A Tale of Three Signatures: practical attack of ECDSA with wNAF

Gabrielle De Micheli[1], Rémi Piau[1,2], and Cécile Pierrot[1]

[1]Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
[2]ENS Rennes, France

Abstract

Attacking ECDSA with wNAF implementation for the scalar multiplication first requires some side channel analysis to collect information, then lattice based methods to recover the secret key. In this paper, we reinvestigate the construction of the lattice used in one of these methods, the *Extended Hidden Number Problem* (EHNP). We find the secret key with only 3 signatures, whereas best previous methods required 4 signatures at least in practice. Our attack is faster than previous attacks, in particular compared to times reported in [FWC16] and for most cases, has better probability of success. To obtain such results, we perform a detailed analysis of the parameters used in the attack and introduce a preprocessing method which reduces by a factor up to 7 the total time to recover the secret key for some parameters. We perform an error resilience analysis which has never been done before in the setup of EHNP. Our construction is still able to find the secret key with a small amount of erroneous traces, up to 2% of false digits, and 4% with a specific type of error. We also investigate Coppersmith's methods as a potential alternative to EHNP and explain why, to the best of our knowledge, EHNP goes beyond the limitations of Coppersmith's methods.

1 Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) [JMV01] is a standard public key signature protocol widely deployed. It benefits from a high security based on the hardness of the elliptic curve discrete logarithm problem and a fast signing algorithm due to its small key size. ECDSA is used in the latest TLS 1.3. It is implemented in OpenSSL, and can also be found in cryptocurrencies such as Bitcoin [Nak09], Ethereum [B$^+$] and Ripple [SYB14].

The ECDSA signing algorithm requires the computation of some scalar multiplication of a point P on an elliptic curve by an ephemeral key k. Since this operation is time-consuming and often the most time-consuming part of the protocol, it is necessary to use an algorithm that is efficient but remains secure in the context of side-channel analysis. The Non Adjacent Form (NAF) and its windowed variant (wNAF) were introduced as an alternative to the binary representation of the nonce k to reduce the execution time of the scalar multiplication. Indeed, the NAF representation does not allow two non-zero digits to be consecutive, thus reducing the Hamming weight of the representation of the scalar. This improves on the time of execution as the latter is dependent on the number of zero digits. Until November 2018, OpenSSL used the wNAF representation in all its versions. However, after the OpenSSL ECC scalar multiplication have been reported to be vulnerable to microarchitectural timing side-channel attack (CVE-2018-5407), OpenSSL moved away from this representation (versions 1.1.0i, 1.0.2q and more recently 1.1.1c in May 2019). However, the wNAF representation remains present in Bitcoin for example, as well as in the library Cryptlib.

Microarchitectural side-channel attacks allow to recover secret information otherwise hidden from the public throughout some observable leakage. They were first introduced about two decades ago

by Kocher et al [KJJ99], and have since been used to break many implementations, and in particular some cryptographic primitives such as AES [ASK06], RSA [AS08], and ECDSA [JMV01]. In particular cache side-channel attacks such as FLUSH+RELOAD [YB14, YF14] have been used to recover information about the sequence of operations used to execute the scalar multiplication in ECDSA. This information is usually referred to as a double-and-add chain or the trace of k. A trace is created when a signature is produced by ECDSA and thus we talk about signatures and traces in an equivalent sense. The nature of the information obtained from the side channel attack allows to determine what kind of attack can be carried out to recover the secret key. Howgrave-Graham and Smart [HGS01] showed how knowing partial information of the nonce k in DSA can lead to a full secret key recovery. Nguyen and Shparlinski [NS03] gave a polynomial time algorithm that recovers the secret key in ECDSA as soon as some consecutive bits of the ephemeral key are known. They showed that using the information leaked by the side channel attack, one can recover the secret key by constructing an instance of the Hidden Number Problem (HNP) [BV96]. This type of attack has been done in [BvdPSY14, vdPSY15, WF17]. Moreover, in [WF17] the authors propose three rules to select traces that improve the attack on the lattice part. In [FWC16], the authors extract even more information from the double-and-add chains and use the Extended Hidden Number Problem (EHNP) to recover the secret key using only 4 error-free traces.

Contribution: In this work, we reinvestigate the attack against ECDSA with wNAF representation for the scalar multiplication using EHNP. We work with the elliptic curve `secp256k1` but none of the techniques introduced in this paper are limited to this specific elliptic curve. We give an explicit lattice construction based on the construction given in [FWC16]. Several parameters occur while building and reducing the lattice. We analyze the performance of the attack with respect to these parameters and present the best parameters that optimize our attack. There are several metrics to compare the results: either we minimize the number of signatures required for the attack to work, or we fix the number of signatures and optimize either the total time to recover the key, or the probability of success. In this paper, when talking about the total time of our attack, we consider the average time of a single experiment multiplied by the number of trials necessary to recover the secret key.[1]

We introduce a new preprocessing method on the traces. The idea of selecting good traces beforhand has already been explored in [WF17]. Given a certain (large) amount of traces available, the lattice is usually built with a much smaller subset of these traces. Trying to identify beforehand the traces that would result in a better attack is a clever option. The aim of our new preprocessing - that completely differs from [WF17] - is to regulate the size of the coefficients in the lattice, and this results in a better lattice reduction time. For instance, with 3 signatures, we were able to reduce the total time of the attack by a factor up to 7.

Overall, combining new methods with a careful choice of parameters allows us to mount an attack which works in practice with only 3 signatures. However, the probability of success in this case is very low. We were able to recover the secret key only once with BKZ-35 over 5000 experiments. If we assume the probability is around 0.02%, as each trial costs 200 seconds in average, this means we can expect to find the secret key after 12 days. On the other hand, if we use our preprocessing method, with $u = 3$ we obtain a probability of success of 0.2% and a total time of key recovery of 39 hours, thus the factor 7 of improvement mentioned above. Despite the low probability of success, this result remains interesting nonetheless. Indeed, when using the FLUSH+RELOAD attack, the authors in [FWC16] mention that on average 105.8 bits per signature for 256-bit ECDSA can be obtained. Hence, they state that in theory the secret key can be recovered using 3 signatures. However, in practice, the key had never been recovered using less than 4 signatures.

[1] We compare our times with the numbers reported in [FWC16], Table 3 for method C. Indeed, methods A and B require extra information on the implementation which we choose not to consider as we want our analysis to remain as general as possible. We believe that to have an accurate comparison, their times have to be multiplied by the number of trials necessary for their attack to recover the secret key, thus increasing their total time a lot. For example, using 5 signatures, their best total time is around 15 hours instead of 18 minutes.

Number of signatures	Our attack		[FWC16]	
u	Time	Probability of success (%)	Time	Probability of success (%)
4	1 hour 17 minutes	5%	41 minutes	1.5%
5	8 minutes 20 seconds	6.5%	18 minutes	1%
6	≈ 5 minutes	25%	18 minutes	22%
7	≈ 3 minutes	17.5%	34 minutes	24%

Table 1: Comparing attack time and success probability with [FWC16].

When using 4 signatures, our attack is slightly slower than the attack in [FWC16] where they use another method, named A, which assumes having some extra information. Since they do not provide times without method A, we compare our times with theirs using A. We suspect that their time for this case without A is at least the time they report using A. When considering more than 4 signatures, our attack is faster than the times reported in [FWC16]. In Table 1, we compare our times with the fastest times reported by [FWC16]. We choose their fastest times but concerning our parameters we choose to report experiments which are faster (not the fastest) with, if possible, better probability too.

We experimented up to 8 signatures to further improve our overall time. In this case, our attack runs at best in 2 minutes and 25 seconds.

One can see that [FWC16] remains competitive while considering the probability of success of one experiment for 7 signatures. Indeed, they have 68% of success with their best parameters whereas we only reach 45% for this setting.

Moreover, we investigate the resilience to errors of our attack. Such an analysis has not yet been done in the setup of EHNP. It is important to underline that collecting traces without any errors using any side-channel attack is very hard. Previous works used perfect traces to mount the lattice attack. Thus, it requires collecting more traces. As pointed out in [FWC16], more or less twice as many signatures are needed if errors are considered. In practice, this led [FWC16] to gather in average 8 signatures to be able to find the key with 4 perfect traces. We experimentally show that we are still able to recover the secret key even in the presence of faulty traces. In particular, we find the key using only 4 faulty traces, but with a very low probability of success. As the percentage of incorrect digits in the trace grows, the probability of success decreases and thus more signatures are required to successfully recover the secret key. For instance, if 2% of the digits are wrong among all the digits of a given set of traces, it is still possible to recover the key with 6 signatures. This result is valid if errors are uniformly distributed over the digits. However, we have a better probability to recover the key if errors consist in 0-digit faulty reads, i.e., 0 digits read as 1. In other words, the attack could work with a higher percentage of errors, around 4%, if we could ensure from the side channel attack and some preprocessing methods that none of the 1 digits have been flipped to 0.

