P$ is a solution to $\mathrm{DCP}_f(g_0, g_1)$
7   for $i = 0$ to $l - 1$ do
8       $C_i \leftarrow \mathcal{G}$
9       for $P \in \mathcal{P}$ do
10          if $\mathcal{O}^*(P)_i = 1$ then $C_i \leftarrow C_i \cap \mathcal{T}_P$
11          else $C_i \leftarrow C_i \setminus \mathcal{T}_P$
12  return $C_0, \dots, C_{l-1}$
```



	3+3b w=3	4+4b w=4	5+5b w=5
proj:madd-2015-rcb(5)	1.1s	5.2s	22.6s
mod:mmadd-2009-bl(3)	0.5s	2.7s	12.4s
proj:add-2002-bj(2)	0.6s	3.1s	14.4s
jac:add-1998-cmo(1)	0.3s	1.9s	8.7s

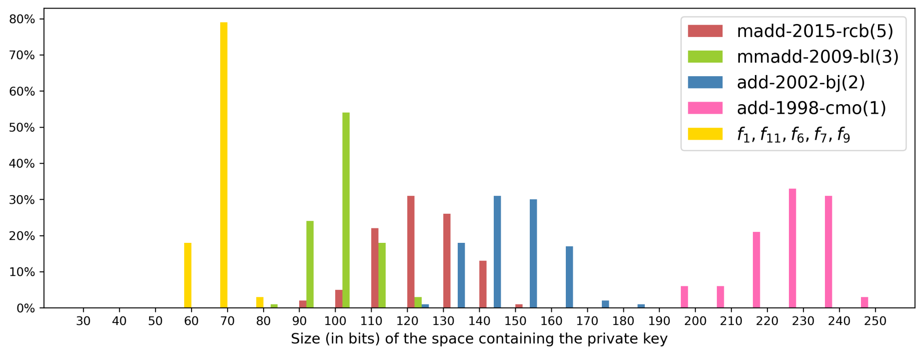
(a) ZVP-GLV timings



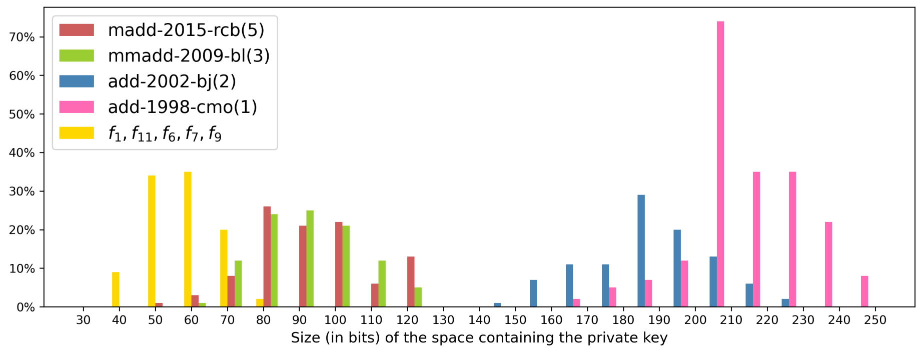
	3+3b w=3	4+4b w=4	5+5b w=5
proj:madd-2015-rcb(5)	2.2b	4.1b	5.5b
mod:mmadd-2009-bl(3)	3.0b	4.5b	5.7b
proj:add-2002-bj(2)	1.3b	1.7b	1.8b
jac:add-1998-cmo(1)	0.6b	1.3b	1.5b

(b) ZVP-GLV recovery

Fig. 6: Experimental results for the ZVP-GLV attack on the interleaving algorithm. Figure 6a shows the timings for four formulas and three target bits. Figure 6b shows the average number of recovered bits.



(a) $w = 3$



(b) $w = 5$

Fig. 7: Histogram of the number of unknown bits after the ZVP-GLV attack on the regular 3-NAF and 5-NAF interleaving algorithm. The attack was performed on 100 scalars for four different addition formulas and one fictitious formula containing IV polynomials with a high success rate.