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. One way of attacking ECDSA with wNAF implementation for the scalar multiplication is to perform a side-channel analysis to collect information, then use a lattice based method 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, thus reaching the theoretical bound given by Fan, Wang and Cheng, whereas best previous methods required at least 4 signatures in practice. Our attack is more efficient than previous attacks, in particular compared to times reported by Fan et al. at CCS 2016 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 overall 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.

Keywords: Public key cryptography · ECDSA · side-channel attack · windowed Non-Adjacent Form · Lattice techniques.

1 Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) [17], first proposed in 1992 by Scott Vanstone [34], is a standard public key signature protocol widely deployed. ECDSA is used in the latest library TLS 1.3 [27], the encryption standard OpenPGP [7] and smart cards [26]. It is also implemented in the well-known library OpenSSL [33], and can be found in cryptocurrencies such as Bitcoin [21], Ethereum [6] and Ripple [31, 8]. 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. Because of this, it is recognized as a standard signature algorithm by several institutes such as ISO since 1998, ANSI since 1999, and IEEE and NIST since 2000.

The ECDSA signing algorithm requires the computation of a 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. 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 execution time as the latter is dependent on the number of non-zero digits. The wNAF representation is present in implementations such as for Bitcoin, as well as in the libraries Cryptlib, BouncyCastle and Apple's Common-Crypto. Moreover, until very recently (May 2019), wNAF was present in all three branches of OpenSSL.

However, implementing the scalar multiplication using wNAF representation makes the protocol vulnerable. Indeed, with wNAF, a common way to attack ECDSA is to first collect valuable information on the ephemeral key k leaked by side-channel attacks before using this knowledge to recover the secret key.

Side-channel attacks were first introduced about two decades ago by Kocher et al. [19], and have since been used to break many implementations, and in particular cryptographic primitives such as AES [2], RSA [1], and ECDSA [17]. They allow to recover secret information otherwise hidden from the public throughout observable leakage. For instance, in our case, this leakage corresponds to differences in the execution time of a part of the signing algorithm, observable by monitoring the cache. This corresponds to a specific type of side-channel attacks called cache timing attacks.

In the case of ECDSA, cache side-channel attacks such as FLUSH+RELOAD [36, 37] have been used to recover information about the sequence of operations used to execute the scalar multiplication. These operations are either doubling or addition depending on the bits of k. This information is usually referred to as a double-and-add chain or the trace of k. The main question considered at this point is how many traces need to be collected to successfully recover the secret key. Indeed, from an attacker's perspective, the least traces necessary, the more efficient the attack is. This quantity depends on how much information can be extracted from a single trace and how combining information from multiple traces is used to recover the key. In this paper, we work on the latter to minimize the number of traces needed.

The nature of the information obtained from the side-channel attack allows to determine what kind of method should be carried out to recover the secret key. Attacks on ECDSA are inspired by attacks on a similar cryptosystem, DSA. In 2001, Howgrave-Graham and Smart [16] showed how knowing partial information of the nonce k in DSA can lead to a full secret key recovery. Later, Nguyen and Shparlinski [24] gave a polynomial time algorithm that recovers the secret key in ECDSA as soon as consecutive bits of the ephemeral key are known. To do so, 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) [5]. HNP allows to recover a secret integer when the attacker is given many samples of consecutive bits of modular multiples of this integer. Moreover, they reduced the instance of HNP to well-known lattice problems: the Closest Vector Problem (CVP) and the Shortest Vector Problem (SVP). Thus, the basic structure of the attack algorithm is to construct a lattice which contains the knowledge of consecutive bits of the epheremal keys, and by solving CVP or SVP, to recover the secret key. This type of attack has been done in [4, 25, 35]. These results consider perfect traces, but obtaining traces without any misreadings is very rare. Instead of assuming that perfect traces are required for the attack to succeed, Dall et al [10] included an error-resilience analysis to their attack: they showed that key recovery with HNP is still possible even in the presence of erroneous traces.

More recently, in 2016, Fan, Wang and Cheng [11] used another lattice-based method to attack ECDSA: the Extended Hidden Number Problem (EHNP) [15]. EHNP mostly differs from HNP by the nature of the information given as input. Indeed, the information required to construct an instance of EHNP is not sequences of consecutive bits, but the positions of the non-zero coefficients in any representation of integers. In particular, this results in a different lattice construction. In [11], the authors are able to extract 105.8 bits of information per signature on average, and thus require in theory only 3 signatures to recover a 256-bit secret key. In practice, they were able to recover the secret key using 4 error-free traces.