Finally, as the EHNP setup consists of a system of modular equations for which we look for integer roots, we investigate the use of Coppersmith's methods for finding small roots of integer polynomials. However, our attempts to apply Coppersmith's methods were not successful as some bound on the unknowns is not satisfied.

Organization: In Section 2, we introduce some background on ECDSA and the wNAF representation. Moreover, we give some necessary details on lattices and well known reduction algorithms such as LLL and BKZ. In Section 3, we explain how the Extended Hidden Number problem can be transformed into a lattice problem. Finding the secret key then consists in solving the shortest vector problem in a given lattice. We explicit the lattice basis and give some analysis on the length of the short vectors found in the reduced basis. In Section 4, we present various improvements on the attack. We first explain a merging method to reduce the lattice dimension given in [FWC16]. We then introduce a new preprocessing method on the traces which allows us to reduce the overall time of our attack. In Section 5, we give experimental results which shows the performance of our attack as a

function of the various parameters that are being considered. In Section 6, we analyze the resilience of our attack to erroneous traces. Finally in Section 7, we describe an attempt to a new approach to EHNP using Coppersmith's methods for finding small roots of modular equations.

2 Preliminaries

2.1 Elliptic Curves Digital Signature Algorithm

The Elliptic Curves Digital Signature Algorithm is a variant of the Digital Signature Algorithm, DSA, [NIS13] which uses elliptic curves instead of finite fields. The parameters used in the ECDSA algorithm are an elliptic curve E over a finite field, a generator G of prime order q and a hash function H. The private key is an integer α such that $1 < \alpha < q-1$ and the public key is $p_k = [\alpha]G$, the scalar multiplication of G by α.

To sign a message m using the private key α, randomly select an ephemeral key $k \leftarrow_R \mathbb{Z}_q$ and compute $[k]G$. Let r be the x-coordinate of $[k]G$. If $r = 0$, select a new nonce k. Then, compute $s = k^{-1}(H(m) + \alpha r) \bmod q$ and again if $s = 0$, select a new nonce k. Finally, the signature is given by the pair (r, s).

In order to verify a signature, first check if $r, s \in \mathbb{Z}_q$, otherwise the signature is not valid. Then, compute $v_1 = H(m) \cdot s^{-1} \bmod q$, $v_2 = r \cdot s^{-1} \bmod q$ and $(x, y) = [v_1]G + [v_2]p_k$. Finally, the signature is valid if $x \equiv r \pmod q$.

Remark 1. *In this paper, we consider a 128 bit level of security and thus α, q and k are all 256-bit integers.*

2.2 wNAF representation

The ECDSA algorithm presented above requires the computation of $[k]G$ which corresponds to a scalar multiplication. In [Gor98], various methods to compute fast exponentiation are presented. One family of such methods is called window methods and comes from NAF representation. Indeed, the NAF representation does not allow two non-zero digits to be consecutive, thus reducing the Hamming weight of the representation of the scalar. The basic idea of a window method is to consider chunks of w bits in the representation of the scalar k, compute powers in the window bit by bit, square w times and then multiply by the power in the next window. The window methods can be combined with the NAF representation of k, as we will explain now. For any $k \in \mathbb{Z}$, a representation

$$k = \sum_{j=0}^{\infty} k_j 2^j$$

is called a NAF if $k_j \in \{0, \pm 1\}$ and $k_j k_{j+1} = 0$ for all $j \geq 0$. Moreover, every k has a unique NAF representation. The NAF representation minimizes the number of non-zero digits k_j. It is presented in Algorithm 1.

The NAF representation can be combined with a sliding window method to further improve the execution time. For instance, in OpenSSL (up to the latest versions using wNAF 1.1.1b for example), the window size usually chosen was $w = 3$. The scalar k is converted into wNAF form using Algorithm 2. Note that in Algorithm 2, the sequence of digits m_i belongs to the set $\{0, \pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$. We can rewrite k as a sum of its non-zero digits, which we rename k_i. More precisely, we get

$$k = \sum_{j=1}^{\ell} k_j 2^{\lambda_j},$$

Input : $k \in \mathbb{Z}^+$
Output: NAF representation of k
$i = 0$;
while $k > 0$ **do**
  if $k \pmod 2 = 1$ **then**
    $k_i = 2 - (k \pmod 4)$;
    $k = k - k_i$;
  else
    $k_i = 0$;
  end
  $k = k/2$;
  $i = i + 1$;
end
return $k_{i-1}, k_{i-2}, \ldots, k_1, k_0$

Algorithm 1: NAF algorithm

Input: $k \in \mathbb{Z}^+$, $w \in \mathbb{N}$
Output: $(m_0, m_1, \ldots, m_n)$, i.e., k in its wNAF representation
$i = 0$;
while $k > 0$ **do**
  if $k \pmod 2 = 1$ **then**
    $m_i = k \pmod{2^{w+1}}$;
    if $m_i \geq 2^w$ **then**
      $m_i = m_i - 2^{w+1}$;
    end
    $k = k - m_i$;
  else
    $m_i = 0$;
  end
  $k = k/2$;
  $i = i + 1$;
end

Algorithm 2: wNAF representation

where ℓ is the number of non-zero digits, and λ_j represents the position of the digit k_j in the wNAF representation.

Example 1. *In binary, we can write*

$$23 = 2^4 + 2^2 + 2^1 + 2^0 = (1, 0, 1, 1, 1)$$

whereas in NAF-representation, we have

$$23 = 2^5 - 2^3 - 2^0 = (1, 0, -1, 0, 0, -1).$$

Using a window size $w = 3$, the wNAF representation gives

$$23 = 2^4 + 7 \times 2^0 = (1, 0, 0, 0, 7).$$

There exists a modified wNAF representation used in OpenSSL for example. In the non-modified wNAF representation, at most one of any $w + 1$ consecutive digits is non-zero and in the modified version, this also stands with the exception that the most significant digit may be only $w - 1$ zeros away from that next non-zero digit.

2.3 Lattice reduction algorithms

A lattice is a discrete additive subgroup of $\mathbb{R}^n$. It is usually specified by giving a *basis matrix* $B \in \mathbb{Z}^{n \times n}$. The lattice $L(B)$ generated by B consists of all integer combinations of the row vectors in B. The determinant of a lattice is the absolute value of the determinant of a basis matrix: $\det L(B) = |\det B|$. The discreteness property ensures that there is some $\lambda_1 > 0$ such that the length of one of the shortest non-zero vectors v_1 in the lattice satisfies $||v_1|| = \lambda_1$. Let λ_i be the i^{th} successive minimum of the lattice. The LLL algorithm [LLL82] takes as an input a lattice basis, and returns in polynomial time in the lattice dimension n a reduced lattice basis whose vectors b_i satisfy the worst-case approximation bound $||b_i||_2 \leq 2^{(n-1)/2}\lambda_i$. In practice, for random lattices, LLL obtains approximation factors such that $b_1 \leq 1.02^n \lambda_1$ as noted by Nguyen and Stehlé [NS06]. Moreover, for random lattices, we note that the Gaussian heuristic implies that

$$\lambda_1 \approx \sqrt{n/(2\pi e)}\det(L)^{1/n}. \tag{1}$$

The BKZ algorithm [Sch87, SE94] is exponential in some given block-size β and polynomial in the lattice dimension n. It outputs a reduced lattice basis whose vectors b_i satisfy the approximation $||b_i||_2 \leq i\gamma_\beta^{(n-i)/(k-1)}\lambda_i$ [Sch94], where γ_β is the Hermite constant. In practice, Chen and Nguyen [CN11] observed that BKZ returns vectors such that $b_1 \leq (1 + \epsilon_\beta)^n \lambda_1$ where ϵ_β depends on the block-size β. For random lattices, they get $1 + \epsilon_\beta = 1.01$ for a block-size $\beta = 85$.

3 Attacking ECDSA using lattices

Using some side-channel attack, one can recover information about the wNAF representation of the nonce k. In particular, it allows us to know the positions of the non-zero coefficients in the representation of k. However, the value of these coefficients are unknown. This information can be used in the setup of the Extended Hidden Number Problem (EHNP) to recover the secret key. For many messages m, we use ECDSA to produce signatures (r, s) and each run of the signing algorithm produces a nonce k. We assume we have the corresponding trace of the nonce, that is, the equivalent of the double-and-add chain of kG using wNAF. The goal of the attack is to recover the secret α while optimizing either the number of signatures required or the total time of the attack.

