IACR Transactions
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–26. DOI:XXXXXXXX

From A to Z: Projective coordinates leakage in the wild

Alejandro Cabrera Aldaya, Cesar Pereida García and Billy Bob Brumley

Tampere University, Tampere, Finland
{alejandro.cabreraaldaya,cesar.pereidagarcia,billy.brumley}@tuni.fi

Abstract. At EUROCRYPT 2004, Naccache et al. showed that the projective coordinates representation of the resulting point of an elliptic curve scalar multiplication potentially allows to recover some bits of the scalar. However, this attack has received little attention by the scientific community, and the status of deployed mitigations to prevent it in widely adopted cryptography libraries is unknown. In this paper, we aim to fill this gap, by analyzing several cryptography libraries in this context. To demonstrate the applicability of the attack, we use a side-channel attack to exploit this vulnerability within `libgcrypt` in the context of ECDSA. To the best of our knowledge, this is the first practical attack instance. It targets the insecure binary extended Euclidean algorithm implementation using a microarchitectural side-channel attack that allows recovering the projective representation of the output point of scalar multiplication during ECDSA signature generation. We captured 100k traces to estimate the number of traces an attacker would need to compromise the `libgcrypt` ECDSA implementation, resulting in less than 2k for commonly used elliptic curve `secp256r1`, demonstrating the attack feasibility. During exploitation, we found two additional vulnerabilities. However, we remark the purpose of this paper is not merely exploiting a library but about providing an analysis on the projective coordinates vulnerability status in widely deployed open-source libraries, filling a gap between its original description in the academic literature and the adoption of countermeasures to thwart it in real-world applications.

Keywords: applied cryptography · projective coordinates leak · open source libraries · side-channel analysis · ECDSA · modular inversion · binary GCD · libgcrypt · CVE-2020-10932 · CVE-2020-11735

1 Introduction

Side-channel attacks (SCA) are a major concern in the context of secure cryptography implementations. They aim at recovering secret key material using state leaks that can be used to compromise a cryptosystem. Since the pioneering works by Kocher [Koc96, KJJ99], every cryptographic algorithm implementation has been an attacker target.

The leakage sources available to the attackers depend on the computing platform where the target algorithm is executed. While power consumption and electromagnetic emanations are commonly exploited channels for embedded targets [KJJ99, Cor99, BCO04], shared microarchitecture components play this role for software implementations running on commodity desktop to high-end CPUs [Per05, AGS07, Ald+19b]. In the microarchitecture domain, several components can be used as leakage sources, such as cache-timings [YF14], cache-access patterns [OST06], branch predictors [AGS07], etc. Each microarchitecture attack vector exploits a shared resource available in microprocessors.

The task of assessing if a given algorithm implementation is SCA vulnerable often involves identifying which secret data the algorithm processes, usually found in the

Received: 20XX-XX-XX, Accepted: 20XX-XX-XX, Published: 20XX-XX-XX

algorithm specification. For instance, the elliptic curve digital signature algorithm (ECDSA) [Fip] during signature generation uses a private key and a secret nonce. Therefore, the implementer (or adversary) knows which primitives must be protected (or targeted).

ECDSA uses the secret nonce to compute a scalar multiplication, a highly targeted operation in the SCA realm [BH09, Fan+10, FV12, Dan+13, AVL19]. The modular inversion of the nonce is also a target [AGS07, PGB17, Ald+19a], as well as ECDSA private key operations [Rya19].

However, while the private key and the secret nonce are well-known values that must be secured, other secret-related values are less obvious. In this regard, in 2004 Naccache, Smart, and Stern [NSS04] demonstrated how the projective representation resulting from the computation of a scalar multiplication can be related to some bits of the scalar. While this attack does not have a direct application to ECDSA, as the projective representation of the scalar multiplication is not made public at the protocol specification level, it could apply if the adversary can recover this projective representation using SCA, for example.

Despite this clever *projective coordinates attack* being known for more than 15 years, it has received marginal attention by the scientific community. Maimut et al. [Mai+13] is the only related work we are aware of, where the authors proposed a set of fault attack models that could be used to recover the projective coordinate Z, allowing them to perform the attack. However, neither this nor the original paper targets a real implementation. This scenario allows us to formulate the following research question: *What is the status of widely deployed open-source software libraries regarding this attack?*

In this paper, we aim to answer this question by filling the gap between the academic results in [NSS04, Mai+13] and SCA mitigations in widely-deployed cryptography libraries. To complement this analysis, we practically demonstrate this attack feasibility by developing an end-to-end attack against the `libgcrypt` cryptography library. The main contributions in this work are: (1) Analysis of the projective coordinates attack status in open-source libraries; (2) first SCA on projective to affine coordinates conversion; (3) first exploitation of the projective coordinates leak in a real-world ECDSA implementation (`libgcrypt`); (4) demonstration that the Montgomery ladder is not a natural protection against this attack; (5) demonstration that randomizing the generator point sometimes does not prevent the attack; (6) first SCA on `libgcrypt` modular inversion; (7) three new vulnerabilities in `libgcrypt`: a projective coordinates attack, single-trace ECDSA nonce recovery, and recovery of a countermeasure mask.

The paper is organized as follows. Section 2 presents background on elliptic curve based cryptography and ECDSA signature generation. Section 3 describes the projective coordinates attack as presented by Naccache, Smart, and Stern [NSS04]. Section 4 analyzes some mitigation strategies against this attack, highlighting straightforward methods to thwart it. Later, Section 5 analyzes several widely-deployed open-source libraries regarding this attack, providing insight into the resistance level of these implementations to this threat. Before concluding in Section 7, Section 6 demonstrates an end-to-end attack against `libgcrypt` using microarchitectural SCA. The goal is to show the feasibility of the projective coordinates attack by running an in-depth experiment against a real-world application, leading to ECDSA private key recovery using fewer than 2000 traces. At the same time, we find and discuss two other vulnerabilities in the ECDSA path.

2 Background

2.1 Elliptic Curve Cryptography

This section provides preliminary background on elliptic curve cryptography (ECC) in the context of the *projective coordinates attack* [NSS04]. Following the work's rationale, the attack description uses a simplified Weierstrass equation for elliptic curves defined over a

finite prime field $\mathbb{F}_p$ of large characteristic. However, other elliptic curve representations can be analyzed following the same idea with corresponding changes.

An elliptic curve E is formed by the points $(x, y) \in \mathbb{F}_p^2$ that satisfy the equation $E : y^2 = x^3 + ax + b$, in addition to the point at infinity represented by $\mathcal{O}$. The set of points of $E(\mathbb{F}_p)$ and the definition of an addition operation form an additive abelian group with $\mathcal{O}$ as the identity element. Group law algebraic formulae of $E(\mathbb{F}_p)$ for point addition and doubling are defined as follows:
Point addition: Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$ and $Q = (x_2, y_2) \in E(\mathbb{F}_p)$ such that $P \neq \pm Q$. Then $P + Q = (x_3, y_3) \in E(\mathbb{F}_p)$, where:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \qquad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$$

Point doubling: Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$ such that $P \neq -P$. Then $2P = (x_3, y_3) \in E(\mathbb{F}_p)$, where:

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \qquad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

The above equations involve the computation of modular inverses in $\mathbb{F}_p$, often undesirable regarding performance due to its long computing time. To address this issue, developers often rely on a projective coordinate system to eliminate this *expensive* operation performed during addition and doubling formulae. A frequently used coordinate system is the Jacobian projective coordinate system, confirmed by our analysis on several popular cryptography libraries, presented in Section 5. We use this coordinate system to describe the attack, but likewise to the elliptic curve representation, the attack is adaptable to other coordinate systems [NSS04, Mai+13].

An affine curve point $(x, y) \in \mathbb{F}_p^2$ can be represented as a Jacobian point (Z^2x, Z^3y, Z) for any $Z \in \mathbb{F}_p^*$. Thus, given a Jacobian projective point (X, Y, Z), the corresponding affine point can be computed as $(\frac{X}{Z^2}, \frac{Y}{Z^3})$. Note that this conversion from Jacobian to affine representation involves the computation of the modular inverse of Z. Equations (1) and (2) (resp.) show point addition $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, Z_2)$ and point doubling $(X_3, Y_3, Z_3) = 2(X_1, Y_1, Z_1)$ formulae in Jacobian coordinates [HMV04].

$$\texttt{JADD} = \begin{cases} A = X_1 Z_2^2 \\ B = Y_1 Z_2^3 \\ C = X_2 Z_1^2 - A \\ D = Y_2 Z_1^3 - B \\ X_3 = -C^3 - 2AC^2 + D^2 \\ Y_3 = -BC^3 + D(AC^2 - X_3) \\ Z_3 = Z_1 Z_2 C \end{cases} \tag{1}$$

$$\texttt{JDBL} = \begin{cases} A = Y_1^2 \\ B = 4A \cdot X_1 \\ D = 3X_1^2 + a \cdot Z_1^4 \\ X_3 = D^2 - 2B \\ Y_3 = D \cdot (B - X_3) - 8A^2 \\ Z_3 = 2Y_1 \cdot Z_1 \end{cases} \tag{2}$$

The security of non-post-quantum ECC schemes is based on the hardness of computing discrete logarithms on elliptic curve groups (ECDLP). Roughly speaking, the ECDLP states that given the points P and G, recovering the integer k that satisfies $P = kG$ is

an intractable problem. The operation kG is known as *scalar multiplication* and consists of adding the point G with itself k times. This is the most computationally expensive operation in ECC, a popular research topic seeking to reduce its execution time [Tav+11, GK15].

