Attacking OpenSSL Implementation of ECDSA with a Few Signatures

Shuqin Fan
State Key Laboratory of Cryptology
P.O. Box 5159
Beijing, 100878, P.R.China
fansq@sklc.org

Wenbo Wang
Luoyang University of Foreign Languages
Luoyang, Henan, P.R.China
&
State Key Laboratory of Cryptology
P.O. Box 5159
Beijing, 100878,P.R.China
wangwenbo0305@-sina.com

Qingfeng Cheng
School of Computer Science and Technology
Xidian University
Xi'an, Shanxi, P.R.China
&
Luoyang University of Foreign Languages
Luoyang, Henan, P.R.China
qingfengc2008@sina.com

ABSTRACT

In this work, we give a lattice attack on the ECDSA implementation in the latest version of OpenSSL, which implement the scalar multiplication by windowed Non-Adjacent Form method. We propose a totally different but more efficient method of extracting and utilizing information from the side-channel results, remarkably improving the previous attacks. First, we develop a new efficient method, which can extract almost all information from the side-channel results, obtaining 105.8 bits of information per signature on average for 256-bit ECDSA. Then in order to make the utmost of our extracted information, we translate the problem of recovering secret key to the Extended Hidden Number Problem, which can be solved by lattice reduction algorithms. Finally, we introduce the methods of elimination, merging, most significant digit recovering and enumeration to improve the attack. Our attack is mounted to the **secp256k1** curve, and the result shows that only 4 signatures would be enough to recover the secret key if the FLUSH+RELOAD attack is implemented perfectly without any error,which is much better than the best known result needing at least 13 signatures.

CCS Concepts

•**Security and privacy** → **Digital signatures; Cryptanalysis and other attacks;** *Side-channel analysis and countermeasures;*

Keywords

ECDSA; OpenSSL; lattice attack; windowed Non-Adjacent Form; Extended Hidden Number Problem; FLUSH+RELOAD attack

CCS'16, October 24-28, 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2976749.2978400

1. INTRODUCTION

The Elliptic Curve Digital Signature Algorithm (ECDSA) [14] which was first proposed by Scott Vanstone [27] in 1992, is the most popular signature scheme due to its small key size and high security. It was included in many standards, including the ISO standard, the ANSI standard, the IEEE standard, and FIPS, etc. ECDSA has been widely used in many cases, such as the Austrian Citizen Card – Austrian e-ID [11], the Apple's CommonCrypto framework (as included in iOS versions 7.1.2 through 8.3), OpenSSL[1], etc. The Bitcoin [19], a decentralized electronic currency system, also relies on ECDSA to authenticate transactions.

The mathematical basis for the security of ECDSA is the computational intractability of the elliptic curve discrete logarithm problem (ECDLP). However, most real-world cryptographic vulnerabilities do not stem from a weakness in the hardness assumption, but rather from implementations issues such as side-channel attacks, software bugs or design flaws.

One of the most popular software implementations of ECDSA is the OpenSSL [1] implementation. As a commonly used open-source cryptographic library, OpenSSL has been widely used to implement Secure Sockets Layer (SSL) protocol and Transport Layer Security (TLS) protocol, as well as OpenPGP and other cryptographic standards. In the latest version of OpenSSL (version 1.0.2h, published in May 3, 2016), the scalar multiplication in ECDSA is implemented using the windowed non-adjacent form (wNAF) algorithm [16, 7, 24] by default.

After the seminal work of side-channel attack by Kocher et al. [15], more recent works [22, 2, 4, 25, 5, 30, 13] focus on cache side-channel attacks on software implementations. In 2013, Yarom and Falkner [29] proposed a new method of cache side-channel attack, the FLUSH+RELOAD attack. It targets the Last Level Cache (LLC) so that the attack can be mounted between different cores. The FLUSH+RELOAD attack can be used to observe the execution of ECDSA in OpenSSL, being able to get the sequence of point additions and doubling (denoted as the double-and-add chain) used to executes the scalar multiplication, which can be further utilized to recover the ECDSA secret key. In fact, a small leakage of information on the scalar (the ephemeral key k) in

each signature can be combined to obtain the entire secret key.

Generally speaking, there are mainly three aspects that may affect the attacks on ECDSA implementation: the first one is the implementation of scalar multiplication, which includes the method of wNAF representation, the double-and-add method, the sliding-window method, etc.; the second one is the method of side-channel attack determining what information can be achieved; the third one is the way of extracting and utilizing the achieved information. Take the Flush+Reload attack for example. If the double-and-add method is used, then obtaining the double-and-add chain using the Flush+Reload attack means that the scalar itself can be fully recovered; if the sliding-window method is used, then according to [10], the secret key can be easily broken since the double-and-add chain helps to directly determine some discrete bits of the ephemeral key. For the wNAF representation, as it converts a scalar into a chain of signed digit, it is not easy to obtain direct information on the bits of the scalar from the double-and-add chain due to the existence of negative digits.