3.1 The Extended Hidden Number Problem

The Hidden Number Problem (HNP), introduced in 1996 [BV96], allows to recover a secret element $\alpha \in \mathbb{Z}_q$ if some information about the most significant bits of random multiples of $\alpha \pmod q$ are known for some prime q. Boneh and Venkatesan show how to recover α in polynomial time with probability greater than 1/2. In [HR07], the authors extend the HNP and present a polynomial time algorithm for solving the instances of this extended problem. The Extended Hidden Number Problem is defined as follows. Given u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod q, \tag{2}$$

where the secret α and $0 \leqslant k_{i,j} \leqslant 2^{\eta_{ij}}$ are unknown, and the values η_{ij}, a_i, $b_{i,j}$, c_i, ℓ_i are all known for $1 \leqslant i \leqslant u$ (see [HR07], Definition 3), one has to recover α in polynomial time. Similarly to the HNP, the EHNP can be transformed into a lattice problem and one can recover the secret α by solving a short vector problem in a given lattice.

3.2 Using EHNP to attack ECDSA

From the ECDSA algorithm, we know that given a message m, the algorithm outputs a signature (r, s) such that

$$\alpha r = sk - H(m) \pmod q. \tag{3}$$

The value $H(m)$ is just some hash of the message m. We consider a set of u signature pairs (r_i, s_i) with corresponding message m_i that satisfies Equation (3). For each signature pair, we have a nonce k. Using the wNAF representation of k, we write $k = \sum_{j=1}^{\ell} k_j 2^{\lambda_j}$, with $k_j \in \{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$ and the choice of w depends on the implementation. Note that the coefficients k_j are unknown, however, the positions λ_j are supposed to be known via some side-channel leakage. It is then possible to represent the ephemeral key k as the sum of a known part, and an unknown part. As the value of k_j is odd, one can write $k_j = 2k'_j + 1$, where $-2^{w-1} \leqslant k'_j \leqslant 2^{w-1} - 1$. Using the same notations as in [FWC16], set $d_j = k'_j + 2^{w-1}$, where $0 \leq d_j \leq 2^w - 1$. In the rest of the paper, we will denote by μ_j the window-size of d_j. Note that here, $\mu_j = w$ but this window-size will be modified later. This allows to rewrite the value of k as

$$k = \sum_{j=1}^{\ell} k_j 2^{\lambda_j} = \bar{k} + \sum_{j=1}^{\ell} d_j 2^{\lambda_j + 1}, \tag{4}$$

with $\bar{k} = \sum_{j=1}^{\ell} 2^{\lambda_j} - \sum_{j=1}^{\ell} 2^{\lambda_j + w}$. The expression of $\bar{k}$ represents the known part of k.

By substituting k in Equation (4), we obtain a system of modular equations of the form

$$\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k_i} - H(m_i)) \equiv 0 \pmod q \tag{5}$$

where the unknowns are α and the $d_{i,j}$. The known values are the ℓ_i which is the number of non-zero digits in k for the i^{th} sample, $\lambda_{i,j}$, which is the position of the j^{th} non-zero digit in k for the i^{th} sample and $\bar{k}$ defined above. We can then use Equation (5) as input to the Extended Hidden Number Problem, following the method explained in [HR07]. The problem of finding the secret key is then reduced to solving the short vector problem in a given lattice which we give in the following section.

3.3 Constructing the lattice

Before giving the lattice basis construction, we redefine Equation (5) to reduce the number of unknown variables in the system. This will allow us to construct a lattice of smaller dimension. Again, we use the same notations as in [FWC16].

Eliminating one variable. One straightforward way to reduce the lattice dimension is to eliminate a variable from the system. In this case, one can eliminate α from Equation (5). Let E_i denote the i^{th} equation of the system. Then by computing $r_1 E_i - r_i E_1$, we get the following new modular equations

$$\sum_{j=1}^{\ell_1} \underbrace{(2^{\lambda_{1,j}+1} s_1 r_i)}_{:=\tau_{j,i}} d_{1,j} + \sum_{j=1}^{\ell_i} \underbrace{(-2^{\lambda_{i,j}+1} s_i r_1)}_{:=\sigma_{i,j}} d_{i,j} - \underbrace{r_1(s_i \bar{k}_i - H(m_i)) + r_i(s_1 \bar{k}_1 - H(m_1))}_{:=\gamma_i} \equiv 0 \pmod{q}. \tag{6}$$

Again, using the same notations as in [FWC16], we define $\tau_{j,i} = 2^{\lambda_{1,j}+1} s_1 r_i$, $\sigma_{i,j} = -2^{\lambda_{i,j}+1} s_i r_1$ and $\gamma_i = r_1(s_i \bar{k}_i - H(m_i)) + r_i(s_1 \bar{k}_1 - H(m_1))$ for $2 \leqslant i \leqslant u$, $1 \leqslant j \leqslant \ell_i$. Even if α is eliminated from the equations, if we are able to recover some $d_{i,j}$ values from a short vector in the lattice, we can recover α using any equation in the modular system (5). We will now use Equation (6) to construct the lattice basis.

From a modular system to a lattice basis. Let $\mathcal{L}$ be the lattice constructed for the attack, and we have $\mathcal{L} = \mathcal{L}(\mathcal{B})$ where the lattice basis $\mathcal{B}$ is given below. Let $m = \max_{i,j} \mu_{ij}$ for $1 \leqslant j \leqslant \ell_i$ and $2 \leqslant i \leqslant u$. We set a scaling factor $\Delta \in \mathbb{N}$ to be defined later. The lattice basis is given by

$$\mathcal{B} = \begin{pmatrix}
Eq\ (6), i=2 & \dots & Eq\ (6), i=u & & & & & & & & & & & & \\
\Delta 2^m q & 0 & 0 & 0 & & & & & & & & & & & \\
0 & \ddots & \vdots & & & & & & & & & & & & \\
0 & \cdots & \Delta 2^m q & 0 & & & & & & & & & & & \\
\Delta 2^m \tau_{1,2} & \dots & \Delta 2^m \tau_{1,u} & 2^{m-\mu_{1,1}} & & & & & & & & & & & \\
\vdots & & \vdots & 0 & \ddots & & & & & & & & & & \\
\Delta 2^m \tau_{\ell_1,2} & \dots & \Delta 2^m \tau_{\ell_1,u} & & & 2^{m-\mu_{1,\ell_1}} & & & & & & & & & \\
\Delta 2^m \sigma_{2,1} & 0 & 0 & & & & 2^{m-\mu_{2,1}} & & & & & & & & \\
\vdots & & \vdots & & & & & \ddots & & & & & & & \\
\Delta 2^m \sigma_{2,\ell_2} & & \vdots & & & & & & 2^{m-\mu_{2,\ell_2}} & & & & & & \\
0 & \ddots & 0 & \vdots & & & & & & \ddots & & & & & \\
\vdots & & \Delta 2^m \sigma_{u,1} & & & & & & & & 2^{m-\mu_{u,1}} & & & & \\
\vdots & & \vdots & & & & & & & & & \ddots & & & \\
0 & 0 & \Delta 2^m \sigma_{u,\ell_u} & 0 & & & & & & & & & 2^{m-\mu_{u,\ell_u}} & & \\
\Delta 2^m \gamma_2 & \dots & \Delta 2^m \gamma_u & 2^{m-1} & \dots & & & & & & & & 2^{m-1} & 2^{m-1}
\end{pmatrix}$$

Let $n = (u-1) + T + 1 = T + u$, with $T = \sum_{i=1}^{u} \ell_i$, be the dimension of the lattice. The $u-1$ first columns correspond to Equation (6) for $2 \leq i \leq u$. Each of the remaining columns, except the last one, correspond to a d_{ij}, and contains coefficients that allow to regulate the size of the d_{ij}. The determinant of $\mathcal{L}$ is given by

$$\det \mathcal{L} = q^{u-1} (\Delta 2^m)^{u-1} 2^{\sum_{i,j}(m-\mu_{i,j})} 2^{m-1}.$$

The lattice is built such that there exists $w \in \mathcal{L}$ which contains the unkowns $d_{i,j}$. To find it, we know there exists some values $t_2, t_2, \dots, t_u$ such that if $v = (t_2, \dots, t_u, d_{1,1}, \dots, d_{u,\ell_u}, -1)$, we get

$$w = v\mathcal{B}, \tag{7}$$

and

$$w = (0, \dots, 0, d_{1,1} 2^{m-\mu_{1,1}} - 2^{m-1}, \dots, d_{u,\ell_u} 2^{m-\mu_{u,\ell_u}} - 2^{m-1}, -2^{m-1}).$$

If we are able to find w in the lattice, then we can reconstruct the secret key α. In order to find w, we estimate its norm and make sure w appears in the reduced basis. After reducing the basis, we look for vectors of the correct shape, i.e., with sufficiently enough zeros at the beginning and the correct last coefficient, and attempt to recover α for each of these.