In this regard, the scientific community has developed several proposals to implement scalar multiplication, taking performance and SCA security as optimization metrics. The projective coordinates attack [NSS04] adapts to different scalar multiplication algorithms [Mai+13], but for the sake of simplicity, and following previous work approaches [NSS04, Mai+13], we describe it using the *Double-and-Add* algorithm (Algorithm 1).

Algorithm 1 often uses R represented in Jacobian coordinates, implying that the resulting point R should be converted back to affine coordinates. As mentioned before, this conversion requires the computation of a modular inverse. Modular inversions are commonly computed using a binary GCD based algorithm which is a well-known side-channel target—see for example: [AGS07, ACSS17, PGB17, WSB18, Ald+19a]. Section 5 and Section 6 expand on these facts.

Note that in Algorithm 1 the execution of a point addition is only performed when the i-th bit of k is set. This property makes implementations of this algorithm highly vulnerable to SCA if an adversary can distinguish a double operation from an addition, yielding the binary representation of k. In Section 6, during a real-world instance of the projective coordinates attack, we demonstrate a SCA-hardened version of this algorithm in the open-source library `libgcrypt` is vulnerable to this attack.

Algorithm 1: *Double-and-Add* scalar multiplication

Input: Integers k and elliptic curve point G
Output: $R = kG$

$R = \mathcal{O}$
for $i = \lfloor \log_2(k) \rfloor$ **downto** 0 **do**
  $R = 2R$
  if $k_i = 1$ **then** $R = R + G$
return R

2.2 ECDSA: The Elliptic Curve Digital Signature Algorithm

ECDSA is the elliptic curve variant of the digital signature algorithm standardized by NIST [Fip]. Algorithm 2 shows pseudocode for the ECDSA signature generation procedure. This algorithm generates a digital signature for a public message (m) using the secret private key (α), where h is the output of a known hash function applied to the message m.

Algorithm 2: ECDSA signature generation

Input: Message m, private key α, domain parameters of elliptic curve E (e.g., G, p, n)
Output: Digital signature (r, s) on m under α

$h = \text{HASH}(m)$
$k =$ random s.t. $0 < k < n$
$(x, y) = kG$
$r = x \bmod n$
$s = k^{-1}(h + r\alpha) \bmod n$
return (r, s)

Each generated signature involves selecting a random *secret* nonce k satisfying $0 < k < n$,

performing a scalar multiplication of this nonce with the elliptic curve generator point G of prime order n, and reducing the resulting value x modulo n [Fip]. At line 5, the linear part of the signature generation computes the modular inverse of k and uses it to calculate the public value s.

We are interested in the scalar multiplication kG, more specifically, in the projective coordinates resulting from the implementation of that operation. This can be achieved by either obtaining the full Jacobian representation (X, Y, Z), or obtaining the affine representation (x, y) and one of the coordinates in the Jacobian representation that allows easily deriving the others.

As mentioned earlier, R in Algorithm 1 is usually represented using projective coordinates to speed up computations [HMV04]. However, ECDSA requires R to be returned in affine coordinates, therefore a conversion from projective to affine coordinates takes place before returning R in Algorithm 1.

On the other hand, it is trivial to compute *any* projective representation given the output point R in its affine representation; there are $p - 1$ of them [HMV04]. However, for the projective coordinates attack described in Section 3 the attacker is interested in a very specific representation: exactly the one stored in R before coordinate conversion takes place.

An ECDSA signature does not explicitly contain all the information needed for the projective coordinates attack, nevertheless it is possible to obtain part of it. In fact, it is possible to obtain the output point in affine coordinates by using the signature verification identity (3), where the point Q is the public key [Fip].

$$(x, y) = kG = hs^{-1} \cdot G + rs^{-1} \cdot Q \tag{3}$$

Moreover, the adversary critically needs to gather some information from the Jacobian point representation obtained during the computation of kG. This Jacobian point cannot be obtained directly from the ECDSA output, therefore ECDSA is not vulnerable to the projective coordinates attack at the specification level; in contrast to the division-free Schnorr signature scheme analyzed in the original paper where the projective coordinates of the output point are public [NSS04].

Hence, to be able to apply the projective coordinates attack to ECDSA, the adversary must gather some information about the Jacobian projective representation of kG by using other means such as an implementation-based attack.

2.3 Projective coordinates attack and ECDSA: previous works

Regarding implementation attacks, to the best of our knowledge, there is only one work [Mai+13] that aims at recovering information from the Jacobian representation of kG. The authors proposed three fault models under which it is possible to recover the coordinate Z. Two of them require that the same scalar value is processed several times, inserting a fault for each of them. However, in ECDSA the same scalar condition is not fulfilled as the nonce k is generated at random for each signature.

In terms of SCA, no work in the public literature has investigated how an adversary can recover this information, a gap that this paper starts to fill. Protecting every bit of k in ECDSA is mandatory to hold security, because even partial leakage can lead to the recovery of the private key using well-known lattice techniques [HS01, NS03, BT11, Ben+14].

Scalar multiplication has received generous attention w.r.t. SCA since the attack vector's inception, and several good surveys exist on the topic [Fan+10, FV12, Dan+13]. But vulnerabilities on other operations have emerged recently, such as in the nonce inversion operation [ACSS17, PGB17], and the modular reduction after $h + r\alpha$ [Rya19].

The projective coordinate attack of Naccache, Smart, and Stern [NSS04] has received very little attention by the scientific community, not only regarding ECDSA but ECC

in general. Therefore, the information leakage from the projective representation of the output point during scalar multiplication is rarely perceived as a threat.

3 Projective coordinates leak

This section describes the projective coordinates attack introduced by Naccache, Smart, and Stern [NSS04]. Assume the adversary knows the projective coordinates of a point R resulting from $R = kG$, i.e., $R = (X, Y, Z)$. For example, this is possible in cryptosystems like ECDSA, where an attacker can obtain the affine representation (x, y) using (3), in addition to some leak allowing Z recovery, hence obtaining $X = xZ^2$ and $Y = yZ^3$.

The main idea is to reverse the execution of the scalar multiplication algorithm starting from the output point R. Without loss of generality, consider Algorithm 1 describing the *Double-and-Add* scalar multiplication algorithm. In this algorithm, the execution of an addition operation is controlled by the i-th bit of the scalar k, while a point doubling operation is always executed. Therefore, if an attacker can distinguish at which iterations i an addition was executed, then the binary representation of k is trivially recovered.

Following Algorithm 1, let us denote $R_i = (X_i, Y_i, Z_i)$ a Jacobian representation of point R at iteration index i, where the output point is R_0. When the adversary reverses the computation done at the last iteration ($i = 0$), it aims at obtaining R_1 and so on. For reversing the scalar multiplication execution, the adversary must guess k_i and attempt to find a no-solution situation that allows to state with certainty the value of k_i.

If $k_0 = 0$, this implies that only a point doubling was executed. The adversary can reverse it by halving the affine representation of R_0, and computing Z_1 using the doubling formula (2) that yields:

$$Z_0 = 2Y_1Z_1 = 2y_1Z_1^4 \implies Z_1 = \sqrt[4]{\frac{Z_0}{2y_1}} \tag{4}$$

According to the above equation, the recovery of Z_1 requires computing a fourth root in $\mathbb{F}_p$, with solutions depending on $p \bmod 4$:

- $p \equiv 3 \bmod 4$: solutions exist with probability $\frac{1}{2}$, and when possible, obtaining two roots.
- $p \equiv 1 \bmod 4$: solutions exist with probability $\frac{1}{4}$, and when possible, obtaining four roots.

Therefore, each computed Z_1 allows to calculate X_1 and Y_1 using the affine representation of $R_1 = (x_1, y_1)$ obtained by halving R_0.

If, on the contrary, $k_0 = 1$, a point doubling and an addition were executed at $i = 0$ so the adversary must first reverse the addition operation and then the double. Let us denote $T = (X_T, Y_T, Z_T)$, the Jacobian point just before an addition occurs in Algorithm 1; while G in projective coordinates is represented as (X_G, Y_G, Z_G) and using lower case notation for their respective affine representations. It is worth noting that addition is usually performed using G represented in affine coordinates—i.e. $Z_G = 1$, to reduce the number of modular multiplications in (1), however we are using the full formula here to support analysis in Section 4.

Applying the Jacobian addition formula (1) yields an expression for Z_T:

$$Z_0 = Z_T^3 Z_G^3 (x_G - x_T) \implies Z_T = Z_G^{-1} \cdot \sqrt[3]{\frac{Z_0}{x_G - x_T}} \tag{5}$$

In this case Z_T requires computing a third root in $\mathbb{F}_p$, with solutions bound by the following rules:

- $p \equiv 2 \bmod 3$: a unique, computable solution always exists.
- $p \equiv 1 \bmod 3$: solutions exist with probability $\frac{1}{3}$, and when possible, obtaining one of three values.

Likewise, from each Z_T the adversary can compute the corresponding X_T and Y_T. To recover R_1, the adversary must reverse the double operation, as explained above, that is executed when $k_0 = 1$, but applying $T = (X_T, Y_T, Z_T)$ as input point.