In order to optimize an attack on ECDSA various aspects should be considered. By minimizing the number of signatures required in the lattice construction, one minimizes the number of traces needed to be collected during the side-channel attack. Moreover, reducing the time of the lattice part of the attack, and improving the probability of success of key recovery allows to reduce the overall time of the attack. Finally, to match a realistic scenario, one should analyze the resilience to errors of the attack. In this paper, we improve on all four of these aspects.

Contributions: We focus on the lattice part of the attack, *i.e.*, the exploitation of the information gathered by the side-channel attack. We first assume we obtain a set of error-free traces from a side-channel analysis. We preselect some of these traces to optimize the attack. The main idea of the lattice part is then to use the ECDSA equation and the knowledge gained from the selected traces to construct a set of modular equations which include the secret key as an unknown. These modular equations are then incorporated into a lattice basis similar to the one given in [11], and a short vector in this lattice will contain the necessary information to reconstruct the secret key. We call "experiment" one run of this algorithm which succeeds if we are able to reconstruct the secret key, and otherwise fails.

It is important to note that we use error-free traces to analyze the attack. However, in a real world scenario, obtaining such traces is very hard. Thus we also show that key recovery is still possible with erroneous traces.

A new preprocessing method. The idea of selecting good traces beforehand has already been explored in [35]. The authors suggest three rules to select traces that improve the attack on the lattice part. 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 - which completely differs from [35] - 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 overall time of the attack by a factor of 7.

Analyzing the attack. 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 either the overall time or the probability of success.

Let us first focus on the attack time. Note that in this paper, when talking about the overall time of the attack, we consider the average time of a single experiment multiplied by the expected number of trials necessary to recover the secret key. We compare our times with the numbers reported in [11, Table 3] with method C.[3] Indeed, methods A and B in [11] use extra information that comes from choices in the implementation which we choose to ignore as we want our analysis to remain as general as possible. When using 4 signatures, our attack is slightly slower[4] than the attack in [11]. However, when considering more than 4 signatures, our attack is faster than the times reported in [11]. We experiment with up to 8 signatures to further improve our overall time. In this case, our attack runs at best in 2 minutes and 25 seconds. Note that timings for 8 signatures are not reported in [11], and the case of 3 signatures was never reached before our work. In Table 1, we compare our times with the fastest times reported by [11]. We choose their fastest times but concerning our results we choose to report experiments which are faster (not the fastest) with, if possible, better probability than theirs.

The overall time of the attack is also dependent on the success probability of key recovery. From Table 2, one can see that our success probability is higher than [11] except for 7 signatures. We gain a factor up to 5 for 5 signatures for example.

Finding the key with only three signatures. Overall, combining a new preprocessing method, a modified lattice construction and a careful choice of parameters allows us to mount an attack which works in practice with only 3 signatures.

[3] In order to have a fair comparison with our methodology and timings, we believe that the times reported in [11] would have to be multiplied by the expected number of trials necessary for their attack to work. This would increase their overall time a lot. For example, using 5 signatures, their best overall time would be around 15 hours instead of 18 minutes.

[4] For 4 signatures, no times are reported without method A. Thus, we have no other choice than to compare our times with theirs, using A. Yet their time for 4 signatures without A should at least be the time they report with it.

Table 1: Comparing attack times with [11].

Number of signatures	Our attack Time	Success (%)	[11] Time	Success (%)
3	39 hours	0.2%	–	–
4	1 hour 17 minutes	0.5%	41 minutes	1.5%
5	8 minutes 20 seconds	6.5%	18 minutes	1%
6	$\approx$ 5 minutes	25%	18 minutes	22%
7	$\approx$ 3 minutes	17.5%	34 minutes	24%
8	$\approx$ 2 minutes	29%	–	–

Table 2: Comparing best attack success probability with [11].

Number of signatures	Our attack Success (%)	Time	[11] Success (%)	Time
3	0.2%	39 hours	–	–
4	4%	25 hours 28 minutes	1.5%	41 minutes
5	20%	2 hours 42 minutes	4%	36 minutes
6	40%	1 hour 4 minutes	35%	1 hour 43 minutes
7	45%	2 hours 36 minutes	68%	3 hours 58 minutes
8	45%	5 hours 2 minutes	–	–

However, the probability of success in this case is very low. Without using our preprocessing method, 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 using a single core. Note that this time can be greatly reduced when parallelizing the process, *i.e.*, each trial can be run on a separate core. On the other hand, if we use our preprocessing method, with 3 signatures we obtain a probability of success of 0.2% and an overall 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 [11] reported that in practice, the key couldn't be recovered using less than 4 signatures and we improve on their result.

Resilience to errors. In addition to the attack analysis, we also 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 required collecting more traces. As pointed out in [11], more or less twice as many signatures are needed if errors are considered. In practice, this led [11] 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 readings, *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 preprocessing methods that none of the 1 digits have been flipped to 0.