In this paper, we focus on the ECDSA implementation of OpenSSL using the wNAF representation, assuming that the Flush+Reload attack is used to obtain the double-and-add chain of the scalar multiplication. Then here comes the question:

Question: How can we take the utmost of the double-and-add chain in order to extract the information of the ephemeral key as much as possible? How should we fully utilize the extracted knowledge so that we can recover the ECDSA secret key using as few signatures as possible?

Analysis: Though it is not easy to obtain direct information of the ephemeral key with the wNAF method, several works give the methods of extracting and utilizing information from the double-and-add chain. In 2014, Benger et al. extracted several least significant bits (LSBs) from the double-and-add chain [3]. But the number of bits extracted from each signature is very small, only average number of 2 bits ($\sum_{i=1}^{\infty} i/2^i \approx 2$) information can be obtained, which results in more than 200 signatures needed to recover a 256-bit secret key.

In 2015, van de Pol et al. derived a more effective way of exacting information from the double-and-add chain [26]. As they stated that, it is able to extract average 47.6 bits per signature, recovering the secret key of the **secp256k1** curve with 13 signatures, assuming a perfect Flush+Reload attack without error. Their method of extracting information relies on the property of some special curves, i.e., the order q of the base point is a pseudo-Mersenne prime, which can be expressed as $2^n - \varepsilon$, where $|\varepsilon| < 2^p$, $p \approx n/2$. They can extract information from consecutive non-zero digits whose positions are between $p+1$ and n, which cover about a half of the double-and-add chain. While according to the ECDSA standards, such curves with order q being a pseudo-Mersenne prime is just a small proportion of all the curves. In the recommended curves of FIPS [8], there are other curves whose order q is a generalized Mersenne prime that cannot be expressed as a pseudo-Mersenne prime, in which case the method of [26] does not work.

Both the above works utilize the extracted information to recover the secret key by a popular lattice method. As in [12, 20, 21, 18], they first use the extracted information to compute triples, each deriving a Hidden Number Problem (HNP) instance, then the secret key can be recovered by solving the HNP, which can be further converted to the closest vector problem (CVP) in a suitable lattice. However, as the author stated, the triples used in [26] are not independent, which contradicts the basic requirement of [21] that the triples are taken uniformly and independently from a distribution that satisfies some conditions. The existence of correlated triples may lead to an error solution, which may decrease the success probability.

Our Answer: We will give a better answer to the above question in this work. Using our new method of extracting information from the double-and-add chain, we are able to obtain as much as 105.8 bits of information per signature for 256-bit ECDSA on average. We believe that our method extract almost all the available information of the double-and-add chain. Moreover, by using the Extended Hidden Number Problem (EHNP) instead of HNP, it seems that we are able to fully utilize our extracted information.

Contribution. In this paper, we attack the ECDSA algorithm implemented by OpenSSL using the wNAF representation with the help of the Flush+Reload cache side-channel attack. We probe into the wNAF representation of the ephemeral key k, developing a totally new way of extracting information from the result of a perfect double-and-add chain. Unlike the previous methods of extracting information from partial double-and-add chain, we take advantage of all positions of digits. We are able to extract as much information as possible, obtaining on average 105.8 bits per signature for 256-bit ECDSA, being 2 times more than that in [26]. This naturally leads to the fact that the number of signatures needed to recover the ECDSA secret key is remarkably reduced. In order to make the utmost of the information we extract, we translate the problem of secret key recovery to the EHNP (instead of the HNP) which can be converted to the approximate shortest vector problem (SVP). Our attack has the following advantages: it does not rely on the property of any special elliptic curve, so the attack can be mounted to all types of curves; each signature is used only once to construct an EHNP instance, thus avoiding the problem of not being independent; all positions of the double-and-add chain are utilized instead of just a proportion of it. We further introduce the methods of elimination, merging, most significant digit recovering and enumeration to substantially increase the success probability of secret key recovery.

We mount our attack to the **secp256k1** curve in this paper – actually it can be applied to all types of curves. The results shows that if the Flush+Reload attack is implemented perfectly without any error, 4 signatures would be enough to recover secret key with probability being about 8%, 5 signatures with probability being about 37.5%, 6 signatures with probability being about 90%, and 7 signatures with probability being about 94%. While considering of the possibility of not conducting a perfect Flush+Reload attack, whose probability is about 42.3% [26], the actual number of signatures needed to obtain 4, 5, 6 and 7 signatures is separately 7, 9, 11 and 13.