How the size of Δ affects the norms of the short vectors. In order to find the vector w in the lattice, we reduce $\mathcal{B}$ using LLL or BKZ. For w to appear in the reduced basis, one should at least set Δ such that

$$||w||_2 \leqslant (1.02)^n (\det L)^{1/n}. \tag{8}$$

The vector w we expect to find has norm $||w||_2 \leqslant 2^{m-1}\sqrt{T+1}$. From Equation (8), one can deduce the value of Δ needed to find w in the reduced lattice, which is given by the expression

$$\Delta \geqslant \frac{(T+1)^{(T+u)/(2(u-1))} 2^{\frac{1+\sum \mu_{i,j}-(u+T)}{u-1}}}{q(1.02)^{\frac{(T+u)^2}{u-1}}} := \Delta_{th}$$

In our experiments, the average value of ℓ_i for $1 \leqslant i \leqslant u$ is $\tilde{\ell} = 26$, and thus $T = u \times \tilde{\ell}$ on average. Moreover, the average value of μ_{ij} is 7 and so on average $\sum \mu_{ij} = 7 \times u \times \tilde{\ell}$. Hence, if we compute Δ_{th} for $u = 3, \ldots, 8$, with these values, we obtain $\Delta_{th} \ll 1$, which does not help us to set this parameter.

In practice, we verify that setting $\Delta = 1$ allows us to recover the secret key. In our experiments, we vary the bitsize of Δ to see whether a slightly larger value affects the probability of success. This comment will be adressed in Section 5.

Too many small vectors. While running BKZ on $\mathcal{B}$, we note that for some specific sets of parameters the reduced basis contains some undesired short vectors, i.e., vectors that are shorter than w. This can be explained by looking at two consecutive rows in the lattice basis given above, say the j^{th} row and the $(j+1)^{th}$ row. For example, one can look at rows which correspond to the $\sigma_{i,j}$ values but the same argument is valid for the rows concerning the $\tau_{j,i}$. From the definitions of the σ values we have

$$\begin{aligned} \sigma_{i,j+1} &= -2^{\lambda_{i,j+1}+1} \cdot s_i r_1 \\ &= -2^{\lambda_{i,j+1}+1} \cdot (\tfrac{\sigma_{i,j}}{-2^{\lambda_{i,j+1}+1}}) \\ &= 2^{\lambda_{i,j+1}-\lambda_{i,j}} \cdot \sigma_{i,j} \end{aligned}$$

Thus the linear combination given by the $(j+1)^{th}$ row minus $2^{\lambda_{i,j+1}-\lambda i,j}$ times the j^{th} row gives a vector

$$(0\,, \cdots, 0\,, -2^{\lambda_{i,j+1}-\lambda_{i,j}+m-\mu_{i,j}}\,, 2^{m-\mu_{i,j+1}}\,, 0\,, \cdots, 0). \tag{9}$$

Yet, this vector is expected to have smaller norm than w. Some experimental observations are detailed in Section 5.

Remark 2. *It would be of interest to understand how one can modify the lattice construction to always find w as the shortest vector of the reduced basis. Indeed, by reducing the number of vectors shorter than w we expect to increase the probability of success of our attack. This would lower the chances of w being a linear combination of short vectors and thus not appearing in the reduced basis.*

Differences with the lattice construction given in [FWC16]. Let $\mathcal{B}'$ be the lattice basis constructed in [FWC16]. Our basis $\mathcal{B}$ is a rescaled version of $\mathcal{B}'$ such that $\mathcal{B} = 2^m \Delta \mathcal{B}'$. This rescaling allows us to ensure that all the coefficients in our lattice basis are integer values. Note that [FWC16] have a value δ in their construction which corresonds to $1/\Delta$. In this work, we give a precise analysis of the value of Δ, both theoretically and experimentally in Section 5, which is missing in [FWC16].

4 Improving the lattice attack

4.1 Reducing the lattice dimension: the merging technique

In [FWC16], the authors present another way to further reduce the lattice dimension, which they call the merging technique. It aims at reducing the lattice dimension by reducing the number of non-zero digits of k. The lattice dimension depends on the value $T = \sum_{i=1}^{u} \ell_i$, and thus reducing T reduces

the dimension. For the understanding of the attack, it suffices to know that after merging, we obtain some new values ℓ' corresponding to the new number of non-zero digits and λ'_j the position of these digits for $1 \leqslant j \leqslant \ell'$. After merging, one can rewrite $k = \bar{k} + \sum_{j=1}^{\ell'} d'_j 2^{\lambda'_j+1}$, where the new d'_j have a new window size which we denote μ_j, i.e., $0 \leqslant d'_j \leqslant 2^{\mu_j} - 1$.

We present here our merging algorithm based on Algorithm 3 given in [FWC16]. Our algorithm modifies directly the sequence $\{\lambda_j\}_{j=1}^{\ell}$, whereas [FWC16] work on the double-and-add chains. This helped us avoid some implementation issues such as an index outrun present in Algorithm 3 [FWC16], line 7. To facilitate the ease of reading of (our) Algorithm 3, we work with dynamic tables. To do so, we first recall various known methods we use in the algorithm: $push\_back(e)$ inserts an element e at the end of the table, $at(i)$ outputs the element at index i, and $last()$ returns the last element of the table. We consider tables of integers indexed in $[0; S-1]$, where S is the size of the table.

Input : v_λ, a table of size n with the positions of non-zero digits in the trace sorted in increasing order and $n \geqslant 1$, a window size w.
Output: $v_{\lambda'}$, a table of size $n' \leqslant n$ containing the merged λ values and table v_μ of same size n', with the values of the window size μ_i.

Initialisation
$i \leftarrow 1$;
$v_{\lambda'} \leftarrow$ empty array;
$v_\mu \leftarrow$ empty array;
Processing
$v_{\lambda'}.push\_back(v_\lambda.at(0))$;
while $i < n$ **do**
  $dist \leftarrow v_\lambda.at(i) - v_\lambda.at(i-1)$;
  if $dist > w + 1$ **then**
    $v_\mu.push\_back(v_\lambda.at(i-1) - v_{\lambda'}.last() + w)$;
    $v_{\lambda'}.push\_back(v_\lambda.at(i))$;
  end
  $i \leftarrow i + 1$;
end
$v_\mu.push\_back(v_\lambda.at(n) - v_{\lambda'}.last() + w)$;
return $(v_{\lambda'}, v_\mu)$

Algorithm 3: Merging algorithm

A useful example of the merging technique is given in [FWC16]. We give in Table 2 the approximate dimension of the lattices we obtain using the elimination and merging techniques. For the traces we consider, after merging the mean of the ℓ_i is 26, the minimum being 17 and the maximum 37 with a standard deviation of 3.

Remark 3. *One could further reduce the lattice dimension by preprocessing traces with small ℓ_i. However, the standard deviation being small, the difference in the reduction times should not be affected too much.*

4.2 Preprocessing the traces

The two main information we can extract and use in our attack are first the number of non-zero digits in the wNAF representation of the nonce k, denoted ℓ and the weight of each non-zero digit, denoted μ_j for $1 \leqslant j \leqslant \ell$. Let $\mathcal{T}$ be the set of traces we obtained from the side-channel leakage representing the wNAF representation of the nonce k used while producing an ECDSA signature. We consider the subset $S_a = \{t \in \mathcal{T} | \max_j \mu_j \leqslant a, 1 \leqslant j \leqslant \ell\}$. We choose to preselect traces in a subset S_a for small values of a. The idea behind this preprocessing is to regulate the size of the coefficients in the lattice.

Number of signatures	Average dimension
3	80
4	110
5	135
6	160
7	190
8	215

Table 2: Average dimensions of the lattices after merging.

Indeed, when selecting u traces for the attack, by upper-bounding $m = max_{i,j}\mu_{i,j}$ for $2 \leqslant i \leqslant u$, we force the coefficients to remain smaller than when taking traces at random.

In practice, we work with a set $\mathcal{T}$ of 2000 traces such that $\min_{t\in\mathcal{T}} \max_j \mu_j = 11$ and $\max_{t\in\mathcal{T}} \max_j \mu_j = 67$. We consider the sets S_{11}, S_{15} and S_{19} in our experiments. In Table 3, we give the proportion of signatures corresponding to the different preprocessing subsets.

preprocessing	Proportion (%)
S_{11}	2
S_{15}	18
S_{19}	44

Table 3: Proportion of preprocessing subsets.

The effect of preprocessing on the total time of the attack is explained in Section 5.