Looking at Coppersmith. 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 method for finding small roots of integer polynomials. This would be an alternative to EHNP. Albeit our attempts to apply Coppersmith's method were not successful as a bound on the unknowns is not satisfied, we briefly sketch our ideas hoping it could lead to further improvements.

Organization: In Section 2, we introduce some background on ECDSA and the wNAF representation. Moreover, we give details on lattices and well known reduction algorithms. In Section 3, we explain how the Extended Hidden Number Problem can be transformed into a lattice problem. We explicit the lattice basis and give an analysis on the length of the short vectors found in the reduced basis. In Section 4, we introduce our preprocessing method 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 our attempt to use Coppersmith's method instead of EHNP.

2 Preliminaries

2.1 Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm is a variant of the Digital Signature Algorithm, DSA, [22] which uses elliptic curves instead of finite fields. The parameters used in the ECDSA algorithm are an elliptic curve E over a prime field, a generator G of prime order q and a hash function H to $\mathbb{Z}_q$. 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, one executes the following steps. Randomly select an ephemeral key $k \leftarrow_R \mathbb{Z}_q$ and compute $[k]G$. Let r be the x-coordinate of $[k]G$ mapped to $\mathbb{Z}_q$. 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 [14], various methods to compute fast exponentiation are presented. One of those introduces the NAF representation of an integer. 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. 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. It is presented in Algorithm 1.