In theory, it is possible to recover the 256-bit ECDSA secret key with 3 signatures ($3 \times 105.8 > 256$). Due to the limited capabilities of computation and lattice reduction algorithms, the best result we can get is that 4 signatures are enough to recover the secret key. Comparisons of our attack with the attack of [3] and [26] are presented in Table 1. As

Table 1: Comparison of our results and the results of [3] and [26]

(N_{sig} denotes number of signatures assuming perfect FLUSH+RELOAD attack results are obtained; p denotes the success probability; N_{sig}/p denotes the ratio of the number of signatures and the success probability)

Attack	N_{sig}	p	N_{sig}/p
[3]	200	3.5%	5714.28
	460	92%	900
[26]*	13	54%	24.07
ours*	4	8%	50
	5	37.5%	13.33
	6	90%	6.67
	7	94%	7.45

(*Note: Assuming perfect FLUSH+RELOAD attack results are obtained)

we can see that, both the number of signatures we need to recover a 256-bit ECDSA secret key and the ratio of the number of signatures and the success probability of our attack is much smaller, which indicates that our attack is far more efficient.

Roadmap. The rest of the paper is organized as follows: In Section 2, we briefly introduce the background of ECDSA, OpenSSL implementation using wNAF and the FLUSH+RELOAD side-channel attack, the EHNP is also introduced as it is the basis of our attack. Then in Section 3, our lattice attack method is introduced based on the EHNP and the wNAF representation, with knowledge of perfect side-channel attack results. We further introduce some improvements to our attack, including the methods of elimination, merging, MSD recovering and enumeration. In Section 4, the concrete experiment results are stated, concerning the success probability and consuming time.

2. PRELIMINARIES

In this section we describe the basic knowledge of ECDSA and its implementation in OpenSSL using the wNAF representation, as well as the FLUSH+RELOAD attack. The EHNP is also introduced for it is an important tool of our attack.

2.1 The Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) [14] is the elliptic curve analogue of the Digital Signature Algorithm (DSA) [8], adapting one step of the DSA from the multiplicative group of a finite field to the group of points on an elliptic curve.

Let E be an elliptic curve defined over a finite field $\mathbb{F}_p$ where p is prime and $G \in E$ be a fixed point of a large prime order q. Both G and q are publicly known. The secret key of a signer is an integer $0 < \alpha < q$, and the public key is the point $Q = \alpha G$. Given a hash function H, the ECDSA signature of a message m is computed as follows:

1. Randomly choose an ephemeral key $0 < k < q$.
2. Compute the point $(x, y) = kG$, and let $r \equiv x \bmod q$; if $r = 0$, then go back to the first step.
3. Compute $s = k^{-1}\left(H(m) + r \cdot \alpha\right) \bmod q$; if $s = 0$, then go back to the first step.

Algorithm 1: Conversion to wNAF

Input: Scalar k and window size w
Output: k in the wNAF: e_0, e_1, e_2, $\ldots e_v$

```
1: i = 0
2: while k > 0 do
3:     if k mod 2 = 1 then
4:         e_i = k mod 2^{w+1}
5:         if e_i ⩾ 2^w then
6:             e_i = e_i − 2^{w+1}
7:         end if
8:         k = k − e_i
9:     else
10:        e_i = 0
11:    end if
12:    k = k/2
13:    i = i + 1
14: end while
```

The pair (r, s) is the ECDSA signature of the message m. Given the knowledge of the ephemeral key k and all the known information of (s, r, m), the secret key can be easily recovered by

$$\alpha = r^{-1}(s \cdot k - H(m)) \mod q\,. \tag{1}$$

2.2 The Windowed Non-Adjacent Form Representation

Let us first describe the implementation of scalar multiplication with windowed Non-Adjacent Form (wNAF).

In OpenSSL, if scalar multiplication kG is to be computed, a window size w is firstly chosen (for curve **secp256k1**, $w = 3$), after which precomputation and storage of the points $\{\pm G, \pm 3G, \ldots, \pm(2^w - 1)G\}$ is executed. Then the scalar k is converted to the Non-Adjacent Form (NAF), represented by a sequence of digits e_i, where $e_i \in \{0, \pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$. Every non-zero element is followed by at least w zero values. Algorithm 1 introduces the concrete method for converting a scalar into its Non-Adjacent Form. We denote those non-zero digits among $\{e_i\}$ as $\{k_i\}_{i=1}^{l}$, where l is the number of non-zero digits. Denote the position of each k_i as λ_i, then the scalar k can be rewritten as $k = \sum_{i=1}^{l} k_i \cdot 2^{\lambda_i}$. We call k_l the **most significant digit** (MSD) and k_{l-1} the second MSD. After converting k to the wNAF representation, the scalar multiplication kG is executed as the Algorithm 2 describes. In the actual OpenSSL execution, instead of computing kG, it adds q or $2q$ to the ephemeral key to make sure that k is $\lfloor \log_2 q \rfloor + 1$ bit long. This method can resist the Brumley and Tuveri remote timing attack [5]. In most cases, the multiplication is done as $(k + q)G$.