Note that the computation of nth roots can lead to no solutions: for example, during guessing $k_0 = 0$ if no fourth roots are obtained, then the attacker concludes that this path was not executed, learning that $k_0 = 1$. In the case that both k_0 paths have solutions, no conclusions are drawn. However, the adversary can backtrack to previous iterations of scalar multiplication looking for a no-solution path that allows to learn more bits of k. It is worth noting that each computed nth-root generates a candidate point for R_i or T_i, therefore the solution tree can grow very quickly. In general terms this implies that the number of bits of k that can be recovered is small, but sometimes sufficient to apply lattice-based cryptanalysis to recover the ECDSA private key using several such signatures (as we practically demonstrate later in Section 6).

Naccache, Smart, and Stern [NSS04] provide some experimental results about the probability of recovering some bits of k for different values of p, concluding that the most vulnerable case is when $p \equiv 1 \bmod 12$. However, it is worth noting that the probability of recovering a number of bits depends on several factors such as:

- The characteristic of the curve's finite field.
- The projective coordinates representation (e.g., classical projective, Jacobian, etc.).
- The scalar multiplication algorithm.
- The elliptic curve form (e.g., Weierstrass, Montgomery, twisted Edwards, etc.).

These factors define the formulae for recovering Z_i and Z_t, hence the nth roots that must be computed (if any after all). It is thus difficult to give a generic approximation about the success rate of the attack, with so many implementation details at play. Therefore, instead of following a case-by-case approach we adopt a more general one that aims at showing that this vulnerability represents a potential security threat. We do this by (1) presenting an analysis of several libraries with respect to the potential impact of the attack (Section 5); and (2) providing experimental results of an *end-to-end* attack against the ECDSA implementation of `libgcrypt`, a widely deployed open source cryptography library (Section 6). The rationale behind this approach is that this leakage source is relative easy to mitigate. Therefore, the use of a simple to implement countermeasure is more important than providing a case-by-case analysis of how vulnerable a particular implementation is.

4 Mitigations analysis

To mitigate the projective coordinates attack, the original authors proposed two countermeasures. The first—left as a challenge to the research community—proposes to transform the point (X, Y, Z) with $(X, \epsilon Y, \epsilon Z)$, where ϵ is chosen at random in $\{-1, 1\}$. Nevertheless, Maimut et al. [Mai+13] showed that this countermeasure only makes the attack harder but does not prevent it, as the attacker can apply the same attack principle and treat the unknown ϵ as another "secret bit" and execute the attack for both values of ϵ and backtrack seeking a no-solution path, like in the original attack.

The other countermeasure proposed by both [NSS04] and [Mai+13], is to randomize the coordinate representation of the *output point*. The idea is to randomize the point (X, Y, Z)

by $(\lambda^2 X, \lambda^3 Y, \lambda Z)$ for λ chosen at random from $\mathbb{F}_p^*$. Using this method, the attacker has no information about the original Z.

Another line of defense is providing SCA resistance to the primitives that handles the output point like the employed modular inversion algorithm. One plausible approach is to replace the use of input-dependent execution flow binary GCD based algorithms by a Fermat's Little Theorem (FLT) instance. The FLT allows to compute the inverse of Z using the modular exponentiation $Z^{-1} = Z^{p-2} \bmod p$. Using this approach, even a modular exponentiation algorithm that leaks the exponent can be used since $p - 2$ is public.

4.1 Analysis of the randomized G countermeasure

In [Sto06], the authors proposed a countermeasure based on the randomization of the projective representation of the generator point G in Algorithm 1, following the rationale that it would prevent an attacker from reversing the addition operation using (5). However, it does not protect as expected.

For example, consider an attacker that first guesses $k_0 = 1$, then it has to reverse the addition operation using (5), but in this case the projective representation of G is randomized, thus Z_G is unknown in (5). However, note that the cubic root computation in (5) does not involve Z_G, and the affine representation of $G = (x_G, y_G)$ is public. Therefore, if no cubic roots exist in (5), it means that $k_0 = 1$ is a no solution path.

Continuing with the attack, now the attacker guesses $k_0 = 0$, thus reversing a point doubling using (4) (that does not involve G). If this path leads to solutions, the attacker can infer that $k_0 = 0$. Therefore, backtracking to several iterations and applying this procedure could lead to the recovery of some zero trailing bits of k. We validated this procedure with 1000 simulated values of k, yielding no wrong recoveries while some trailing zeros bits of k were recovered.

This analysis is a good example about the attack adaptability and the challenges of threat assessment for each implementation. It evidences that instead of a careful analysis for each implementation, an easier and more secure approach is to use one or both of the proposed countermeasures.

4.2 Why the Montgomery ladder does not offer protection

In [Mai+13] the authors marked the Montgomery ladder scalar multiplication algorithm as not threatened by the projective coordinates attack. At first this seems like a plausible countermeasure. However, we demonstrate that it is not, providing analytical and experimental evidence that the attack could indeed by applied to Montgomery ladder-based implementations.

Algorithm 3 shows pseudocode of the Montgomery ladder for scalar multiplication. This algorithm updates two points R and S at every iteration. However, it only outputs the point R and this is the one that gets converted back to affine coordinates.

Considering an attacker that targets the projective to affine coordinates conversion (e.g., modular inversion of Z), it can only get information on the projective representation of R but nothing about S. The lack of information on S is the reason why the projective coordinates attack was considered not applicable to the Montgomery ladder in [Mai+13].

However, analyzing Algorithm 3 it can be observed that if $k_i = 0$ the update of R does not depend on S at all. Thus, if k has trailing zeros, during the processing of the corresponding iterations R will only get updated by $R = 2R$ (line 6).

Following the projective attack, the adversary first guesses $k_0 = 0$ and try to reverse the computation of $R = 2R$. If solutions exist the guess $k_0 = 0$ could be right. However, to be certain about it, the adversary needs that the guess $k_0 = 1$ leads to a no solution path. During this branch R is updated by $R = R + S$, where S is unknown to the attacker.

Algorithm 3: *Montgomery ladder* scalar multiplication

Input: Integers k and elliptic curve point G
Output: $R = kG$

```
1  R = G
2  S = 2R
3  for i = ⌊log2(k)⌋ − 1 downto 0 do
4      if k_i = 0 then
5          S = R + S
6          R = 2R
7      else
8          R = R + S
9          S = 2S
10 return R
```

Note that this scenario is very similar to the one analyzed in Section 4.1 that showed an unknown addition operand does not prevent the attack.

As demonstrated in that section, for Weierstrass curves represented using Jacobian projective coordinates, the detection of no solution paths based on an addition operation like $R = R+S$ does not require knowing the projective representation of S. Instead, it only requires knowing its affine coordinates (see (5)). Unfortunately, the adversary can know the affine representation of S by exploiting a Montgomery ladder invariant: $S = R + G$ [JY02]. Thus the affine coordinates of R and G are known to the adversary, and hence S.

Therefore, if during the guess of $k_0 = 1$, no cubic root exist in (5), the adversary learns that it is a no solution path, concluding that $k_0 = 0$ if this path indeed gave solutions. If the attacker recover $k_0 = 0$, it has a number of candidates for R, thus it can repeat the process for other iterations. This way the attacker can recover trailing zero-valued bits of k. We have validated this variant of the projective coordinates attack with 1000 simulated values of k allowing to recover some bits with absolute certainty and free of errors.

It is important to remark that, similar to the analysis in Section 4.1, this attack against the Montgomery ladder applies to Weierstrass curves represented using Jacobian projective coordinates (see Section 2). Therefore, this attack could potentially be inapplicable to others curve forms and/or coordinates system where it could be impossible to detect no solution paths.

Similarly to the analysis performed in Section 4.1, porting the attack to work against the Montgomery ladder shows the difficulty of assessing if a given implementation is indeed threatened by this attack. This motivates a secure-by-default approach, applying a countermeasure regardless of an actual proven attack against a given implementation.

5 Threat analysis in the wild: open-source libraries

This section analyzes the susceptibility of several open source cryptography libraries to the projective coordinates attack. The aim is to provide a perspective about how prepared widely deployed software libraries are regarding this attack.

Table 1 summarizes this analysis. For each inspected library we used the latest version available and performed a source code-based static analysis to determine the scalar multiplication algorithm used in each case. We focused this study on simplified Weierstrass elliptic curves defined over prime finite fields, however the analysis can be extended to others as well.

The column $Z^{-1} \bmod p$ specifies the modular inversion algorithm used during the

Table 1: Open-source library status regarding the projective coordinates attack.