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

The NAF representation can be combined with a sliding window method to further improve the execution time. 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. For instance, in OpenSSL (up to the latest versions using wNAF, 1.1.1b for example), the window size usually chosen was $w = 3$. Apple's CommonCrypto library uses $w = 1$. The scalar k is converted into wNAF form using Algorithm 2. Note that in Algorithm 2, the sequence of digits m_i belongs

```
Input: k ∈ Z⁺, w ∈ N
Output: (m_0, m_1, ..., m_n), i.e., k in its wNAF representation
i = 0;
while k > 0 do
    if k (mod 2) = 1 then
        m_i = k (mod 2^{w+1});
        if m_i ≥ 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

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

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

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 $\mathcal{B} \in \mathbb{Z}^{n\times n}$. The lattice $\mathcal{L}(\mathcal{B})$ generated by $\mathcal{B}$ consists of all integer combinations of the row vectors in $\mathcal{B}$. The determinant of a lattice is the absolute value of the determinant of a basis matrix: $\det \mathcal{L}(\mathcal{B}) = |\det \mathcal{B}|$. The discreteness property ensures that there is a $\lambda_1 > 0$ such that the length of one of the shortest non-zero vectors v_1 in the lattice satisfies $||v_1|| = \lambda_1$. The LLL algorithm [20] takes as 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 $||b_i||_2 \leq 2^{(n-1)/2}\lambda_i$, where λ_i is the i^{th} successive minimum of the lattice. 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 [23]. 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 [28, 30] is exponential in a given block-size β and polynomial in the lattice dimension n. It outputs a reduced lattice basis whose vectors b_i satisfy the approximation $||b_i||_2 \leq i\gamma_\beta^{(n-i)/(k-1)}\lambda_i$ [29], where γ_β is the Hermite constant. In practice, Chen and Nguyen [9] 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 a 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 digits in the representation of k. However, the value of these digits 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 different nonce k. We assume we have the corresponding trace of the nonce, that is, the equivalent of the double-and-add chain of $[kG]$. The goal of the attack is to recover the secret α while optimizing either the number of signatures required or the overall time of the attack.

3.1 The Extended Hidden Number 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 q, η_{ij}, a_i, $b_{i,j}$, c_i, ℓ_i are all known for $1 \leqslant i \leqslant u$ (see [15], 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 the 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 digits k_j are unknown, however, the positions λ_j are supposed to be known via 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 [11], 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 the expansion of k in Equation (3), we obtain a system of modular equations of the form

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

for $1 \leqslant i \leqslant u$, 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} equation, $\lambda_{i,j}$, which is the position of the j^{th} non-zero digit in k for the i^{th} equation and $\bar{k}_i$ defined above. We can then use Equation (5) as input to the Extended Hidden Number Problem, following the method explained in [15]. 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 [11].

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 [11], 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 $1 \leqslant j \leqslant \ell_i$ and $2 \leqslant i \leqslant u$. 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. Recall that μ_j denotes the window-size of the coefficient d_j. For now, this value is equal to w, the window-size considered in the wNAF algorithm. However, since this value will be modified later, we use the notation μ_j. Let $m = \max_{i,j} \mu_{ij}$ for $1 \leqslant j \leqslant \ell_i$ and $2 \leqslant i \leqslant u$. We now explicit the construction of the lattice basis $\mathcal{B}$ used in our attack. We use a scaling factor $\Delta \in \mathbb{N}$ to be defined later. The lattice basis is given by

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

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, corresponds to a d_{ij}. The determinant of the lattice $\mathcal{L} = \mathcal{L}(\mathcal{B})$ is given by

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

The lattice is built such that there exists $z \in \mathcal{L}$ which contains the unknowns $d_{i,j}$. To find it, we know there exists values $t_2, t_3, \dots, t_u$ such that if $v = (t_2, \dots, t_u, d_{1,1}, \dots, d_{u,\ell_u}, -1)$, we get

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

and

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

If we are able to find z in the lattice, then we can reconstruct the secret key α. In order to find z, we estimate its norm and make sure z 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 z in the lattice, we reduce $\mathcal{B}$ using BKZ. For z to appear in the reduced basis, one should at least set Δ such that

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