Note that OpenSSL uses the modified wNAF representation instead of the generalized one as stated in Algorithm 1 to avoid length expansion in some cases and to make exponentiation more efficient. The representation of modified wNAF is very similar to the wNAF. Each non-zero coefficient is followed by at least w zero coefficients, except for the most significant digit which is allowed to violate this condition in some cases. As the use of modified wNAF affects the attack results little, we only consider the case of the wNAF for simplification.

Algorithm 2: OpenSSL implementation of kG using wNAF

Input: Scalar k in the wNAF $e_0, e_1, \ldots, e_v$ and precomputed points $\{\pm G, \pm 3G, \ldots, \pm(2^w - 1)G\}$
Output: kG

```
1: Q = G
2: for i = v, v − 1, ..., 0 do
3:     Q = 2 · Q
4:     if e_i ≠ 0 then
5:         Q = Q + e_i G
6:     end if
7: end for
```

2.3 The Flush+Reload Attack on wNAF Representation

According to Algorithm 2, if the **if-then** block is ran into, digit e_i is non-zero, and vice versa. So if we are able to detect whether the **if-then** block is executed or not during each loop of **for-do**, we can determine whether e_i is zero or not. The Flush+Reload attack is just a perfect tool for doing this.

The Flush+Reload attack was first proposed by Yarom and Falkner [29]. Unlike most of the other side-channel attacks that target the L1 cache level [4, 25, 30] or the branch prediction buffer [2], it targets LLC, which is shared between cores, thus making it possible to mount the attack between different cores. Using the Flush+Reload attack, Yarom and Falkner snooped on the square-and-multiply exponentiation in the GnuPG implementation of RSA and being able to recover the RSA secret key [29]. They also use the Flush+Reload attack to recover 95% of the ephemeral key of ECDSA which is implemented by OpenSSL over characteristic two fields using the Montgomery ladder for scalar multiplication [28], which means that only one signature would be enough to fully recover the ECDSA secret key.

The Flush+Reload attack can be used to obtain the double-and-add chain of the OpenSSL implementation of kG using wNAF. It uses a spy program to monitor cache hits/misses so that we can determine whether the program has ran into the **if-then** block of Algorithm 2. Denote "A" for an add operation in the **if-then** block, and "D" for a double operation. Under the assumption of a perfect side-channel, we may obtain a double-and-add chain of information like this (assume that the window size $w = 3$, and the sequence is written from the higher index to the lower index):

"ADDDADDDDDDAD . . . . . DDDADD"

In fact, a double operation is done every time before the **if-then** block is executed, so there is a "D" right before each "A", but we omit these "D"s for simplification.

From the wNAF representation rule introduced in Section 2.2, though we can decide a digit is zero or not according to the double-and-add chain of a scalar multiplication, it is impossible to determine the sign of the non-zero digit. If the non-zero digit is negative, there exists carries into the next bits. So it is not easy to get direct information on the value of bits from the double-and-add chain in most cases but some positions of non-zero digits.

2.4 The Extended Hidden Number Problem

The Extended Hidden Number Problem (EHNP) introduced in [10] is originally used to recover the secret key of a DSA signature, given some discrete leaked bits of the ephemeral key.

Let N be a prime number. Given u congruences

$$\beta_i x + \sum_{j=1}^{l_i} a_{i,j} k_{i,j} \equiv c_i \mod N, 1 \le i \le u ,$$

where $k_{i,j}$ and x are unknown variables satisfying $0 \le k_{i,j} \le 2^{\varepsilon_{i,j}}$ and $0 < x < N$, β_i, $a_{i,j}$, c_i, l_i and $\varepsilon_{i,j}$ are all known. The EHNP is to find the unknown x (the hidden number) which satisfies the conditions above.

For the OpenSSL implementation of DSA signature with the method of sliding window exponentiation algorithm (only available for OpenSSL versions up to 0.9.7g), using the "squaring-and-multiplication" chain observed from the side-channel attack, it is able to get many discrete bits of the ephemeral key directly. Then the problem of recovering the secret key can be transformed to the EHNP, which can be further converted to the approximate SVP in some suitable lattice.

3. ATTACKING ECDSA

As introduced in Section 2.3, it is commonly difficult to extract information directly from the double-and-add chain. In this section, we propose a totally new way of deriving information from the double-and-add chain, being able to extract on average 105.8 bits of the ephemeral key per signature for ECDSA with 256-bit secret key. After that, we give an efficient way of utilizing the gained information, translating the problem of recovering ECDSA secret key to the EHNP, which can be further transformed to the problem of solving approximate SVP in some lattice using the lattice reduction algorithms.