5 Performance analysis

Recall that a trace corresponds to the double-and-add chain of the scalar multiplication kG. To the best of our knowledge, the only information we can recover are the positions of the non-zero digits. We are not able to determine the sign or the value of the digits in the wNAF representation. In [FWC16], the authors exploit the fact that the length of the binary string of k is fixed in some implementations such as OpenSSL, and thus more information can be recovered by comparing this length to the length of the double-and-add chain. In particular, they were able to recover the MSB of k, and in some cases the sign of the second MSB. We do not consider this extra information as we want our analysis to remain as general as possible.

We report some calculations ran on some error-free traces where we evaluate the total time necessary to recover the secret key and the probability of success of the attack. Our experiments have two possible outputs: either we are able to reconstruct the secret key α and thus consider the experiment to be a success, or we were not able to recover the secret key, and hence the experiment failed. In order to compute the success probability of our attack and the average time of one reduction, we run 5000 experiments for some specific sets of parameters using Sage's default BKZ implementation [dt16]. The experiments were ran using the cluster Grid'5000 on a single core of an Intel Xeon Gold 6130 with 192 GB of RAM. We recall that the total time of our attack is the average time of a single reduction multiplied by the number of trials necessary to recover the secret key. For a fixed number of signatures, we can either optimize the total time of the attack or its probability of success. We report numbers in Tables 4, 5.

u	Total time	Parameters	Probability of success
3	39 h	BKZ-35, Pre-preprocessing S_{11}, $\Delta \approx 2^3$	0.2 %
4	1 h 17	BKZ-25, Pre-preprocessing S_{15}, $\Delta \approx 2^3$	5 %
5	8 min 20	BKZ-25, Pre-preprocessing S_{19}, $\Delta \approx 2^3$	6.5 %
6	3 min 55	BKZ-20, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	7 %
7	2 min 43	BKZ-20, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	17.5 %
8	2 min 25	BKZ-20, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	29 %

Table 4: Fastest key recovery with respect to the number of signatures u.

u	Probability of success	Parameters	Total time
3	0.2 %	BKZ-35, Pre-preprocessing S_{11}, $\Delta \approx 2^3$	39 h
4	4 %	BKZ-35, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	25 h 28
5	20 %	BKZ-35, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	2 h 42
6	40 %	BKZ-35, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	1 h 04
7	45 %	BKZ-35, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	2 h 36
8	45 %	BKZ-35, Pre-preprocessing S_{all}, $\Delta \approx 2^3$	5 h 02

Table 5: Highest probability of success with respect to the number of signatures u.

Experimentally, we vary the parameters that are considered in the attack: the bitsize of Δ, the preprocessing subset and the block-size used in BKZ.

Only 3 signatures. Using $\Delta \approx 2^3$ and no preprocessing, we were able to recover the secret key using 3 signatures with BKZ-35 only once and three times with BKZ-40. When using pre-processing S_{11}, BKZ-35 and $\Delta \approx 2^3$, the probability of success went up to 0.2%. Since all the probabilities remain much less than 1% an extensive analysis would have been too much time consuming to do. For this reason, in the rest of this section, the number of signatures only vary between 4 and 8. However, we want to emphasize that it is precisely this detailed analysis on a slightly higher number of signatures that allowed us to understand the impact of the parameters on the performance of the attack and resulted in finding the right ones allowing to mount the attack with 3 signatures.

Varying the bitsize of Δ. In Figure 1, we analyze the total time to recover the secret key as a function of the bitsize of Δ. We fix the block-size of BKZ to 25 and take traces without any preprocessing. We are able to recover the secret key by setting $\Delta = 1$, which is the lowest theoretical value one can choose. However, we observed a slight increase in the probability of success by taking a larger Δ. Without any surprise, we note that the total time to recover the secret key increases with the bitsize of Δ as the coefficients in the lattice basis become larger. Details of the experiments are given in Appendix A.

Analyzing the effect of preprocessing. We also analyze the influence of our preprocessing method on the attack time. We fix BKZ block-size to 25. The effect of preprocessing is influenced by the bitsize

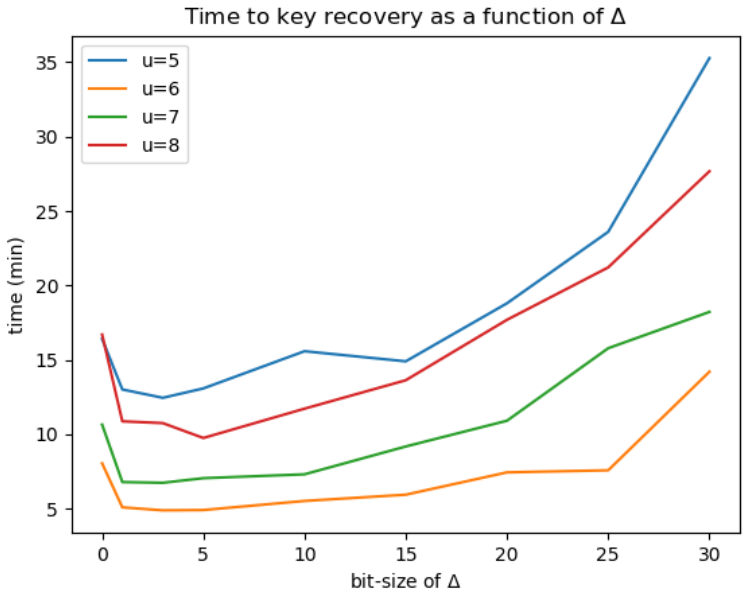

Figure 1: Analyzing the overall time to recover the secret key as a function of the bitsize of Δ. We report the numbers BKZ-25 and no preprocessing. The optimal value for Δ is around 2^3 except for $u = 8$ where it is 2^5.

of Δ and we give here an analyze for $\Delta \approx 2^{25}$ since the effect is more noticeable. We report results for $\Delta \approx 2^3$ in Appendix B. We still gain time using the preprocessing but less than with $\Delta \approx 2^{25}$.

The effect of preprocessing is difficult to predict since its behavior varies a lot depending on the parameters, having both positive and negative effects. On the one hand, we reduce the size of all the coefficients in the lattice, thus reducing the reduction time. On the other hand, we generate more potential small vectors[2] with norms smaller than the norm of w. For this reason, the probability of success of the attack decreases, the vector w more likely to be a linear combination of vectors already in the reduced basis. For example, with 7 signatures we find in average w to be the third or fourth vector in the reduced basis without preprocessing, whereas with S_{11} it is more likely to appear in position 40 on average.

The positive effect of preprocessing is most noticeable for $u = 4$ and $u = 5$, as shown in Figure 2. For instance, using S_{15} and $u = 4$ lowers the overall time by a factor up to 5.7. For $u = 5$, we gain a factor close to 3 by using either S_{15} or S_{19}.

For $u > 5$, using preprocessed traces is less impactful. For large Δ such as $\Delta \approx 2^{25}$, we still note some lower overall times when using S_{15} and S_{19}, up to a factor 2. When the bitsize gets smaller, reducing the size of the coefficients in the lattice is less impactuf. Details are given in Appendix B.

Balancing the block-size of BKZ. Finally, we vary the block-size in the BKZ algorithm. We fix $\Delta \approx 2^3$ and use no preprocessing. We plot the results in Figure 3 for 6 and 7 signatures. For other values of u, the plot is very similar and we omit them in Figure 3 for ease of lecture. Without any surprise, we see that as we increase the block-size, the probability of success increases, however the reduction time increases significantly as well. This explains the results shown in Table 4 and Table 5: to reach the best probability of success one needs to increase the block-size in BKZ (we did not try any block-size greater than 40), but to get the fastest key recovery attack, the block-size is chosen between

[2]In the sense of vectors exhibited in (9).

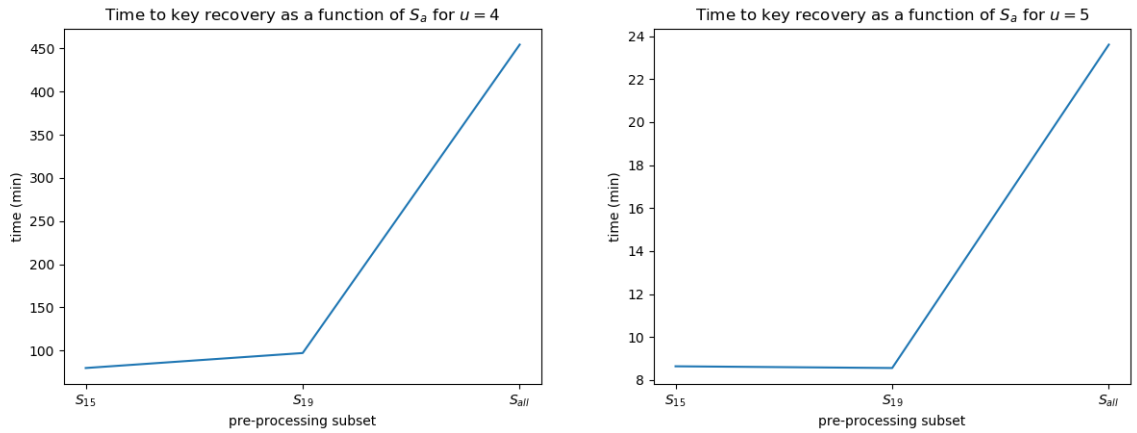

Figure 2: Analyzing the overall time to recover the secret key as a function of the preprocessing subset for 4 and 5 traces. The other parameters are fixed: $\Delta \approx 2^{25}$ and BKZ-25.