The vector z we expect to find has norm $||z||_2 \leqslant 2^{m-1}\sqrt{T+1}$. From Inequality (8), one can deduce the value of Δ required to find z 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 $T = 26 \times u$ on average. Moreover, the average value of μ_{ij} is 7 and so on average $\sum \mu_{ij} = 7 \times 26 \times u$. Hence, if we compute Δ_{th} for $u = 3, \dots, 8$, with these values, we obtain $\Delta_{th} \ll 1$.

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 specific sets of parameters the reduced basis contains undesired short vectors, *i.e.*, vectors that are shorter than z. 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

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

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 z. Experimental observations on the position of z in the basis are detailed in Section 5.

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

Differences with the lattice construction given in [11]. Let $\mathcal{B}'$ be the lattice basis constructed in [11]. 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 [11] have a value δ in their construction which corresponds 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 [11]. Moreover, [11] does not mention the systematic short vectors that should appear in the reduced basis.

4 Improving the lattice attack

4.1 Reducing the lattice dimension: the merging technique

In [11], 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. Indeed, the 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 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 [11, Algorithm 3]. Our algorithm modifies directly the sequence $\{\lambda_j\}_{j=1}^{\ell}$, whereas [11] works on the double-and-add chains. This helped us avoid implementation issues such as an index outrun present in [11, Algorithm 3], 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 [11]. We give in Table 3 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.

Table 3: Average dimensions of the lattices after merging.

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

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 too important.

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 after merging, 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 $S_a = \{t \in \mathcal{T} | \max_{1 \leqslant j \leqslant \ell} \mu_j \leqslant a\}$. 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. 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 4, we give the proportion of signatures corresponding to the different preprocessing subsets.

Table 4: Proportion of preprocessing subsets.

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

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

5 Performance analysis

We work with the elliptic curve `secp256k1` but none of the techniques introduced in this paper are limited to this specific elliptic curve. 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 [11], 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 most significant bit (MSB) of k, and in some cases the sign of the second MSB. This extra information leads to the methods A and B presented in [11]. We do not consider this extra information as we want our analysis to remain as general as possible.

We report calculations ran on error-free traces where we evaluate the overall 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 are not able to recover the secret key, and hence the experiment fails. In order to compute the success probability of our attack and the average time of one reduction, we run 5000 experiments for specific sets of parameters using Sage's default BKZ implementation [32]. 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 overall time of our attack is the average time of a single reduction multiplied by the expected number of trials necessary to recover the secret key. For a fixed number of signatures, we can either optimize the overall time of the attack or its probability of success. We report numbers in Tables 5 and 6.

Table 5: Fastest key recovery with respect to the number of signatures.

Number of signatures	Total time	Parameters BKZ	Preprocessing	Δ	Probability of success (%)
3	39 hours	35	S_{11}	$\approx 2^3$	0.2
4	1 hour 17	25	S_{15}	$\approx 2^3$	0.5
5	8 min 20	25	S_{19}	$\approx 2^3$	6.5
6	3 min 55	20	S_{all}	$\approx 2^3$	7
7	2 min 43	20	S_{all}	$\approx 2^3$	17.5
8	2 min 25	20	S_{all}	$\approx 2^3$	29

Table 6: Highest probability of success with respect to the number of signatures.

Number of signatures	Probability of success (%)	Parameters BKZ	Preprocessing	Δ	Total time
3	0.2	35	S_{11}	$\approx 2^3$	39 hours
4	4	35	S_{all}	$\approx 2^3$	25 hours 28
5	20	35	S_{all}	$\approx 2^3$	2 hours 42
6	40	35	S_{all}	$\approx 2^3$	1 hour 04
7	45	35	S_{all}	$\approx 2^3$	2 hours 36
8	45	35	S_{all}	$\approx 2^3$	5 hours 02

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. In the following, we give a detailed analysis for each parameter.

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 overall 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 overall 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.

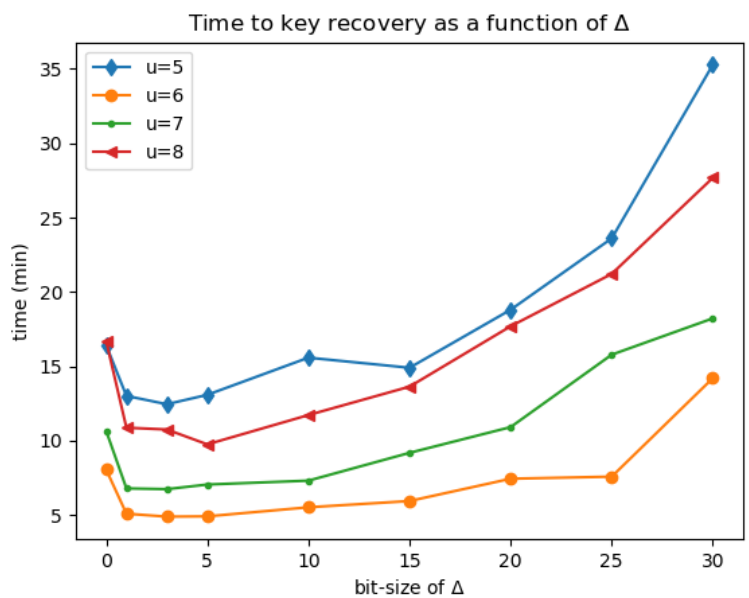

Fig. 1: Analyzing the overall time to recover the secret key as a function of the bitsize of Δ. We report numbers for BKZ-25 and no preprocessing. The optimal value for Δ is around 2^3 except for $u = 8$ where it is 2^5.

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