Assuming we get the entire double-and-add chain of the scalar multiplication without error after a perfect Flush+Reload attack. Suppose in the double-and-add chain, there are l number of "A"s, whose positions are separately λ_i ($1 \leqslant i \leqslant l$) counted from the lower index. Then we have $k = \sum_{i=1}^{l} k_i 2^{\lambda_i}$, where $k_i \in \{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$ is unknown and its position in the wNAF representation $\{e_j\}_{j=1}^{v}$ of k is just λ_i.

3.1 Extracting Information

Since the i-th non-zero digit k_i of the ephemeral key is always an odd number, we can rewrite it as $k_i = 1 + 2 \cdot k'_i$, where $-2^{w-1} \leqslant k'_i \leqslant 2^{w-1} - 1$. Let $d_i = k'_i + 2^{w-1}$, then $0 \leqslant d_i \leqslant 2^w - 1$. So the ephemeral key k can be rewritten as

$$\begin{aligned} k &= \sum_{i=1}^{l} k_i \cdot 2^{\lambda_i} = \sum_{i=1}^{l} (1 + 2 \cdot k'_i) \cdot 2^{\lambda_i} \\ &= \overline{k} + \sum_{i=1}^{l} d_i \cdot 2^{\lambda_i + 1} , \end{aligned} \tag{2}$$

where $\overline{k} = \sum_{i=1}^{l} 2^{\lambda_i} - \sum_{i=1}^{l} 2^{\lambda_i + w}$.

It is known that there are approximately $(\lfloor \log_2 q \rfloor + 1)/(w+2) - 1 = 50.4$ non-zero digits in a double-and-add chain of a 257-bit[1] nonce $k+q$ when $w = 3$.

From Equation (2), since $0 \leqslant d_i \leqslant 2^w - 1$, there are approximately $50.4w = 151.2$ bits being unknown, which means we have on average $(\lfloor \log_2 q \rfloor + 1) - 50.4w = 105.8$ bits information. In theory, three signatures would be enough to recover the 256-bit secret key, as $3 \times 105.8 = 317.4 > 256$.

What's needed to be added is that our method of extracting information does not rely on any special property of elliptic curves, we can extract on average 105.8 bits of information per signature for any type of 256-bit elliptic curve.

3.2 Basic Attack

From Equation (1) we have $\alpha r - sk + H(m) \equiv 0 \bmod q$. Substituting k by Equation (2), we can get that there exists an $h \in \mathbb{Z}$ such that

$$\alpha r - \sum_{j=1}^{l} (2^{\lambda_j + 1} \cdot s) d_j - \big(s\overline{k} - H(m)\big) + hq = 0\,. \quad (3)$$

The values of $0 < \alpha < q$, $0 \leqslant d_j \leqslant 2^w - 1$ and h are unknown, and others are known.

Suppose we successfully get u signature pairs (r_i, s_i) of message m_i $(1 \leqslant i \leqslant u)$ using the same secret key α. From Equation (3) we have the following u equations:

$$\begin{cases} \alpha r_1 - \sum_{j=1}^{l_1} (2^{\lambda_{1,j}+1} \cdot s_1) d_{1,j} - \big(s_1\overline{k_1} - H(m_1)\big) + h_1 q = 0 \\ \qquad\qquad \cdots\cdots \\ \alpha r_i - \sum_{j=1}^{l_i} (2^{\lambda_{i,j}+1} \cdot s_i) d_{i,j} - \big(s_i\overline{k_i} - H(m_i)\big) + h_i q = 0 \\ \qquad\qquad \cdots\cdots \\ \alpha r_u - \sum_{j=1}^{l_u} (2^{\lambda_{u,j}+1} \cdot s_u) d_{u,j} - \big(s_u\overline{k_u} - H(m_u)\big) + h_u q = 0 \end{cases} \quad (4)$$

where l_i is the number of non-zero digits of the i-th ephemeral key, $\lambda_{i,j}$ is the position of the j-th non-zero digit $k_{i,j}$ of the i-th ephemeral key, $\overline{k_i} = \sum_{j=1}^{l_i} 2^{\lambda_{i,j}} - \sum_{j=1}^{l_i} 2^{\lambda_{i,j}+w}$ and $d_{i,j} = (k_{i,j} - 1)/2 + 2^{w-1}$. The values of α, $d_{i,j}$ and h_i are unknown.