Library	Version	Scalar Multiplication	$Z^{-1} \bmod p$	Countermeasures	Threatened
Libgcrypt	1.8.5	Double-and-add always	BEEA	Nonce padding	Yes
MatrixSSL	4.2.2	Sliding Window	BEEA	-	Yes
Mozilla NSS**	3.46.1	wNAF	BEEA	Nonce padding	Yes
Crypto++	8.2.0	Sliding Window	BEEA	Nonce padding	Yes
Cryptlib	3.4.5	wNAF	BEEA	-	Yes
mbedTLS	2.16.3	SCA-Hardened Comb	BEEA	Starting point randomization	Yes
Libtomcrypt	1.18.2	Montgomery ladder	BEEA	-	Yes
wolfSSL	4.2.0	Montgomery ladder	BEEA	-	Yes
OpenSSL	1.1.1d	Montgomery ladder*	FLT	Starting point randomization	No
LibreSSL	3.0.2	Montgomery ladder	EEA	Starting point randomization	No
BoringSSL	5d62952	Sliding Window	FLT	-	No
Nettle	3.4.1	Sliding Window	CT-BEEA	-	No

*OpenSSL uses Montgomery ladder for all curves except `secp224r1`, `secp256r1`, and `secp521r1` that use a windowed scalar multiplication.
**NSS data is for `secp384r1` and `secp521r1` (`secp256r1` uses FLT to compute the targeted inversion).

projective to affine coordinates conversion. We used this operation as a reference to identify where there could be leakage in the output point projective representation for the ECDSA scalar multiplication kG result. This decision is supported by the fact, evidenced in Table 1, that modular inversions are often computed using insecure variants of the Binary Extended Euclidean Algorithm (BEEA).

The "Countermeasures" column refers to those protections in the ECDSA paths that have an impact on the scalar multiplication algorithm. This means that other countermeasures like blinding k to protect its inversion in ECDSA are not reflected in Table 1 as they do not have any impact on the scalar multiplication kG. Also, as the scalar multiplication algorithms have their own column in Table 1 we omit them from the "Countermeasures" column.

We titled the last column *Threatened*, instead of vulnerable, because to our consideration a vulnerability is only present when it is *demonstrated* for the targeted implementation. In this regard we have made such a demonstration on the `libgcrypt` library, where we developed an end-to-end attack that leads to the recovery of ECDSA private keys (see Section 6). This last column shows those libraries whose ECDSA implementations are threatened (i.e. potentially vulnerable) to the projective coordinates attack due to leakage during the modular inversion of Z in $\mathbb{F}_p^*$, being `libgcrypt` one of them.

At the end of Section 3 we summarized that the projective coordinates attack depends on several factors. Therefore, performing end-to-end attacks on every possible implementation is a time-consuming task with several degrees of freedom, especially since Table 1 combinations are not exhaustive at all!

We instead choose to analyze mitigation strategies that are independent of these factors. In this regard, Section 4 describes generic countermeasures to thwart this attack without a significant performance loss. This rationale follows our main objective on providing an analysis of the projective coordinates threat in widely deployed cryptography libraries considering the little attention that this attack has received by the academy and industry.

Analysis of scalar multiplication algorithms. In the original work by Naccache, Smart, and Stern [NSS04], the authors describe the attack when the *Double-and-Add* algorithm is in use. Additionally, they also provide simulation results for the *Sliding Window* algorithm. Later, Maimut et al. [Mai+13] complemented this study by analyzing other scalar multiplication algorithms, like variants of the first two, in addition to algorithms based on the *Montgomery ladder*.

Following the analysis and [NSS04, Mai+13], in addition to our results on the Montgomery ladder in Section 4.2: We conclude that the scalar multiplication algorithms summarized in Table 1 do not offer natural protection against this attack.

Analysis of modular inversion algorithms. Table 1 shows four algorithms used to invert Z. BEEA represents those based on a variable-time binary GCD algorithm. Library-specific implementations vary, but after a detailed analysis, we can confirm that those labeled as BEEA can be targeted using well-known SCA techniques [AGS07, ACSS17, PGB17], hence potentially SCA vulnerable.

Related to the BEEA, we labeled Euclid's classic algorithm for computing modular inverses as EEA (Extended Euclidean Algorithm). LibreSSL uses this algorithm as a side-channel hardened modular inversion function, even though the execution flow of this algorithm also depends on its inputs. No evidence of a side-channel attack on this algorithm variant has been published yet, so its resilience against SCA remains as an open problem. Therefore, to the best of our knowledge, this algorithm is considered safe in this context.

An interesting approach is the one used by the Nettle library. Nettle uses a constant-time BEEA (CT-BEEA), that removes branches from the algorithm to offer SCA protection. Therefore, it is considered safe to be used for computing the inversion of Z.

The fourth approach is orthogonal to GCD-based modular inversion algorithms and is based on the FLT. As analyzed in Section 4, it is considered a safe way for computing modular inverses when the modulus is public.

Analysis of countermeasures. Table 1 shows two countermeasures in place in the analyzed libraries. *Nonce padding* ensures that k always has the same bit-length by adding a multiple of n to it. Section 6 describes this countermeasure in detail in the context of projective coordinates attack vulnerability demonstration for the `libgcrypt` library, showing it is possible to bypass this countermeasure, having no effect on the attack.

On the other hand, the countermeasure labeled *Starting point scalar randomization* follows the same mathematical principle of the projective randomization countermeasure described in Section 4. However, there is an important difference that makes the way it is used in the mbedTLS, OpenSSL, and LibreSSL libraries meaningless to thwart the projective coordinates attack. These libraries apply projective randomization to the starting point, whereas the countermeasure to prevent this attack requires applying this randomization to the *output* point before the projective to affine conversion. This is due to the fact that the projective coordinates attack uses the output point to reverse the scalar multiplication algorithm, looking for paths with no solutions: paths that can exist even when the starting point is randomized. For these reasons, the countermeasure column is red-colored to indicate that they do not have any influence on the attack.

Conveniently, the fact that these libraries have already implemented the projective randomization countermeasure makes it extremely straightforward to add protection against the projective coordinates attack by using this primitive before the projective to affine coordinates conversion. Even in safe inversion cases, we feel this low-cost countermeasure provides future proof hardening.

6 End-to-end attack: `libgcrypt` ECDSA

This section demonstrates the projective coordinates attack against the widely deployed open source library `libgcrypt`. We target ECDSA signature generation and perform an extensive experiment with the NIST elliptic curve `secp256r1` [Fip].

Our source code analysis for the ECDSA implementation in this library reveals that, for Weierstrass curves, the scalar multiplication is performed using Jacobian projective coordinates and employing the *double-and-add always* algorithm, as depicted in Algorithm 4[1].

[1] https://github.com/gpg/libgcrypt/blob/d9c41830/mpi/ec.c#L1691

This is an SCA-hardened version of the *double-and-add* algorithm, where an addition is executed at every iteration, and an SCA-secured conditional assignment ensures that the point R gets updated with the correct value depending on the value of k_i. This means in an abstract point of view that Algorithm 4 is essentially a *double-and-add* algorithm, and the projective coordinates attack described in Section 3 applies without further changes, even with this SCA mitigation.

Algorithm 4: *Double-and-Add always*

Input: Integers k and elliptic curve point G
Output: $R = kG$

$R = \mathcal{O}$
for $i = \lfloor \log_2 k \rfloor$ **downto** 0 **do**
  $R = 2R$
  $T = R + G$
  $R =$ `cond_assign`(T, R, k_i)
return R

The library performs the conversion from Jacobian to affine coordinates using the function `_gcry_mpi_ec_get_affine`[2]. Computing the modular inverse of Z in $\mathbb{F}_p^*$ using the function `_gcry_mpi_invm`[3], which is a variant of the well-known BEEA [Knu98].

The BEEA has input-dependent execution flow, hence a suitable SCA target [AGS07, ACSS17, PGB17, WSB18, Ald+19a]. An attacker can recover Z using SCA techniques during the computation of $Z^{-1} \bmod p$, hence obtaining the Jacobian representation of the resulting point kG during ECDSA signature generation.

This means that from the knowledge of Z the adversary can use the projective coordinates attack to recover some bits of k. Repeating the process for a set of signatures, the attacker can build a lattice to recover the private key by solving an instance of the well-known hidden number problem [HS01, NS03, BT11, Ben+14].

The remainder of this section evidentiates the vulnerability of the `libgcrypt` library to a projective coordinates attack in the context of the ECDSA scheme. For this task we captured 100k independent traces corresponding to the generation of ECDSA signatures. This extensive experiment allows us to estimate the number of recovered bits probability mass function (pmf) for the projective coordinates attack and also estimate the average number of signatures an attacker would need to recover the private key. The attack roadmap as covered in the following subsections is:

1. Side-channel attack during the computation of $Z^{-1} \bmod p$.
2. Results of applying the projective coordinates attack against `libgcrypt` ECDSA implementation with curve `secp256r1`.
3. Lattice attack for private key recovery.

6.1 Side-channel attack on `libgcrypt` BEEA

We frame our experiments on a `libgcrypt` instance running as part of a *shielded* environment provided by Intel Software Guard Extensions (Intel SGX). This technology has received generous attention recently due to its promising approach for securing computations on Intel CPUs. This has motivated academia to investigate security features of open source libraries like `libgcrypt` when secured with this technology in the face of side-channel adversaries.

[2] https://github.com/gpg/libgcrypt/blob/d9c41830/mpi/ec.c#L1051
[3] https://github.com/gpg/libgcrypt/blob/d9c41830/mpi/mpi-inv.c#L160

For instance, Shinde et al. [Shi+16] provide security analysis of several symmetric and public key algorithm implementations in `libgcrypt` and OpenSSL libraries. Moghimi, Irazoqui, and Eisenbarth [MIE17] introduced the CacheZoom attack, showing how the `libgcrypt` AES implementation leaks when running inside an Intel SGX enclave. On the other hand, Van Bulck et al. [VB+17] and Wang et al. [Wan+17] tested the side-channel resistance of the `libgcrypt` EdDSA implementation hardened with Intel SGX.