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[5] with norms smaller than the norm of z. For this reason, the probability of success of the attack decreases, the vector z more likely to be a linear combination of vectors already in the reduced basis. For example, with 7 signatures we find in average z 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, in the case of 4 signatures, using S_{15} lowers the overall time by a factor up to 5.7 compare to S_{all}. For 5 signatures, we gain a factor close to 3 by using either S_{15} or S_{19} instead of S_{all}.

For $u > 5$, using preprocessed traces is less impactful. For large $\varDelta$ such as $\varDelta \approx 2^{25}$, we still note 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 impacful. Details are given in Appendix B.

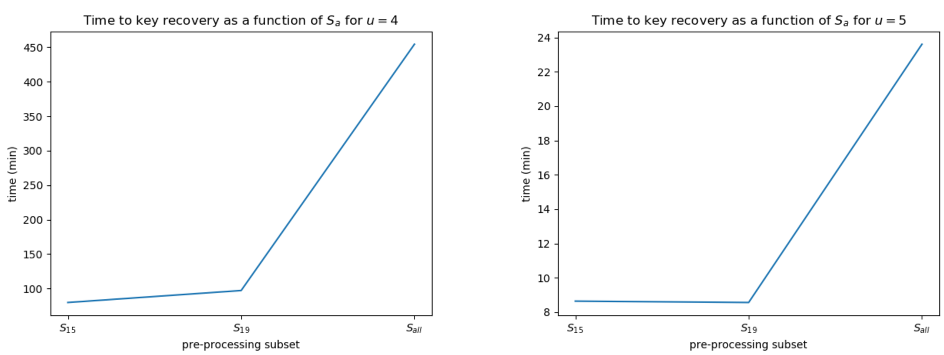

Fig. 2: Analyzing the overall time to recover the secret key as a function of the preprocessing subset for 4 and 5 traces. The other parameters are fixed: $\varDelta \approx 2^{25}$ and BKZ-25.

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

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 reading. 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 5 and Table 6: 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 20 and 25, except for 3 signatures where the probability of success is too low with these parameters. Details are given in Appendix C.

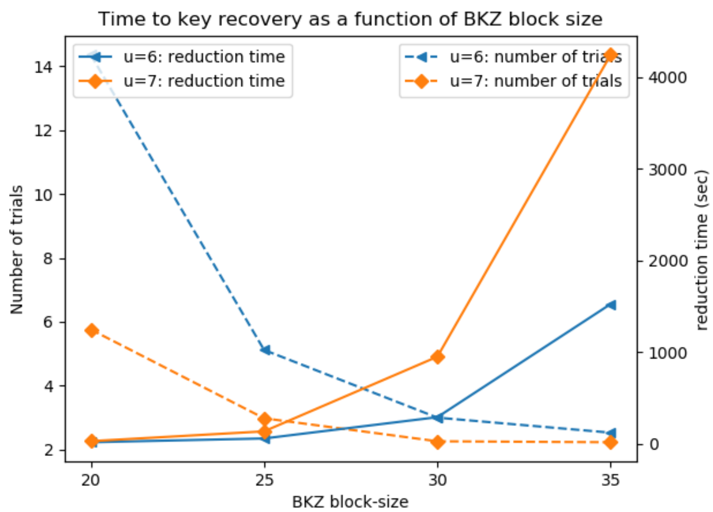

Fig. 3: Analyzing the number of trials to recover the secret key and the reduction time of the lattice as a function of the block-size of BKZ. We consider the cases where $u = 6$ and $u = 7$. The dotted lines correspond to the number of trials, and the continued lines to the reduction time in seconds.

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 [10], the presence of errors in traces was either ignored or preprocessing was done on the traces until an error-free sample was found, see [13, 3]. In [10], 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 [10] 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 [25] and [10] give an analysis of the attacks `FLUSH` + `RELOAD` and `Prime` + `Probe` in real life scenarios.

In [11], 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 a perfect reading of the traces being 56 % as pointed out in [25]. 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 7 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.

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

Number of signatures	Probability of success (%) 0 errors	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	$\ll 1$

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 digits 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 10 in Appendix D.

The other type of error consists of a non-zero digit 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 considered. More details on the probabilities of success of these two types of errors can be seen in Appendix D.

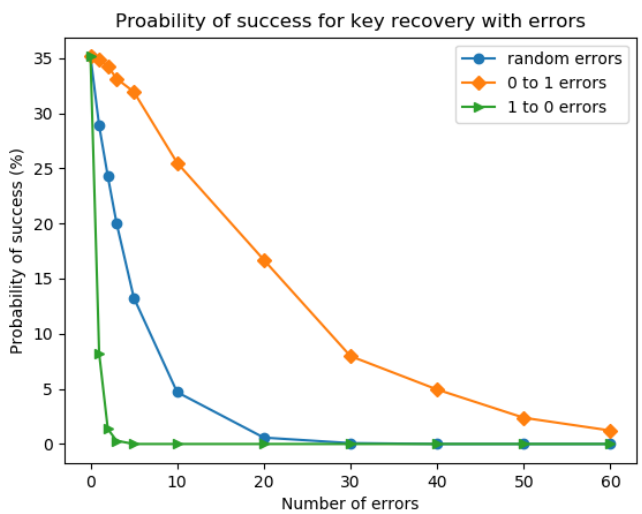

Fig. 4: Probability of success for key recovery with various types of errors when using $u = 8$, BKZ-25, $\Delta \approx 2^3$, and no preprocessing.

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 with the unknowns $(\alpha, d_{1,1}, \cdots, d_{u,l_u})$, it is natural to ask whether this system can be solved using Coppersmith's method for finding small modular roots of integer polynomials. Admittedly, α is of the order of magnitude of q so not so small, but small root means that it is sufficient to know a bound on each variable.

Coppersmith's methods in the case of bivariate polynomials can be expressed as the following theorem [12, Theorem 19.2.1]. It states that a small modular root of a bivariate polynomial can be found as an integer root of other polynomials.

Theorem 1. *Let $F(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ be a polynomial of total degree t. Let $X_1, X_2, q \in \mathbb{N}$ be such that $X_1 \cdot X_2 < q^{1/t-\epsilon}$ for some $0 < \epsilon < 1/t$. Then one can compute in time polynomial in $\log(q)$ and $1/\epsilon > t$ two polynomials $F_1(x_1, x_2)$ and $F_2(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ such that for all $(x_1^{(0)}, x_2^{(0)}) \in \mathbb{Z}^2$ with $|x_1^{(0)}| < X_1, |x_2^{(0)}| < X_2$ and $F(x_1^{(0)}, x_2^{(0)}) \equiv 0 \pmod q$, one has $F_1(x_1^{(0)}, x_2^{(0)}) = F_2(x_1^{(0)}, x_2^{(0)}) = 0$ over $\mathbb{Z}$.*

Theorem 1 is generalized to m variables in [18]. Let $|x_i^{(0)}| < X_i$ for $i = 1, \cdots, m$. In this setting, the condition $X_1 \cdot X_2 < q^{1/t-\epsilon}$ for some $0 < \epsilon < 1/t$ is replaced by $X_1 \cdot X_2 \cdots X_m < q^{1/t-\epsilon}$.

In our setup. We consider the system of $u-1$ modular equations after elimination of α, with $T = \sum_{i=2}^{u} \ell_i$ variables. This allows us to have one less unknown and to avoid having to recover α which would be our largest variable.

We have the following 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. Note that F_i is a linear polynomial and its total degree is $t = 1$.

Let D be a bound on the unknowns d_{ij}, *i.e.*, $|d_{ij}| < D$. The condition in the theorem requires that

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