20 and 25, except for 3 signatures where the probability of success is too low with these parameters. Details are given in Appendix C.

6 Error resilience analysis

It is not unexpected to have errors in the traces collected during the side-channel attack. Obtaining a set of error-free traces requires some amount of work on the signal processing side. Prior to [DDME+18], the presence of errors in traces was either ignored or preprocessing was done on the traces until an error-free sample was found, see [GPP+16, ARAM17]. In [DDME+18], it is shown that the lattice attack still successfully recovers the secret key even when some traces contain errors. An error in the setup given in [DDME+18] corresponds to an incorrect bound on the size of the values being collected. In our setup, a trace without errors corresponds to a trace where every single coefficient in the wNAF representation of k has been identified correctly as either non-zero or not. The probability of having an error in our setup is thus much higher. Side-channel attacks without any errors are very rare. Both [vdPSY15] and [DDME+18] give some analysis of the attacks `FLUSH + RELOAD` and `Prime + Probe` in real life scenarios.

In [FWC16], the results presented in the paper assume the `FLUSH + RELOAD` is implemented perfectly, without any error. In particular, to obtain 4 perfect traces and be able to run their experiment and find the key, one would need to have in average 8 traces from `FLUSH + RELOAD` – the probability to conduct to a perfect reading of the traces being 56 % as pointed out in [vdPSY15]. In our work, we show that it is possible to recover the secret key using only 4, even erroneous, traces. However, the probability of success is very low.

Recall that an error in our case corresponds to a flipped digit in the trace of k. The following Table 6 shows the probability of success of the attack in the presence of errors. We ran experiments for BKZ-25 using $\Delta \approx 2^3$ and traces taken from S_{all}. We average over 5000 experiments.

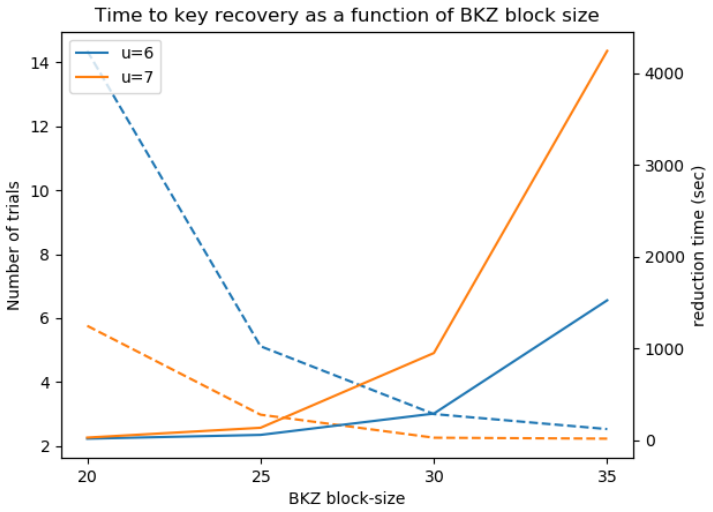

Figure 3: Analyzing the number of trials to recover the secret key and the reduction time of the lattice as a function of the block-size of BKZ. We consider the cases where $u = 6$ and $u = 7$. The dotted lines correspond to the number of trials, and the continued lines to the reduction time in seconds.

Number of signatures	Probability of success (%)				
u	0 error	5 errors	10 errors	20 errors	30 errors
4	0.28	$\ll 1$	0	0	0
5	4.58	0.86	0.18	$\ll 1$	0
6	19.52	5.26	1.26	0.14	$\ll 1$
7	33.54	10.82	3.42	0.32	$\ll 1$
8	35.14	13.26	4.70	0.58	$\ll 1$

Table 6: Error analysis using BKZ-25, $\Delta \approx 2^3$ and S_{all}.

We write $\ll 1$ when the attack succeeded less than five times over 5000 experiments, thus making it difficult to evaluate the probability of success.

The attack works up to a resilience to 2% of errors, i.e., of flipped digits. Indeed, for $u = 6$, we were able to recover the secret key with 30 errors, meaning 30 flipped digits over 6×257 digits.

Different types of errors. There exists two possible types of errors. In the first case, a coefficient which is zero is evaluated as a non-zero coefficient. In theory, this only adds a new variable to the system, i.e., the number ℓ of non-zero coefficients is overestimated. This does not affect the probability of success much. Indeed, we just have an overly-constrained system. We can see in Figure 4 that the probability of success of the attack indeed decreases slowly as we add errors of this form. With errors only of this form, we were able to recover the secret key up to nearly 4% of errors, for instance with $u = 6$, using BKZ-35, see Table 9 in Appendix D.

The other type of errors consists of a non-zero coefficients which is misread as a zero coefficient. In this case, we lose information necessary for the key recovery and thus this type of error affects the probability of success far more importantly as can also be seen in Figure 4. In this setup, we were not able to recover the secret key when more than 3 errors of this type appear in the set of traces

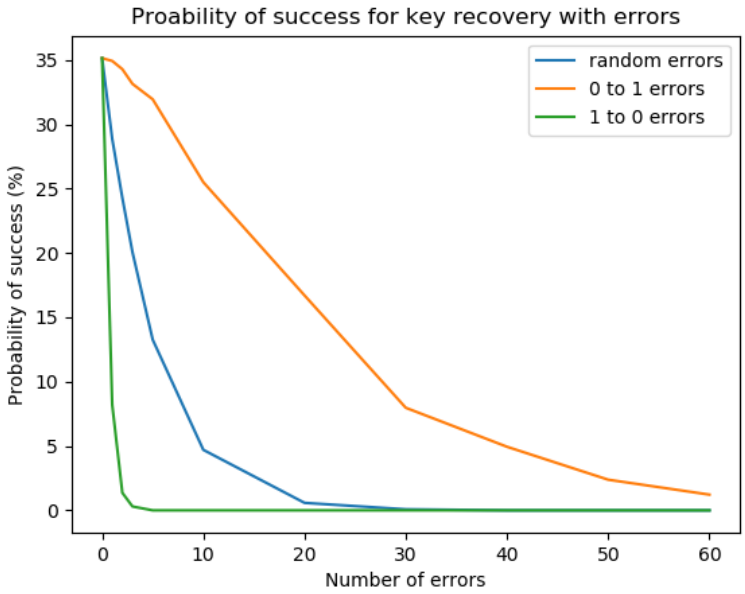

Figure 4: Probability of success for key recovery with various types of errors when using $u = 8$, BKZ-25, $\Delta \approx 2^3$, and no preprocessing.

considered. More details on the probabilities of success of these two types of errors can be seen in Appendix D.

If the signal processing method is hesitant between a 1 or 0 digit, we would recommend to favor putting 1 instead of 0 to increase the chance of having an error of type $0 \to 1$, for which the attack is a lot more tolerant.

7 An attempt at using Coppersmith's methods

Given that the setup of the Extended Hidden Number Problem gives a system of modular equations, it is natural to ask whether this system can be solved using Coppersmith's method for finding small roots of integer polynomials. The idea is to use LLL to construct as many polynomials as unknowns which share the same roots as the original polynomials. Coppersmith's methods in the case of bivariate polynomials can be expressed as the following theorem [Gal12, Theorem 19.2.1]

Theorem 1. *Let $F(x, y) \in \mathbb{Z}[x, y]$ be a polynomial of total degree d. Let $X, Y, M \in \mathbb{N}$ be such that $XY < M^{1/d-\epsilon}$ for some $0 < \epsilon < 1/d$. Then one can compute in time polynomial in $\log(M)$ and $1/\epsilon > d$ polynomials $F_1(x, y), F_2(x, y) \in \mathbb{Z}[x, y]$ such that for all $(x_0, y_0) \in \mathbb{Z}^2$ with $|x_0| < X, |y_0| < Y$ and $F(x_0, y_0) \equiv 0 \pmod{M}$, one has $F_1(x_0, y_0) = F_2(x_0, y_0) = 0$ over $\mathbb{Z}$.*

When Theorem 1 is generalized to m variables, it requires conditions of the form $|x_i| < M^{\alpha_i}$ for $1 \leqslant i \leqslant m$, and $\sum \alpha_i < 1/d$ [Jut98].