Intel SGX provides software implementations the possibility to isolate computations in a secure environment called *enclave* [Int19b, CD16]. This technology offers authenticity and integrity protections to processes (called enclaves) running even in the presence of a malicious OS. On the other hand, Intel SGX does not prevent SCA leaks, and it leaves the task to protect against this attack vector to developers. This highlights the importance of the projective coordinates threat especially because Intel SGX considers an adversarial OS, which means that under the Intel SGX threat model, an SCA-capable attacker controls the OS, and can have access to reliable leakage sources.

For instance, the *controlled-channels attack* proposed by Xu, Cui, and Peinado [XCP15] provides access to the sequence of memory pages executed by the victim enclave, providing a leakage source of 4 KB granularity that can be used to track the enclave execution [Wan+17, Shi+16, VB+17, WSB18]. This attack relies on Intel SGX which leaves control of its memory pages to the untrusted OS. Therefore, an adversarial OS can mark a memory page with SCA relevance as *non-executable* and monitor it. A triggered fault indicates the execution of the monitored page [XCP15]. Repeating the process for a set of memory pages allows the adversary to track the sequence of executed memory pages, thus potentially leaking secret data processed by the enclave.

The `SGX-Step` framework proposed by Van Bulck, Piessens, and Strackx [VBPS17] provides a toolbox for developing *page-fault* based attacks, allowing carrying out this attack against Intel SGX enclaves. In order to test the security of the `libgcrypt` Jacobian to affine conversion regarding SCA, we integrated `SGX-Step` into the `Graphene-SGX` framework [TPV17]. `Graphene-SGX` allows running unmodified code inside an Intel SGX enclave, providing a straightforward approach to execute `libgcrypt` code in an enclave and assess its SCA resistance. It is worth noting that the `Graphene-SGX` framework is not a requirement for the attack, since `libgcrypt` can be ported to Intel SGX in the same way other libraries such as OpenSSL or mbedTLS have been [Int19a, Sil19].

Threat model and experiment setup. The experiments were performed on a desktop workstation using an Intel i7-7700 processor with SGX support running Ubuntu 18.04 LTS with kernel 5.0.0-29. Linux SGX support is achieved using Intel SGX SDK v2.2 and Linux SGX driver v2.1. `SGX-Step` v1.2.0 was integrated into `Graphene-SGX` (commit `58e6087`) as discussed before. The vulnerability is demonstrated against an SGX enclave that generates ECDSA signatures using `libgcrypt` v1.8.5 as cryptography backend.

Following the Intel SGX threat model with an adversarial OS, an adversary can trace the sequence of pages executed by the enclave [XCP15, VBPS17, CD16]. It is assumed that the adversary knows all page address(es) of every function of interest in the compiled enclave hence their corresponding memory pages. This can be obtained by static analysis, or by reverse engineering, thus it is not a limiting assumption.

As mentioned in Section 6, Jacobian to affine coordinates conversion in `libgcrypt` involves the computation of a modular inverse using a variant of the BEEA. Figure 1 shows a snippet of the source code, where only the execution flow control statements and function calls are left, taking into account that a page-fault attack has a 4 KB granularity and this code fits in one page.

Previous work analyzes the relation between classical BEEA execution flow with its inputs [AGS07, ACSS17], concluding that the adversary needs to gather how many times a division by two (right-shift) is executed at every iteration, in addition to the result of an

```
1     do {
2         do {
3             if( mpi_test_bit(t1, 0) )
4                 mpi_add(t1, t1, v);
5             mpi_rshift(t1, t1, 1);  /* Zi */
6             mpi_rshift(t3, t3, 1);
7         } while( !mpi_test_bit( t3, 0 ) );
8
9         if( !t3->sign ) {
10            mpi_set(u1, t1);        /* Xi = u */
11            mpi_set(u3, t3);
12        } else {
13            mpi_sub(v1, v, t1);     /* Xi = v */
14              ....
15            mpi_set(v3, t3);
16              ....
17        }
18
19        mpi_sub(t1, u1, v1);
20        mpi_sub(t3, u3, v3);
21
22        if( t1->sign )
23            mpi_add(t1, t1, v);
24
25    } while( mpi_cmp_ui( t3, 0 ) );
```

Figure 1: BEEA variant as implemented in `libgcrypt`.

integer comparison that controls the execution of two branches.

The structure of the `libgcrypt` variant differs slightly from the classical BEEA. The right-shift related execution flow can be easily identified at line 6 while the integer comparison operation is not explicit. However, it can be easily verified that the condition instruction at line 9 in Figure 1 is equivalent to the integer comparison in the classical BEEA. Therefore, it is possible to apply previous SCA analysis results on classical BEEA [AGS07, ACSS17] to the `libgcrypt` variant.

One important difference between classical BEEA and the `libgcrypt` variant is that in the latter, the conditional statement at line 9 results in *unbalanced* branches, regarding the number of function calls to `_gcry_mpi_set`. On the other hand, in the classical version the equivalent branches are balanced regarding function calls.

This difference has a significant impact on the security of the `libgcrypt` implementation regarding SCA. In the SCA realm it is well-known that *unbalanced branches* facilitate the distinction of execution flow control statement results. Applied to the `libgcrypt` analysis, the result of the condition statement at line 9 can be inferred by an adversary monitoring how many times `_gcry_mpi_set` is called, while in the classical version it is one of the limitations that an attacker must face to increase the number of recovered bits [ACSS17, PGB17].

Therefore, adapting previous SCA results of classical BEEA to `libgcrypt` allows stating that the adversary needs to recover how many times t_3 is divided by two at each algorithm iteration, in addition to the result of the condition statement at line 9. Gathering this execution flow information for all executed iterations, it is possible to recover algorithm inputs in polynomial time as proved in [AGS07, ACSS17].

Trace capture. Employing the `SGX-Step` framework it is possible to track the sequence of executed memory pages by the victim enclave [VBPS17]. This allows the adversary to obtain a side-channel trace related to the execution flow during the inversion of $Z^{-1} \bmod p$. For recovering the execution flow corresponding to this computation us-

Table 2: Tracked functions and their memory pages.

Function	Memory page
`_gcry_ecc_ecdsa_sign`	0xa1000
`_gcry_mpi_invm`	0xcf000
`_gcry_mpih_rshift`	0xd8000
`_gcry_mpi_set`	0xd5000
`_gcry_mpi_add`	0xcd000
`_gcry_mpih_sub_n`	0xd8000

ing the `_gcry_mpi_invm` function, we monitored the memory pages shown in Table 2. Tracking `_gcry_ecc_ecdsa_sign` allows identifying different calls to the BEEA function `_gcry_mpi_invm` during this procedure and isolate the one that corresponds to $Z^{-1} \bmod p$.

A trace about the tracked memory pages reveals that during an ECDSA signature generation in `libgcrypt` the BEEA is executed four times, corresponding to the computations of:

1. The inverse of the secret mask b related to [Rya19].
2. The inverse of 2 modulo p (used internally during the scalar multiplication).
3. Projective to affine conversion: $Z^{-1} \bmod p$.
4. The inversion of the secret nonce k.

Regarding the projective coordinates attack, we are interested in the third one. Therefore, the subtrace corresponding to the sequence of executed memory pages during the computation of $Z^{-1} \bmod p$ should have information about its execution flow. Two of the other three executions of the BEEA also leak secret information—we will expand on this later.

Monitoring `_gcry_mpih_rshift` allows to recover the number of times t_3 is divided by two, while the number of executed `_gcry_mpi_set` gives information about the condition statement result at line 9 as required. However, auxiliary memory pages should be monitored to distinguish additional calls to these functions from other functions. For instance, the `_gcry_mpi_set` is a lower-layer multiprecision integer arithmetic function that copies integer objects. Therefore, it is expected to be called frequently by other functions in Figure 1. Hence, the adversary must distinguish between them to reliably identify the calls that belong to the highlighted lines in Figure 1: this is where the auxiliary memory pages help.

For example, let us represent a sequence of executed pages by `".S.S.LSLS."` where each character represents a memory page (arbitrary). In this sequence it is possible to distinguish the first two `S` symbols from the last two by observing their preceding executed pages. Employing this approach it is possible to reliably count the number of `_gcry_mpi_set` calls and recover the condition result at line 9. It also allows a straightforward distinction of `_gcry_mpih_rshift` from `_gcry_mpih_sub_n` executions even if they share a memory page.

The tracked memory pages needed to succeed could vary between `libgcrypt` versions and toolchains used to build it. However, without claiming a formal proof, the attacker can search for auxiliary pages that allows identifying the execution flow as required [XCP15]. During this research, we targeted the function `_gcry_mpi_invm` from three `libgcrypt` versions which have different memory page mappings, always being able to reliably recover the execution flow of this function.

After processing the 100k traces obtained during the computation of ECDSA signatures, it was possible to identify the computation of $Z^{-1} \bmod p$ in all of them. Processing the

sequences of tracked pages it was possible to recover the execution flow of all BEEA executions without any error[4] leading to a reliable recovery of Z in all 100k traces.