which means $D < q^{(1-\epsilon)/T}$. When $\epsilon \to 1$, we get that $D < 1$, and when $\epsilon \to 0$, we have $D < q^{1/T}$. If we consider the attack scenario where the number of signatures u belongs to $[3, 8]$, the value of T grows with u and for $u = 3$, the value T is around 150 on average. This results in the condition $D \leqslant 3$. But restricting the bound on the d_{ij} to 3 at best seems too restrictive for the key recovery to be successful. Indeed, it means that the algorithm will miss all the solutions with at least one $3 < d_{i,j} < 2^{\mu_{i,j}}$.

Remark 4. We also considered the equations without elimination, *i.e.*, keeping the variable α. However, this resulted in an even stronger condition on D (we always have $D < 1$) due to the size of α.

Remark 5. The theorem mentioned above is one of the many variations of Coppersmith's method. The proof of the theorem relies on the construction of a lattice whose coefficients correspond to the coefficients of the polynomials for which we want to find a modular root. The idea is to use LLL on this lattice to construct new polynomials with small coefficients, small enough so that the expected modular root is in truth a root of these new polynomials over the integers.

We have tested various constructions for the lattice basis. In particular, we give one of our lattice constructions in the elimination case in Appendix E. However, none of our constructions have allowed us to successfully recover the secret key.

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

1. Acıiçmez, O., Schindler, W.: A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL. In: Malkin, T. (ed.) Topics in Cryptology – CT-RSA 2008. pp. 256–273. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
2. Acıiçmez, O., Schindler, W., Koç, C.K.: Cache based remote timing attack on the AES. In: Proceedings of the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology. pp. 271–286. CT-RSA'07, Springer-Verlag, Berlin, Heidelberg (2006)
3. Angel, J., Rahul, R., Ashokkumar, C., Menezes, B.: DSA signing key recovery with noisy side channels and variable error rates. In: INDOCRYPT. Lecture Notes in Computer Science, vol. 10698, pp. 147–165 (2017)
4. Benger, N., van de Pol, J., Smart, N.P., Yarom, Y.: "Ooh Aah... just a little bit": A small amount of side channel can go a long way. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2014. pp. 75–92. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
5. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 129–142. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
6. Buterin, V.: Ethereum White Paper: A next-generation smart contract and decentralized application platform (2013)
7. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP message format (2007)
8. Chase, B., MacBrough, E.: Analysis of the XRP ledger consensus protocol. arXiv CoRR (2018)

9. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 7073, pp. 1–20. Springer (2011)
10. Dall, F., De Micheli, G., Eisenbarth, T., Genkin, D., Heninger, N., Moghimi, A., Yarom, Y.: 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), https://tches.iacr.org/index.php/TCHES/article/view/879
11. Fan, S., Wang, W., Cheng, Q.: Attacking OpenSSL implementation of ECDSA with a few signatures. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1505–1515. CCS '16 (2016)
12. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press, New York, NY, USA, 1st edn. (2012)
13. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., Yarom, Y.: ECDSA key extraction from mobile devices via nonintrusive physical side channels. In: CCS. pp. 1626–1638 (2016)
14. Gordon, D.M.: A survey of fast exponentiation methods. Journal of Algorithms **27**(1), 129–146 (Apr 1998)
15. Hlaváč, M., Rosa, T.: Extended hidden number problem and its cryptanalytic applications. In: Biham, E., Youssef, A.M. (eds.) Selected Areas in Cryptography. pp. 114–133. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
16. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. Designs, Codes and Cryptography **23**(3), 283–290 (2001). https://doi.org/10.1023/A:1011214926272
17. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security **1**(1), 36–63 (2001). https://doi.org/10.1007/s102070100002
18. Jutla, C.S.: On finding small solutions of modular multivariate polynomial equations. In: Nyberg, K. (ed.) Advances in Cryptology — EUROCRYPT'98. pp. 158–170. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
19. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
20. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**(4), 515–534 (1982)
21. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com (2009)
22. National Institute of Standards and Technology: Digital Signature Standard (DSS) (2013)
23. Nguyen, P., Stehlé, D.: LLL on the average. In: Proceedings of the 7th International Conference on Algorithmic Number Theory. pp. 238–256. ANTS'06, Springer-Verlag, Berlin, Heidelberg (2006)
24. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Designs, Codes and Cryptography **30**(2), 201–217 (Sep 2003)
25. Van de Pol, J., Smart, N.P., Yarom, Y.: Just a little bit more. In: Nyberg, K. (ed.) Topics in Cryptology — CT-RSA 2015. pp. 3–21. Springer International Publishing, Cham (2015)
26. Rankl, W.: Smart Card Applications: Design Models for Using and Programming Smart Cards. Wiley (2007)
27. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3 (RFC 8446) (2018)

28. Schnorr, C.P.: A hierarchy of polynomial time lattice basis reduction algorithms. Theoretical Computer Science **53**(2-3), 201–224 (1987)