We denote this EHNP instance as ECDSA-EHNP, which can be converted to an approximate SVP using the method introduced in [10]. For $1 \leqslant i \leqslant u$ and $1 \leqslant j \leqslant l_i$, let $\beta_i = s_i\overline{k_i} - H(m_i) \mod q$, $\rho_{i,j} = 2^{\lambda_{i,j}+1} \cdot s_i \bmod q$, and $\mu_{i,j}$ be the value that $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}} - 1$, which equals to w. We use Equation (4) to construct a lattice L spanned by the matrix M defined by Equation (5), where δ is a parameter with a proper value.

[1] As introduced previously, in the actual OpenSSL execution of ECDSA, $(k+q)G$ is done in most cases, we assume the wNAF method is applied with $(k+q)G$. And the double-and-add chain is obtained from a scalar of $\lfloor \log_2 q \rfloor + 1 = 256 + 1 = 257$ bits of a 256-bit ECDSA

In the lattice $L = L(M)$, there exists a vector

$$\begin{aligned} \vec{w} &= (h_1, \cdots, h_u, \alpha, d_{1,1}, \cdots, d_{1,l_1}, \cdots, d_{u,1}, \cdots, d_{u,l_u}, -1) \cdot M \\ &= \Big(0, \cdots, 0, \frac{\alpha}{q}\delta - \frac{\delta}{2}, \cdots, \frac{d_{1,1}}{2^{\mu_{1,1}}}\delta - \frac{\delta}{2}, \cdots, \frac{d_{1,l_1}}{2^{\mu_{1,l_1}}}\delta - \frac{\delta}{2}, \cdots, \\ &\qquad \frac{d_{u,1}}{2^{\mu_{u,1}}}\delta - \frac{\delta}{2}, \cdots, \frac{d_{u,l_u}}{2^{\mu_{u,l_u}}}\delta - \frac{\delta}{2}, -\frac{\delta}{2}\Big) \in L \end{aligned}$$

whose Euclidean norm satisfies that $\|\vec{w}\| \leqslant \frac{\delta}{2}\sqrt{T+2}$, where $T = \sum_{i=1}^{u} l_i$.

Note that there is another vector that is quite short, say $\vec{v} = (-r_1, \cdots, -r_u, q, 0, \cdots, 0) \cdot M = (0, \cdots, 0, \delta, 0, \cdots, 0) \in L$, which is likely to be the first vector of the reduced lattice basis.

The lattice determinant is $|L| = q^{u-1} \cdot \delta^{T+2}/2^{U+1}$, where $U = \sum_{i,j} \mu_{i,j}$, and the dimension of L is $n = T + u + 2$. As we can see, if the value of δ is appropriately chosen, vector $\vec{w}$ is very likely to be a short vector of L. So we can use the lattice reduction algorithm like LLL [17] or BKZ [23] to get the vector $\vec{w}$, thus recovering the secret key α.

Choice of δ.

As we state above, we need to set the value of δ appropriately. A natural question is how to choose δ so that we can recover the secrete key.

Actually, the value of δ is to adjust the target vector $\vec{w}$ to be a short vector concerning the lattice M so that the lattice reduction algorithm can find it. Thus on one hand, we need the target vector $\vec{w}$ to be short, while on the other hand, the target vector can not be too short to be achieved by the lattice reduction algorithm such as LLL and BKZ.

First, we need to ensure the target vector $\vec{w}$ to be relatively short. Consider the Gaussian heuristic to estimate the shortest vector in lattice, we have

$$\lambda_1(L) \approx GH(L) \approx \sqrt{\frac{n}{2\pi \mathrm{e}}} |L|^{1/n}\,.$$

So if the ratio $\|\vec{w}\|/GH(L)$ is relatively small, $\vec{w}$ may be relatively short.

While in the actual experiments, we need to adjust the value of δ so that $\vec{w}$ is within the ability of the lattice reduction algorithm. In [9], Gamma and Nguyen gave a picture of the actual behavior of lattice reduction algorithms by experiments. For example, the length of the first vector output by the reduction algorithm BKZ with blocksize 20 is approximately $(1.0128)^n |L|^{1/n}$. So we choose the value of δ such that the length of target vector $\|\vec{w}\| \approx (1.0128)^n |L|^{1/n}$ and make a little adjustment to it to increase the success probability if we use BKZ-20.

Note that if the dimension of the lattice is very low($\leqslant 70$), we need not to consider the ability of the lattice reduction algorithm, since it behaves just like a SVP solver [9].

3.3 Analysis of the Basic Attack

Assume our attack is applied to the curve **secp256k1**. According to Section 3.1, we can obtain on average 105.8 bits of information per signature. So in theory, three signatures would be enough to recover the 256-bit secret key.