The bound essentially comes from the construction of the lattice built to find the solutions. One usually considers the polynomials

$$q_{i_1,\dots,i_m,j}(x_1, \dots, x_m) = x_1^{i_1} \dots x_m^{i_m} F(x_1, \dots, x_m)^j$$

and builds a lattice basis whose rows coefficients are the coefficients of these polynomials.

In our setup. Without eliminating the α variable, we are given the following system of modular equations

$$F_i(d_{1,1}, \ldots, d_{1,\ell_1,}, \ldots d_{u,1}, \ldots, d_{u,\ell_u}, \alpha) = \alpha r_i - \sum_{j=1}^{\ell_i} \rho_{ij} d_{ij} - \beta_i \equiv 0 \pmod{q}$$

for $1 \leqslant i \leqslant u$, and where $\rho_{ij} = 2^{\lambda_{ij}} s_i \pmod{q}$ and $\beta_i = s_i \bar{k}_i - H(m_i) \pmod{q}$, for $1 \leqslant j \leqslant \ell_i$ using the same notations as in the rest of the paper. This system has u equations and $T + 1$ unknowns. Recall that $T = \sum_{i=1}^{u} \ell_i$ and all the F_i are linear polynomials.

With elimination, we are given the following system of modular equations

$$F_i(d_{1,1}, \ldots, d_{1,\ell_1,}, \ldots d_{u,1}, \ldots, d_{u,\ell_u}) = \sum_{j=1}^{\ell_1} \tau_{j,i} d_{1,j} + \sum_{j=1}^{\ell_i} \sigma_{i,j} d_{i,j} - \gamma_i \equiv 0 \pmod{q}$$

for $2 \leqslant i \leqslant u$, and where τ_{ji}, σ_{ij} and γ_i are defined as in Section 3.3. This system has $u - 1$ equations and T unknowns. Again, all the F_i are linear polynomials. In both cases, the total degree of the polynomial F_i is $d = 1$.

Let D be a bound on the unknowns d_{ij}, i.e., $|d_{ij}| < D$. The variable α is bounded by q by definition. The condition in the theorem requires that

$$D^T q < q^{1-\epsilon}$$

in the first case without elimination which necessarily implies that $D < 1$. In the case where α is being eliminated from the equations, we have

$$D^T < q^{1-\epsilon}$$

which means $D < q^{(1-\epsilon)/T}$. When $\epsilon \to 1$, we get again that $D < 1$, and when $\epsilon \to 0$, we have $D < q^{1/T}$. This bound D is actually determined by the inequality $(1.02)^n (\det \mathcal{L})^{1/n} < q$, with $n = \dim \mathcal{L}$ which essentially guarantees that solving a system over the integers will yield the same solutions as our initial modular system.

If we consider the attack scenario where the number of signatures $u \in [3, 8]$, the value of T is lower bounded by 150 on average. This results in the condition $D < 2$, but restricting the bound on the d_{ij} to 2 at best seems too restrictive for the key recovery to be successful.

We have implemented the lattice basis with and without shifts, i.e., multiplying our polynomials by some of the d_{ij} to confirm this and have failed to recover the secret key. We give our lattice construction in the elimination case in Appendix E.

Acknowledgement

We would like to thank Nadia Heninger for discussions about possible lattice constructions, Medhi Tibouchi for answering our side-channel questions, Alenka Zajic and Milos Prvulovic for providing us with traces from OpenSSL that allowed us to confirm our results on a deployed implementation, Daniel Genkin for pointing us towards the Extended Hidden Number Problem, and Pierrick Gaudry for his precious support and reading. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations.

References

[ARAM17] Jiji Angel, R. Rahul, C. Ashokkumar, and Bernard Menezes. DSA signing key recovery with noisy side channels and variable error rates. In *INDOCRYPT*, volume 10698 of *Lecture Notes in Computer Science*, pages 147–165, 2017.

[AS08] Onur Acıiçmez and Werner Schindler. A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL. In Tal Malkin, editor, *Topics in Cryptology – CT-RSA 2008*, pages 256–273, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[ASK06] Onur Aciçmez, Werner Schindler, and Çetin K. Koç. Cache based remote timing attack on the AES. In *Proceedings of the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'07, pages 271–286, Berlin, Heidelberg, 2006. Springer-Verlag.

[B+] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform.

[BV96] Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 129–142, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[BvdPSY14] 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 Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, pages 75–92, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.

[DDME+18] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):171–191, May 2018.

[dt16] The FPLLL development team. FPLLL, a lattice reduction library. Available at `https://github.com/fplll/fplll`, 2016.

[FWC16] Shuqin Fan, Wenbo Wang, and Qingfeng Cheng. Attacking OpenSSL implementation of ECDSA with a few signatures. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1505–1515, New York, NY, USA, 2016. ACM.

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, New York, NY, USA, 1st edition, 2012.

[Gor98] Daniel M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, April 1998.

[GPP+16] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *CCS*, pages 1626–1638, 2016.

[HGS01] N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Des. Codes Cryptography*, 23(3):283–290, July 2001.

[HR07] Martin Hlaváč and Tomáš Rosa. Extended hidden number problem and its cryptanalytic applications. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography*, pages 114–133, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Secur.*, 1(1):36–63, August 2001.

[Jut98] Charanjit S. Jutla. On finding small solutions of modular multivariate polynomial equations. In *EUROCRYPT*, 1998.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[NIS13] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*, 2013.

[NS03] Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30(2):201–217, Sep 2003.

[NS06] Phong Nguyen and Damien Stehlé. LLL on the average. *Algorithmic Number Theory*, pages 238–256, 2006.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53(2-3):201–224, August 1987.

[Sch94] Claus-Peter Schnorr. Block reduced lattice bases and successive minima. *Combinatorics, Probability & Computing*, 3:507–522, 1994.

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66(2):181–199, September 1994.

[SYB14] David Schwartz, Noah Youngs, and Arthur Britto. The ripple protocol consensus algorithm. 2014.

[vdPSY15] Joop van de Pol, Nigel P. Smart, and Yuval Yarom. Just a little bit more. In Kaisa Nyberg, editor, *Topics in Cryptology — CT-RSA 2015*, pages 3–21, Cham, 2015. Springer International Publishing.

[WF17] Wenbo Wang and Shuqin Fan. Attacking OpenSSL ECDSA with a small amount of side-channel information. *Science China Information Sciences*, 61(3):032105, Aug 2017.

[YB14] Yuval Yarom and Naomi Benger. Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack. *IACR Cryptology ePrint Archive*, 2014:140, 2014.

[YF14] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 719–732, Berkeley, CA, USA, 2014. USENIX Association.

A Bitsize of Δ effect over the key recovery total time

We analyze the effect of the bitsize of Δ. We fix BKZ-25 and use no preprocessing. We average over 5000 experiments.

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Key recovery total time (min)
4	25	S_{all}	0	0.14	31	375
4	25	S_{all}	1	0.16	31	330
4	25	S_{all}	3	0.28	32	**191**
4	25	S_{all}	5	0.22	30	234
4	25	S_{all}	10	0.24	33	228
4	25	S_{all}	15	0.16	39	411
4	25	S_{all}	20	0.20	45	379
4	25	S_{all}	25	0.20	54	454
4	25	S_{all}	30	0.10	31	515
5	25	S_{all}	0	3.74	37	16
5	25	S_{all}	1	4.60	36	13
5	25	S_{all}	3	4.58	34	**12**
5	25	S_{all}	5	4.38	34	13
5	25	S_{all}	10	3.92	36	15
5	25	S_{all}	15	4.62	41	15
5	25	S_{all}	20	4.60	52	19
5	25	S_{all}	25	4.52	64	23
5	25	S_{all}	30	4.18	88	35
6	25	S_{all}	0	15.94	77	8
6	25	S_{all}	1	19.96	61	5
6	25	S_{all}	3	19.52	57	**5**
6	25	S_{all}	5	20.10	59	5
6	25	S_{all}	10	19.04	63	5
6	25	S_{all}	15	20.34	72	6
6	25	S_{all}	20	20.58	92	7
6	25	S_{all}	25	20.02	91	7
6	25	S_{all}	30	19.26	164	14
7	25	S_{all}	0	28.86	185	10
7	25	S_{all}	1	33.00	134	7
7	25	S_{all}	3	33.54	136	**6**
7	25	S_{all}	5	33.69	142	7
7	25	S_{all}	10	33.99	149	7
7	25	S_{all}	15	33.81	186	9
7	25	S_{all}	20	34.94	229	11
7	25	S_{all}	25	31.68	300	15
7	25	S_{all}	30	32.08	351	18
8	25	S_{all}	0	32.12	322	16
8	25	S_{all}	1	36.40	237	101
8	25	S_{all}	3	35.14	227	10
8	25	S_{all}	5	36.00	211	**9**
8	25	S_{all}	10	34.86	245	11
8	25	S_{all}	15	36.18	296	13
8	25	S_{all}	20	35.48	376	17
8	25	S_{all}	25	36.12	460	21
8	25	S_{all}	30	34.54	573	27

B Preprocessing effect over the key recovery total time

We analyze the effect of the preprocessing. We fix BKZ-25 and $\Delta \approx 2^3, 2^{25}$. We average over 5000 experiments.