Attack requirement and noise influence relation. The projective coordinates attack only requires that the adversary knows the projective representation of the output point. This point should be error free, because if it does not, the adversary would recover incorrect bits of k. In this regard, other side-channels could be used to recover the targeted point, but if this channel is noisy, the adversary should reduce noise influence to the minimum by for example discarding suspicious traces or using leakage redundancy. The main idea, is to mitigate the noise as much as possible such that trace oversampling does not make the attack impractical.

Additional vulnerabilities. As analyzed previously in this section, an ECDSA trace reveals four use cases of the BEEA. In addition to the projective to affine conversion (item 3), two additional use cases of the BEEA (item 1 and item 4) also leak information during the ECDSA signature generation path.

During our experiments, we also found *two additional vulnerabilities* in the ECDSA path. These vulnerabilities rely on the same insecure `libgcrypt` BEEA implementation. As a response to the attack in [Rya19], a proposed countermeasure protects the ECDSA computation $(h + r\alpha) \bmod n$. The countermeasure relies on multiplying h and r by a secret mask b, before involving computations with the private key α. However, to ensure ECDSA correctness, $(h + r\alpha)$ should be multiplied by $b^{-1} \bmod n$, where this inversion is computed using the insecure BEEA in `libgcrypt`. Notice that knowing the mask b allows launching the attack described in [Rya19].

The other vulnerability is present during the `libgcrypt` ECDSA inversion of k, computed with the same function used to invert Z (`_gcry_mpi_invm`), therefore the previous analysis described applies directly.

For the sake of completeness we also targeted the nonce inversion vulnerability, by exploiting the same BEEA primitive and obtaining similar results, full recovery of the k in all 100k traces. More importantly, this last vulnerability represents a bigger threat to `libgcrypt` than the projective coordinates one, as it only requires a single trace to succeed. However, the purpose of this paper is not about merely exploiting the `libgcrypt` library, but about providing an analysis of the projective coordinates threat status, filling a gap between its original description by Naccache, Smart, and Stern [NSS04] and the adoption of countermeasures to thwart it in real-world applications. A plausible approach for fixing both vulnerabilities is using the FLT for computing the corresponding inverses (see Section 4).

6.2 Projective coordinates attack results

Regarding the projective coordinates attack described in Section 3, the `libgcrypt` ECDSA implementation has another important difference. The `libgcrypt` implementation has a countermeasure to fix the length of k as a response to [BT11], and more recently to Minerva[5].

This countermeasure modifies the value of k by adding a multiple of n to it; the scalar multiplication algorithm processes this padded version. However, this countermeasure has no effect on the projective coordinates attack results, but the adversary needs to know that it is in place to be able to revert it, as detailed later in Section 6.3.

After processing the 100k traces and extracting the correct value of Z in all of them, we launched the projective coordinates attack in the computer algebra system `sage` for

[4] Verified using the known private key used in the experiment.

[5] https://minerva.crocs.fi.muni.cz/

each Z. We limited the backtracking to eight iterations, implying that the best leakage we obtained is eight bits of k (see Section 3). This decision was made considering the next attack phase (lattice-based cryptanalysis in Section 6.3) that allows recovering the private key even with fewer than eight bits leaked per signature.

The projective coordinates attack outputs the number of bits recovered, in addition to the value of those bits. With the extensive set of 100k traces it was possible to estimate the leak in terms of number of recovered bits probability mass function, allowing us to estimate the probability to recover at least ℓ bits of k.

Table 3 shows these probabilities for different values of ℓ. This table suggests that the probability of recovering ℓ bits decreases very quickly as this variable increases. However, as demonstrated in Section 6.3 these results are sufficient to compromise the security of ECDSA using a reasonable number of traces.

Table 3: Probability of recovering at least ℓ bits of k.

ℓ	1	2	3	4	5	6	7	8
Pr[leak $\geq \ell$] (%)	66.62	13.75	4.55	1.50	0.47	0.16	0.05	0.02

6.3 Private key recovery using lattices

The previous projective coordinates attack provides us partial information of the values of k used to generate each of the 100k ECDSA signatures. For each signature i this attack produces a tuple (c_i, ℓ_i), where ℓ_i denotes the number of recovered LSBs of the corresponding k_i, and c_i the value of those bits. Thus, we can combine and use this information to formulate a *Hidden Number Problem (HNP)* [BV96] instance and solve it using lattices, which upon success leads to the recovery of the long-term ECDSA private key, α.

We follow the formalization of [NS02, NS03] with the construction by Benger et al. [Ben+14, Sec. 4]. More specifically, in `libgcrypt` the nonce $\hat{k}$ is padded to protect against remote timing attacks [BT11], resulting in

$$k = \hat{k} + \rho \cdot n \tag{6}$$

where n is the group order, and ρ is the minimum value in $\{1, 2\}$ that satisfies:

$$\lfloor \lg(\hat{k} + \rho \cdot n) \rfloor = \lfloor \lg(n) + 1 \rfloor.$$

NIST standardized elliptic curves define n close to a power of two, implying the value of ρ is known in advance with overwhelming probability [Fip, Ben+14]. Regarding NIST curve `secp256r1`, it can be easily verified that $\rho = 1$, hence we can simplify the above equation to $k = \hat{k} + n$. Therefore, for a signature index i, the leakage about its corresponding k_i can be described as:

$$c_i \equiv \hat{k_i} + n \pmod{2^{\ell_i}}$$
$$\hat{k_i} \equiv c_i - n \pmod{2^{\ell_i}}$$

where ℓ_i and c_i are the information on k_i produced by the projective coordinates attack.

We set $a_i = c_i - n \pmod{2^{\ell_i}}$, thus resulting in equations $k_i = 2^{\ell_i} b_i + a_i$ where ℓ_i and a_i are known. Define the following values known by an attacker:

$$t_i = \lfloor r_i / (2^{\ell_i} s_i) \rfloor_n$$
$$u_i = \lfloor (a_i - h_i / s_i) / 2^{\ell_i} \rfloor_n + n / 2^{\ell_i + 1}$$

where $\lfloor x \rceil_n$ denotes modular reduction of x to the interval $[0\,..\,n-1]$ and $|x|_n$ to the interval $[-(n-1)/2\,..\,(n-1)/2]$. We obtain

$$v_i = |\alpha t_i - u_i|_n \leq n/2^{\ell_i+1},$$

i.e. integers ω_i exist such that $\text{abs}(\alpha t_i - u_i - \omega_i n) \leq n/2^{\ell_i+1}$ holds. The v_i result in a smaller value (by a factor of 2^{ℓ_i+1}) since this difference is closer than a uniformly random value from $[1\,..\,n-1]$. We can now proceed to perform a lattice attack: recover α given many (t_i, u_i) pairs.

To find the solution to the lattice we use the embedding strategy by Goldreich, Goldwasser, and Halevi [GGH97, Sec. 3.4] to reduce CVP approximations to Shortest Vector Problem (SVP) approximations. For CVP consider the following vectors

$$\begin{aligned}\vec{x} &= (\omega_1, \ldots, \omega_d, \alpha)\\ \vec{y} &= (2^{\ell_1+1}v_1, \ldots, 2^{\ell_d+1}v_d, \alpha)\\ \vec{u} &= (2^{\ell_1+1}u_1, \ldots, 2^{\ell_d+1}u_d, 0)\end{aligned}$$

and with the rational $d+1$-dimension lattice generated by the rows of the matrix

$$B = \begin{bmatrix} 2^{\ell_1+1}n & 0 & \ldots & \ldots & 0 \\ 0 & 2^{\ell_2+1}n & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \ldots & 0 & 2^{\ell_d+1}n & 0 \\ 2^{\ell_1+1}t_1 & \ldots & \ldots & 2^{\ell_d+1}t_d & 1 \end{bmatrix}$$

we establish the relationship $\vec{x}B - \vec{u} = \vec{y}$. Solving the CVP with inputs B and $\vec{u}$ yields $\vec{x}$ and hence α.

Now, for the SVP consider the rational $d+2$-dimension lattice generated by the rows of the following matrix.

$$\hat{B} = \begin{bmatrix} B & 0 \\ \vec{u} & n \end{bmatrix}$$

There is a reasonable chance that lattice-reduced $\hat{B}$ will contain the short lattice basis vector $(\vec{x}, -1)\hat{B} = (\vec{y}, -n)$, revealing α. Inspired by Gama, Nguyen, and Regev [GNR10, Sec. 5] we use the randomization technique to extend the search space, shuffling the order of t_i and u_i and multiplying by a random sparse unimodular matrix between lattice reductions.

End-to-end attack results. One important metric to evaluate the impact of this attack on `libgcrypt` is the average number of signatures/traces an adversary needs to capture for recovering the private key. In this regard we estimated this metric using the 100k signatures/traces generated as part of this experiment.

Following an attacker perspective, we mimic an on-demand approach to capture traces, that is, the attacker only captures and processes a trace if needed. In theory the lattice-based cryptanalysis described below can work even with a 2-bit leak [NS03], however, exploiting this small (fewer than three bits) leak for 256-bit elliptic curves has not been demonstrated in practice [TTA18]. On the other hand, exploiting a leak of at least three bits is a common practice [Ben+14, PGB17, Tuv+18, Ald+19b].