We mount the attack with signatures being 3 to 7 separately using the BKZ-20 algorithm, but the results turn out to be very disappointing. Neither three signatures nor four signatures can succeed. For five signatures, we only succeed once among 100 times of attacks, i.e., the success probability

$$M = \begin{pmatrix}
q & & & & & & & & & & & \\
& \ddots & & & & & & & & & & \\
& & q & & & & & & & & & \\
r_1 & \dots & r_u & \delta/q & & & & & & & & \\
\rho_{1,1} & & & & \delta/2^{\mu_{1,1}} & & & & & & & \\
\vdots & & & & & \ddots & & & & & & \\
\rho_{1,l_1} & & & & & & \delta/2^{\mu_{1,l_1}} & & & & & \\
& \ddots & & & & & & \ddots & & & & \\
& & \rho_{u,1} & & & & & & \delta/2^{\mu_{u,1}} & & & \\
& & \vdots & & & & & & & \ddots & & \\
& & \rho_{u,l_u} & & & & & & & & \delta/2^{\mu_{u,l_u}} & \\
\beta_1 & \dots & \beta_u & \delta/2 & \delta/2 & \dots & \delta/2 & \dots & \delta/2 & \dots & \delta/2 & \delta/2
\end{pmatrix} \tag{5}$$

is 1%. When the attack is mounted to six and seven signatures, the dimension of the lattice becomes too large for the lattice reduction algorithm to execute, let alone obtain the right solution. As there are average 50.4 non-zero digits on average in the wNAF representation of each k, the dimension of the lattice is $50.4u + u + 2 = 51.4u + 2$, which means that, for $u = 3, 4, 5, 6, \dots$, the dimension of the lattice is separately 157, 208, 259, 311, ... For $u \geqslant 6$, the dimension is larger than 300, which makes it relatively hard for the LLL algorithm or the BKZ algorithm to find a short vector, meanwhile it takes a relatively long time to execute a lattice reduction algorithm.

4. IMPROVEMENTS

We seek for solutions to make the lattice reduction algorithm easier to find the target vector and to increase the success probability. One direct way is to reduce the lattice dimension. In order to do this, we use the method of elimination and merging to reduce the number of unknown variables.

Furthermore, we carefully exploit the double-and-add chain to recover the MSD and determine whether the second MSD is positive or negative in some cases, which can be used to not only decrease the dimension of lattice, but also increase the determinant of the lattice. At last, we enumerate the MSDs which can not be recovered, so we can further decrease the dimension, thus improve the success probability with the sacrifice of a little more time.

In most of our improvements, we can decrease the ratio $\|\vec{w}\|/GH(L)$ so that the key recovery can be made easier.

4.1 Reducing Lattice Dimension

There are mainly two ways to reduce the dimension of the lattice. The first one is the method of elimination, and the second one is to reduce the number of non-zero digits of ephemeral key k, which we name as the method of merging.

4.1.1 Elimination

As observed from Equation (4), the secret key α can be eliminated to establish a set of new equations, thus reducing the lattice dimension of L.

Denote the i-th equation in simultaneous Equations (4) as E_i. For $2 \leqslant i \leqslant u$, we compute $r_1 E_i - r_i E_1$ and get the following equations:

$$\sum_{j=1}^{l_1} (2^{\lambda_{1,j}+1} s_1 r_i) d_{1,j} - \sum_{j=1}^{l_i} (2^{\lambda_{i,j}+1} s_i r_1) d_{i,j} - \gamma_i + t_i q = 0,$$

where $\gamma_i = r_1(s_i\overline{k_i} - H(m_i)) - r_i(s_1\overline{k_1} - H(m_1)) \mod q$. $0 \leqslant d_{1,j}, d_{i,j} \leqslant 2^{\mu_{i,j}} - 1$ and t_i are unknown. Let $\tau_{j,i} = 2^{\lambda_{1,j}+1} s_1 r_i \mod q$ $(1 \leqslant j \leqslant l_1)$ and $\sigma_{i,j} = -2^{\lambda_{i,j}+1} s_i r_1 \mod q$ $(1 \leqslant j \leqslant l_i)$. As before, we can construct a lattice $L' = L(M')$ spanned by the basis matrix M', which is defined by Equation (6), where δ is again a parameter with an appropriate value.

In lattice $L' = L(M')$, there exists a vector

$$\begin{aligned}
\vec{w}' &= (t_2, \cdots, t_u, d_{1,1}, \cdots, d_{1,l_1}, \cdots, d_{u,1}, \cdots, d_{u,l_u}, -1) \cdot M' \\
&= \Big(0, \cdots, 0, \frac{d_{1,1}}{2^{\mu_{1,1}}}\delta - \frac{\delta}{2}, \cdots, \frac{d_{1,l_1}}{2^{\mu_{1,l_1}}}\delta - \frac{\delta}{2}, \cdots, \frac{d_{u,1}}{2^{\mu_{u,1}}}\delta - \frac{\delta}{2}, \\
&\qquad \cdots, \frac{d_{u,l_u}}{2^{\mu_{u,l_u}}}\delta - \frac{\delta}{2}, -\frac{\delta}{2}\Big) \in L' .
\end{aligned}$$

Its Euclidean norm satisfies that $\|\vec{w}'\| \leqslant \frac{\delta}{2}\sqrt{T+1}$, where $T = \sum_{i=1}^{u} l_i$.