29. Schnorr, C.P.: Block reduced lattice bases and successive minima. Combinatorics, Probability & Computing **3**, 507–522 (1994)
30. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming **66**(2), 181–199 (1994)
31. Schwartz, D., Youngs, N., Britto, A.: The Ripple Protocol Consensus Algorithm (2014)
32. The FPLLL development team: FPLLL, a lattice reduction library (2016), https://github.com/fplll/fplll
33. The OpenSSL Project: OpenSSL: The open source toolkit for SSL/TLS (2003), www.openssl.org
34. Vanstone, S.: Responses to NIST's proposals (1992)
35. Wang, W., Fan, S.: Attacking OpenSSL ECDSA with a small amount of side-channel information. Science China Information Sciences **61**(3), 032105 (2017)
36. Yarom, Y., Benger, N.: Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack. IACR Cryptology ePrint Archive **2014**, 140 (2014)
37. Yarom, Y., Falkner, K.: FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In: Proceedings of the 23rd USENIX Conference on Security Symposium. pp. 719–732. SEC'14, USENIX Association, Berkeley, CA, USA (2014)

A Bitsize of $\boldsymbol{\Delta}$ effect over the key recovery overall time

We analyze the effect of the bitsize of Δ. We fix BKZ-25 and use no preprocessing. We average over 5000 experiments. The overall shortest time and the corresponding parameters are written in bold.

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Overall time to key recovery (min)
4	25	S_{all}	0	0.14	31	375
4	25	S_{all}	1	0.16	31	330
4	**25**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**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

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Overall time to key recovery (min)
6	25	S_{all}	0	15.94	77	8
6	25	S_{all}	1	19.96	61	5
6	**25**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**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 overall time

We analyze the effect of the preprocessing. We fix BKZ-25 and $\Delta \approx 2^3, 2^{25}$. We average over 5000 experiments. The overall shortest time and the corresponding parameters are written in bold. For $\Delta = 2^{25}$:

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Overall time to key recovery (min)
4	**25**	$\boldsymbol{S_{11}}$	**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**	$\boldsymbol{S_{19}}$	**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**	$\boldsymbol{S_{19}}$	**25**	**25.12**	**77**	**5**
6	25	S_{all}	25	20.02	91	7

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Overall time to key recovery (min)
7	25	S_{11}	25	3.40	55	27
7	25	S_{15}	25	26.20	151	9
7	**25**	$\boldsymbol{S_{19}}$	**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**	$\boldsymbol{S_{19}}$	**25**	**43.90**	**267**	**10**
8	25	S_{all}	25	36.12	460	21

For $\Delta = 2^3$:

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Overall time to key recovery (min)
4	25	S_{11}	3	0.18	9	89
4	**25**	$\boldsymbol{S_{15}}$	**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**	$\boldsymbol{S_{19}}$	**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**	$\boldsymbol{S_{19}}$	**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**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**3**	**35.14**	**227**	**10**

C BKZ block-size effect over the key recovery overall 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. The overall shortest time and the corresponding parameters are written in bold.

Parameters				Results		
u	BKZ-β	Preprocessing	Δ bitsize	Probability of success (%)	Time of one experiment (sec)	Overall time to key recovery (min)
4	20	S_{all}	3	0	5	0
4	**25**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**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**	$\boldsymbol{S_{all}}$	**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.

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

Number of signatures	Probability of success (%)								
	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	1.80	1.02	0.50
8	35.14	34.92	31.94	25.50	16.70	7.96	4.94	2.48	1.22

Table 9: Error $1 \rightarrow 0$ analysis using BKZ-25, $\Delta \approx 2^3$ and S_{all}.

Number of signatures	Probability of success (%)			
	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

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 10.

Table 10: Errors $0 \rightarrow 1$ analysis with $\Delta \approx 2^3$, S_{all} and increasing block-size.

Number of signatures	Probability of success (%)															
	30 errors				40 errors				50 errors				60 errors			
	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	0.12
7	4.54	6.44	7.32	8.73	1.80	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	2.18

E Lattice construction for Coppersmith's methods

We consider $u-1$ equations given after elimination. We construct the following lattice basis $\mathcal{B}$

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

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

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

Coppersmith's method uses LLL to produce polynomials with integer roots equal to those from the inital modular equations. To do so, it is required for the norms of the vectors in the reduced basis to be smaller than the modulo q, and thus the lattice basis must satisfy $(1.02)^n (\det \mathcal{L})^{1/n} < q$, 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 $u \in [3, 8]$.