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Key recovery total time (min)
4	25	S_{11}	25	0.20	9	**79**
4	25	S_{15}	25	0.52	24	79
4	25	S_{19}	25	0.50	29	97
4	25	S_{all}	25	0.20	54	454
5	25	S_{11}	25	1.70	17	17
5	25	S_{15}	25	5.74	29	8
5	25	S_{19}	25	6.28	32	**8**
5	25	S_{all}	25	4.52	64	23
6	25	S_{11}	25	3.64	38	17
6	25	S_{15}	25	22.12	77	5
6	25	S_{19}	25	25.12	77	**5**
6	25	S_{all}	25	20.02	91	7
7	25	S_{11}	25	3.40	55	27
7	25	S_{15}	25	26.20	151	9
7	25	S_{19}	25	43.90	162	**7**
7	25	S_{all}	25	31.68	300	15
8	25	S_{11}	25	4.50	85	31
8	25	S_{15}	25	32.50	237	12
8	25	S_{19}	25	43.90	267	**10**
8	25	S_{all}	25	36.12	460	21

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Key recovery total time (min)
4	25	S_{11}	3	0.18	9	89
4	25	S_{15}	3	0.52	24	**77**
4	25	S_{19}	3	0.38	29	130
4	25	S_{all}	3	0.28	32	191
5	25	S_{11}	3	1.18	19	27
5	25	S_{15}	3	5.90	30	8
5	25	S_{19}	3	6.50	32	**8**
5	25	S_{all}	3	4.58	34	12
6	25	S_{11}	3	4.04	40	16
6	25	S_{15}	3	20.36	78	6
6	25	S_{19}	3	24.76	72	**4**
6	25	S_{all}	3	19.52	57	5
7	25	S_{11}	3	4.15	60	24
7	25	S_{15}	3	27.00	158	9
7	25	S_{19}	3	35.25	173	8
7	25	S_{all}	3	33.54	135	**6**
8	25	S_{11}	3	4.40	88	33
8	25	S_{15}	3	35.20	249	11
8	25	S_{19}	3	40.70	268	11
8	25	S_{all}	3	35.14	227	**10**

C BKZ block-size effect over the key recovery total time

We analyze the effect of the BKZ block-size. We set $\Delta \approx 2^3$ and use no preprocessing. We average over 5000 experiments.

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Key recovery total time (min)
4	20	S_{all}	3	0	5	0
4	25	S_{all}	3	0.28	32	**191**
4	30	S_{all}	3	1.30	302	387
4	35	S_{all}	3	4.10	3763	1529
5	20	S_{all}	3	0.82	9	19
5	25	S_{all}	3	4.58	34	**12**
5	30	S_{all}	3	11.60	225	32
5	35	S_{all}	3	20.18	1964	162
6	20	S_{all}	3	6.96	16	**4**
6	25	S_{all}	3	19.52	57	5
6	30	S_{all}	3	32.96	290	14
6	35	S_{all}	3	39.52	1525	64
7	20	S_{all}	3	17.35	28	**2**
7	25	S_{all}	3	33.54	136	6
7	30	S_{all}	3	44.20	950	35
7	35	S_{all}	3	44.80	4245	158
8	20	S_{all}	3	29.40	43	**2**
8	25	S_{all}	3	35.14	227	10
8	30	S_{all}	3	46.66	1894	68
8	35	S_{all}	3	44.70	8119	302

D Analysis of errors

We analyze the effect of two possible kind of errors on the probability of success of our attack, using BKZ-25, $\Delta \approx 2^3$ and no preprocessing. We average over 5000 experiments. We write $\ll 1$ when the attack succeeded less than five times over 5000 experiments.

Number of signatures	Probability of success (%)								
u	0 errors	1 error	5 errors	10 errors	20 errors	30 errors	40 errors	50 errors	60 errors
4	0.28	0.18	0.10	$\ll 1$	0	0	0	0	0
5	4.58	3.82	2.70	1.06	0.32	$\ll 1$	0	0	0
6	19.52	10.79	13.88	7.90	2.94	0.86	0.36	0.10	$\ll 1$
7	33.54	31.06	26.04	18.36	9.24	4.54	?	1.02	0.50
8	35.14	34.92	31.94	25.50	16.70	7.96	4.94	2.48	1.22

Table 7: Error $0 \to 1$ analysis using BKZ-25, $\Delta \approx 2^3$ and S_{all}.

Number of signatures	Probability of success (%)			
u	0 errors	1 error	2 errors	3 errors
4	0.28	0	0	0
5	4.58	0.36	$\ll 1$	0
6	19.52	2.70	0.36	$\ll 1$
7	33.54	5.54	1.00	0.12
8	35.14	8.20	1.36	0.30

Table 8: Error $1 \to 0$ analysis using BKZ-25, $\Delta \approx 2^3$ and S_{all}.

When considering many errors, the probability of success can be increased by augmenting the block-size in the BKZ algorithm, as can be seen in Table 9.

Number of signatures	Probability of success (%)															
	30 errors				40 errors				50 errors				60 errors			
BKZ-β	25	30	35	40	25	30	35	40	25	30	35	40	25	30	35	40
5	$\ll 1$	0.24	0.35	0.75	0	$\ll 1$	$\ll 1$	0.42	0	0	0	0	0	0	0	0
6	0.86	2.48	3.58	3.97	0.36	0.90	1.18	2.28	0.10	0.36	0.58	0.94	$\ll 1$	$\ll 1$	0.12	?
7	4.54	6.44	7.32	8.73	?	3.54	3.48	4.58	1.02	1.26	1.84	3.26	0.50	0.62	1.20	1.43
8	7.96	10.46	11.78	10.98	4.94	6.12	6.73	7.12	2.48	3.26	3.78	4.64	1.22	1.84	1.89	?

Table 9: Errors $0 \to 1$ analysis with $\Delta \approx 2^3$, S_{all} and increasing block-size.

E Lattice construction for Coppersmith's methods

We construct the following lattice basis

$$\begin{pmatrix}
d_{2,1} & d_{2,2} & \dots & d_{2,\ell_2} & \dots & d_{u,1} & \dots & d_{u,\ell_u} & d_{1,1} & d_{1,2} & \dots & d_{1,\ell_1} & 1 & \\
D & \sigma_{2,2}\sigma_{2,1}^{-1}D & \dots & \sigma_{2,\ell_2}\sigma_{2,1}^{-1}D & 0 & 0 & 0 & 0 & \tau_{1,2}\sigma_{2,1}^{-1}D & \tau_{2,2}\sigma_{2,1}^{-1}D & \dots & \tau_{\ell_1,2}\sigma_{2,1}^{-1}D & -\gamma_1\sigma_{2,1}^{-1} & Eq_2 \\
 & Dq & & & & & & & & & & & & \\
 & & \ddots & & & & & & & & & & & \vdots \\
 & & & Dq & & & & & & & & & & \\
 & & & & \ddots & D & \dots & \sigma_{u,\ell_u}\sigma_{u,1}^{-1}D & \tau_{1,u}\sigma_{u,1}^{-1}D & \tau_{2,u}\sigma_{u,1}^{-1}D & \dots & \tau_{\ell_u,u}\sigma_{u,1}^{-1}D & -\gamma_u\sigma_{u,1}^{-1} & Eq_u \\
 & & & & & & Dq & & & & & & & \\
 & & & & & & & \ddots & & & & & & \\
 & & & & & & & & \ddots & & & & & \\
 & & & & & & & & & \ddots & & & & \\
 & & & & & & & & & & \ddots & & & \\
 & & & & & & & & & & & Dq & & \\
 & & & & & & & & & & & & q &
\end{pmatrix}$$

The dimension of this lattice is $\dim L = T + 1$ and the determinant is given by

$$\det L = D^T q^{T-u+2}$$

In order for the Coppersmith method to work, we require that $(1.02)^n (\det L)^{1/n} < q$, where $n = \dim \mathcal{L}$. This implies we need the condition

$$D < \left(\frac{q^{u-1}}{1.02^{(T+1)^2}} \right)^{1/T}.$$

Numerically, we get $D < 1$ for $u \in [3, 8]$.