For building a lattice to recover the private key we only used those traces that after applying the projective coordinates attack led to a leak of at least three bits. Hence, all traces that did not fulfill this requirement were discarded. Note that the estimated probability of recovering at least three bits is 4.55%. While this might seem low, it is

sufficient to recover the private key after capturing a reasonable number of traces, as demonstrated below.

As part of the lattice attack, the attacker must decide how many signatures are going to be used to build the lattice, which defines the lattice dimension $d + 2$. A *rule-of-thumb* is that the sum of known bits used to build the lattice aggregates up to the number of bits of n in (6), in this case 256 bits.

With these two metrics, we split the 100k signatures in smaller sets, s.t. the sum of leaked bits in each set is at least 256, and considering only those signatures with a minimum 3-bit leak. This means that each set has a group of signatures that add information to the lattice (at least three bits of leakage), and another group that does not (fewer than three bits of leakage). This approach resulted in 60 sets with an average total number of signatures of 1616. For each of these sets, we built an HNP instance, and its corresponding lattice, and we tried to solve them using the default lattice reduction algorithm in `sage v8.1`.

To parallelize the attack and to reduce computation time, we used the randomization technique by Gama, Nguyen, and Regev [GNR10] to build 1000 different lattices per HNP instance (i.e. set of signatures) and we scheduled them to run concurrently on a cluster for up to four hours. After processing all sets, in 51 of them the private key was successfully recovered, giving an estimated success rate of 85%.

Additionally, we performed a second experiment in which we split the 100k signatures in 50 sets of 2000 signatures, running the lattice attack on each on them. However, in this experiment, we only used one lattice per HNP instance, again, running for up to four hours. The experiment yielded the private keys for all 50 sets. Note that this approach is computationally friendlier, and perfectly doable by a modern desktop workstation.

By combining both experiment results we can convincingly state that the average number of traces that an attacker would need to capture and process to succeed with very high probability is between 1616 and 2000. This demonstrates the attack feasibility and its impact on the `libgcrypt` library, as it is possible to recover an ECDSA private key with a reasonable number of traces.

7 Conclusion

Our analysis of several open-source cryptography libraries shows that some of them could be vulnerable to the projective coordinates attack. The lack of adoption of specific countermeasures to prevent this attack in these libraries reminds us how important is to fill the gap between academic results and industry.

This paper is another in a long list that demonstrate how input-dependent execution flow in cryptography primitives can allow attackers to bypass cryptography protections using SCA. The projective to affine coordinates conversion involves the computation of a modular inverse. The analysis of several open-source cryptography libraries revealed that this operation is frequently implemented using SCA-insecure algorithms such as the BEEA, opening the door for attackers using SCA techniques to recover the projective representation of a scalar multiplication output point, and finally performing the projective coordinates attack.

The success rate of Naccache, Smart, and Stern [NSS04] projective coordinates attack depends on several factors such as: (1) elliptic curve form; (2) finite field over which the EC curve is defined; (3) scalar multiplication algorithm employed; and (4) projective coordinates representation. Considering that each of these factors adds ample degrees of freedom, a general assessment of attack success rate is a non-trivial task. For this reason, we conclude that instead of evaluating the attack effectiveness against every possible factor combination, a practical approach is to adopt specific countermeasures to prevent this attack.

One of our recommendations is to apply projective coordinates randomization before converting a point to affine coordinates, a straightforward countermeasure for cryptography libraries that already include an algorithm for such a task. Another line of defense is to use an SCA-secure projective to affine coordinates conversion, especially for modular inversion operations that are often computed using insecure BEEA variants. A recommended approach to invert Z, is to perform modular inversion using FLT, a solution already deployed in some libraries. However, it is important to highlight that all algorithms involved in coordinate conversion should be protected.

Regarding countermeasures, we demonstrated that the Montgomery ladder is not a natural countermeasure against this attack, contrary to what was believed. We also demonstrated that randomizing the projective representation of the generator point G does not prevent the attack in some cases, providing experimental evidence for it.

As part of our experimental analysis, we developed an end-to-end attack on `libgcrypt`, successfully demonstrating the practicality of the projective coordinates attack. We targeted ECDSA signature generation using the elliptic curve `secp256r1`. During this analysis we captured 100k traces and estimated that an adversary would need less than 2000 traces to recover the ECDSA private key with high probability. We released part of our tooling and data in support of Open Science [APGB20].

This work highlights once more the need of adopting a secure-by-default approach in cryptography libraries, instead of only protecting what has been demonstrated vulnerable. To our consideration, the most important conclusion is that it is safer to adopt straightforward countermeasures like projective coordinate randomization than analyzing if a particular implementation is vulnerable. Especially considering that the number of variables involved in the projective coordinates attack and its adaptability to handle them make the vulnerability assessment a non-trivial task.

These challenges exacerbate when a cryptography primitive depends on several moving parts in a software library. Even something as common as an update process opens the possibility to an attack that was rendered meaningless in a previous ad-hoc vulnerability assessment. Therefore, a secure-by-default approach is preferred.

Responsible disclosure. We contacted the threatened library security teams, reporting the status in the face of the projective coordinates attack. In response to our findings `libgcrypt` developed a constant-time modular inversion algorithm, fixing the three reported vulnerabilities (and potentially others that exploits insecure modular inversion)[6]. WolfSSL followed the FLT approach for protecting against the projective coordinates attack (CVE-2020-11735)[7]. mbedTLS security team acknowledges the issue, tracking their solution with CVE-2020-10932.

Acknowledgments

We thank Tampere Center for Scientific Computing (TCSC) for generously granting us access to computing cluster resources.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 804476).

[6] Commits 20082ca9 and ada758e3.
[7] `https://github.com/wolfSSL/wolfssl/pull/2859`.

References

[ACSS17] Alejandro Cabrera Aldaya, Alejandro J. Cabrera Sarmiento, and Santiago Sánchez-Solano. "SPA vulnerabilities of the binary extended Euclidean algorithm". In: *J. Cryptographic Engineering* 7.4 (2017), pp. 273–285. DOI: 10.1007/s13389-016-0135-4. URL: https://doi.org/10.1007/s13389-016-0135-4.

[AGS07] Onur Acıiçmez, Shay Gueron, and Jean-Pierre Seifert. "New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures". In: *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*. Ed. by Steven D. Galbraith. Vol. 4887. Lecture Notes in Computer Science. Springer, 2007, pp. 185–203. DOI: 10.1007/978-3-540-77272-9_12. URL: https://doi.org/10.1007/978-3-540-77272-9_12.

[Ald+19a] Alejandro Cabrera Aldaya, Cesar Pereida García, Luis Manuel Alvarez Tapia, and Billy Bob Brumley. "Cache-Timing Attacks on RSA Key Generation". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.4 (2019), pp. 213–242. DOI: 10.13154/tches.v2019.i4.213-242. URL: https://doi.org/10.13154/tches.v2019.i4.213-242.

[Ald+19b] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. "Port Contention for Fun and Profit". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 870–887. DOI: 10.1109/SP.2019.00066. URL: https://doi.org/10.1109/SP.2019.00066.

[APGB20] Alejandro Cabrera Aldaya, Cesar Pereida García, and Billy Bob Brumley. *From A to Z: Projective coordinates leakage in the wild: research data and tooling*. Zenodo, Apr. 2020. DOI: 10.5281/zenodo.3752634. URL: https://doi.org/10.5281/zenodo.3752634.

[AVL19] Rodrigo Abarzúa, Claudio Valencia, and Julio López. *Survey for Performance & Security Problems of Passive Side-channel Attacks Countermeasures in ECC*. Cryptology ePrint Archive, Report 2019/010. https://eprint.iacr.org/2019/010. 2019.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model". In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2. URL: https://doi.org/10.1007/978-3-540-28632-5_2.

[Ben+14] Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. ""Ooh Aah... Just a Little Bit": A Small Amount of Side Channel Can Go a Long Way". In: *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. Lecture Notes in Computer Science. Springer, 2014, pp. 75–92. DOI: 10.1007/978-3-662-44709-3_5. URL: https://doi.org/10.1007/978-3-662-44709-3_5.

[BH09] Billy Bob Brumley and Risto M. Hakala. "Cache-Timing Template Attacks". In: *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*. Ed. by Mitsuru Matsui. Vol. 5912.

Lecture Notes in Computer Science. Springer, 2009, pp. 667–684. DOI: 10.1007/978-3-642-10366-7_39. URL: https://doi.org/10.1007/978-3-642-10366-7_39.

[BT11] Billy Bob Brumley and Nicola Tuveri. "Remote Timing Attacks Are Still Practical". In: *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*. Ed. by Vijay Atluri and Claudia Díaz. Vol. 6879. Lecture Notes in Computer Science. Springer, 2011, pp. 355–371. DOI: 10.1007/978-3-642-23822-2_20. URL: https://doi.org/10.1007/978-3-642-23822-2_20.

[BV96] Dan Boneh and Ramarathnam Venkatesan. "Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes". In: *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 129–142. DOI: 10.1007/3-540-68697-5_11. URL: https://doi.org/10.1007/3-540-68697-5_11.

[CD16] Victor Costan and Srinivas Devadas. "Intel SGX Explained". In: *IACR Cryptology ePrint Archive* 2016.86 (2016). URL: http://eprint.iacr.org/2016/086.

[Cor99] Jean-Sébastien Coron. "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems". In: *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. Lecture Notes in Computer Science. Springer, 1999, pp. 292–302. DOI: 10.1007/3-540-48059-5_25. URL: https://doi.org/10.1007/3-540-48059-5_25.

[Dan+13] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. "A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards". In: *J. Cryptographic Engineering* 3.4 (2013), pp. 241–265. DOI: 10.1007/s13389-013-0062-6. URL: https://doi.org/10.1007/s13389-013-0062-6.

[Fan+10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. "State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures". In: *HOST 2010, Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 13-14 June 2010, Anaheim Convention Center, California, USA*. Ed. by Jim Plusquellic and Ken Mai. IEEE Computer Society, 2010, pp. 76–87. DOI: 10.1109/HST.2010.5513110. URL: https://doi.org/10.1109/HST.2010.5513110.

[Fip] *Digital Signature Standard (DSS)*. FIPS PUB 186-4. National Institute of Standards and Technology, 2013. DOI: 10.6028/NIST.FIPS.186-4. URL: https://doi.org/10.6028/NIST.FIPS.186-4.

[FV12] Junfeng Fan and Ingrid Verbauwhede. "An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost". In: *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*. Ed. by David Naccache. Vol. 6805. Lecture Notes in Computer Science. Springer, 2012, pp. 265–282. DOI: 10.1007/978-3-642-28368-0_18. URL: https://doi.org/10.1007/978-3-642-28368-0_18.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. "Public-Key Cryptosystems from Lattice Reduction Problems". In: *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. Ed. by Burton S. Kaliski Jr. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 112–131. DOI: 10.1007/BFb0052231. URL: https://doi.org/10.1007/BFb0052231.

[GK15] Shay Gueron and Vlad Krasnov. "Fast prime field elliptic-curve cryptography with 256-bit primes". In: *J. Cryptographic Engineering* 5.2 (2015), pp. 141–151. DOI: 10.1007/s13389-014-0090-x. URL: https://doi.org/10.1007/s13389-014-0090-x.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. "Lattice Enumeration Using Extreme Pruning". In: *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 257–278. DOI: 10.1007/978-3-642-13190-5_13. URL: https://doi.org/10.1007/978-3-642-13190-5_13.

[HMV04] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Professional Computing. Springer-Verlag, New York, 2004, pp. xx+311. ISBN: 0-387-95273-X. DOI: 10.1016/s0012-365x(04)00102-5. URL: https://doi.org/10.1016/s0012-365x(04)00102-5.

[HS01] Nick Howgrave-Graham and Nigel P. Smart. "Lattice Attacks on Digital Signature Schemes". In: *Des. Codes Cryptogr.* 23.3 (2001), pp. 283–290. DOI: 10.1023/A:1011214926272. URL: https://doi.org/10.1023/A:1011214926272.

[JY02] Marc Joye and Sung-Ming Yen. "The Montgomery Powering Ladder". In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 291–302. DOI: 10.1007/3-540-36400-5_22. URL: https://doi.org/10.1007/3-540-36400-5_22.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25. URL: https://doi.org/10.1007/3-540-48405-1_25.

[Knu98] Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms*. 3rd ed. Addison-Wesley, 1998. ISBN: 0201896842. URL: http://www.worldcat.org/oclc/312898417.

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9. URL: https://doi.org/10.1007/3-540-68697-5_9.

[Mai+13] Diana Maimut, Cédric Murdica, David Naccache, and Mehdi Tibouchi. "Fault Attacks on Projective-to-Affine Coordinates Conversion". In: *Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers*. Ed. by Emmanuel Prouff. Vol. 7864. Lecture Notes in Computer Science. Springer, 2013, pp. 46–61. DOI: 10.1007/978-3-642-40026-1_4. URL: https://doi.org/10.1007/978-3-642-40026-1_4.

[MIE17] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. "CacheZoom: How SGX Amplifies the Power of Cache Attacks". In: *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 69–90. DOI: 10.1007/978-3-319-66787-4_4. URL: https://doi.org/10.1007/978-3-319-66787-4_4.

[NS02] Phong Q. Nguyen and Igor E. Shparlinski. "The Insecurity of the Digital Signature Algorithm with Partially Known Nonces". In: *J. Cryptology* 15.3 (2002), pp. 151–176. DOI: 10.1007/s00145-002-0021-3. URL: https://doi.org/10.1007/s00145-002-0021-3.

[NS03] Phong Q. Nguyen and Igor E. Shparlinski. "The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces". In: *Des. Codes Cryptogr.* 30.2 (2003), pp. 201–217. DOI: 10.1023/A:1025436905711. URL: https://doi.org/10.1023/A:1025436905711.

[NSS04] David Naccache, Nigel P. Smart, and Jacques Stern. "Projective Coordinates Leak". In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 257–267. DOI: 10.1007/978-3-540-24676-3_16. URL: https://doi.org/10.1007/978-3-540-24676-3_16.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES". In: *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*. Ed. by David Pointcheval. Vol. 3860. Lecture Notes in Computer Science. Springer, 2006, pp. 1–20. DOI: 10.1007/11605805_1. URL: https://doi.org/10.1007/11605805_1.

[Per05] Colin Percival. "Cache Missing for Fun and Profit". In: *BSDCan 2005, Ottawa, Canada, May 13-14, 2005, Proceedings*. 2005. URL: http://www.daemonology.net/papers/cachemissing.pdf.

[PGB17] Cesar Pereida García and Billy Bob Brumley. "Constant-Time Callees with Variable-Time Callers". In: *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. Ed. by Engin Kirda and Thomas Ristenpart. USENIX Association, 2017, pp. 83–98. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/garcia.

[Rya19] Keegan Ryan. "Return of the Hidden Number Problem: A Widespread and Novel Key Extraction Attack on ECDSA and DSA". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.1 (2019), pp. 146–168. DOI: 10.13154/tches.v2019.i1.146-168. URL: https://doi.org/10.13154/tches.v2019.i1.146-168.

[Shi+16] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. "Preventing Page Faults from Telling Your Secrets". In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*. Ed. by Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang. ACM, 2016, pp. 317–328. DOI: 10.1145/2897845.2897885. URL: https://doi.org/10.1145/2897845.2897885.

[Sil19] Fábio Silva. *mbedtls-compat-sgx: mbedTLS Intel(r) SGX Compatibility Layer*. https://github.com/ffosilva/mbedtls-compat-sgx/tree/c66e974. 2019.

[Sto06] Marc Stoecklin. *Projective Coordinates Leak*. Semester Project. École Polytechnique Fédérale de Lausanne, 2006, pp. 1–84. URL: http://www.stoecklin.net/academics/projects/projectivecoordinatesleak/page_data/stoecklin_marc_WS2006_projective_coordinates_leak_REPORT.pdf.

[Tav+11] Jonathan Taverne, Armando Faz-Hernández, Diego F. Aranha, Francisco Rodríguez-Henríquez, Darrel Hankerson, and Julio López. "Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction". In: *J. Cryptographic Engineering* 1.3 (2011), pp. 187–199. DOI: 10.1007/s13389-011-0017-8. URL: https://doi.org/10.1007/s13389-011-0017-8.

[TPV17] Chia-che Tsai, Donald E. Porter, and Mona Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX". In: *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017*. Ed. by Dilma Da Silva and Bryan Ford. USENIX Association, 2017, pp. 645–658. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai.

[TTA18] Akira Takahashi, Mehdi Tibouchi, and Masayuki Abe. "New Bleichenbacher Records: Fault Attacks on qDSA Signatures". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 331–371. DOI: 10.13154/tches.v2018.i3.331-371. URL: https://doi.org/10.13154/tches.v2018.i3.331-371.

[Tuv+18] Nicola Tuveri, Sohaib ul Hassan, Cesar Pereida García, and Billy Bob Brumley. "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*. ACM, 2018, pp. 147–160. DOI: 10.1145/3274694.3274725. URL: https://doi.org/10.1145/3274694.3274725.

[VB+17] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution". In: *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. Ed. by Engin Kirda and Thomas Ristenpart. USENIX Association, 2017, pp. 1041–1056. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck.

[VBPS17] Jo Van Bulck, Frank Piessens, and Raoul Strackx. "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control". In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*. ACM, 2017, pp. 1–6. DOI: 10.1145/3152701.3152706. URL: https://doi.org/10.1145/3152701.3152706.

[Wan+17] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM, 2017, pp. 2421–2434. DOI: 10.1145/3133956.3134038. URL: https://doi.org/10.1145/3133956.3134038.

[WSB18] Samuel Weiser, Raphael Spreitzer, and Lukas Bodner. "Single Trace Attack Against RSA Key Generation in Intel SGX SSL". In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*. Ed. by Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim. ACM, 2018, pp. 575–586. DOI: 10.1145/3196494.3196524. URL: http://doi.acm.org/10.1145/3196494.3196524.

[XCP15] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems". In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 640–656. DOI: 10.1109/SP.2015.45. URL: https://doi.org/10.1109/SP.2015.45.

[YF14] Yuval Yarom and Katrina Falkner. "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack". In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*. USENIX Association, 2014, pp. 719–732. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom.

[Int19a] Intel. *Intel Software Guard Extensions SSL*. https://github.com/intel/intel-sgx-ssl. 2019.

[Int19b] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*. 2019.