The dimension of lattice $L' = L(M')$ is $n' = T + u$, 2 dimensions less than that of L, while the determinant of L' being $|L'| = q^{u-1} \cdot \delta^{T+1}/2^{U+1}$, $1/\delta$ times of that of lattice L. Since δ is always a very small number, the method of elimination reduces the dimension of the lattice and the length of the target vector, meanwhile increase the determinant of the lattice. As a result, the ratio of $\|\vec{w}\|/GH(L)$ is reduced, which makes it easier for the lattice reduction algorithms to find the vector $\vec{w}'$. On the other hand, the reduction of the lattice dimension leads to a more efficient execution of the lattice reduction algorithm.

Note that, unlike the case of the basic attack that exists a very short vector $\vec{v} = (0, \cdots, 0, \delta, 0, \cdots, 0)$, there is no similar vector in the lattice L' after the elimination, so the target vector $\vec{w}'$ is likely to be the first vector of a reduced lattice basis.

4.1.2 Merging

The method of merging aims to reduce the number of non-zero digits l of each ephemeral key k in Equation (2), meanwhile keeping the number of unknown bits as small as possible. In the language of lattice, it correspondingly reduces the dimension of lattice L or L' and keeping the determinant as large as possible.

$$M' = \begin{pmatrix}
q & & & & & & & & & & & & & & \\
 & \ddots & & & & & & & & & & & & & \\
 & & q & & & & & & & & & & & & \\
\tau_{1,2} & \dots & \tau_{1,u} & \delta/2^{\mu_{1,1}} & & & & & & & & & & & \\
\vdots & & \vdots & & \ddots & & & & & & & & & & \\
\tau_{l_1,2} & \dots & \tau_{l_1,u} & & & \delta/2^{\mu_{1,l_1}} & & & & & & & & & \\
\sigma_{2,1} & & & & & & \delta/2^{\mu_{2,1}} & & & & & & & & \\
\vdots & & & & & & & \ddots & & & & & & & \\
\sigma_{2,l_2} & & & & & & & & \delta/2^{\mu_{2,l_2}} & & & & & & \\
 & \ddots & & & & & & & & \ddots & & & & & \\
 & & \sigma_{u,1} & & & & & & & & \delta/2^{\mu_{u,1}} & & & & \\
 & & \vdots & & & & & & & & \ddots & & & & \\
 & & \sigma_{u,l_u} & & & & & & & & & & & \delta/2^{\mu_{u,l_u}} & \\
\gamma_2 & \dots & \gamma_u & \delta/2 & \dots & \delta/2 & \delta/2 & \dots & \delta/2 & \dots & \delta/2 & \dots & \delta/2 & \delta/2 &
\end{pmatrix} \tag{6}$$

The core idea is to merge two or more consecutive non-zero digits as one new non-zero digit. The distance of two adjacent non-zero digits in the wNAF form of k can be computed as $\lambda_{i+1} - \lambda_i$, where $\lambda_{i+1} - \lambda_i \geqslant w+1$ $(1 \leqslant i \leqslant l-1)$. From Section 3.1, d_i has a one-to-one correspondence with the non-zero digit k_i, the merging of digit d_i is equivalent to the merging of k_i, so we only consider the merging of digit d_i. The position of d_i is defined to be the same with that of k_i.

THEOREM 1. *For $h \geqslant 1$, suppose $h+1$ consecutive digits $d_i, ..., d_{i+h}$ are merged as a new digit d'_i. Then we have $0 \leqslant d'_i \leqslant 2^{\mu_i} - 1$, where $\mu_i = \lambda_{i+h} - \lambda_i + w$ is defined as the window size of d'_i.*

PROOF. Suppose that $h+1$ consecutive digits are merged as one, i.e., $d'_i = d_i + 2^{\lambda_{i+1}-\lambda_i} d_{i+1} + 2^{\lambda_{i+2}-\lambda_i} d_{i+2} + \cdots + 2^{\lambda_{i+h}-\lambda_i} d_{i+h}$.

We can easily obtain the following inequalities:

$$\begin{cases}
0 \leqslant d_i \leqslant 2^w - 1 \\
0 \leqslant 2^{\lambda_{i+1}-\lambda_i} d_{i+1} \leqslant 2^{\lambda_{i+1}-\lambda_i+w} - 2^{\lambda_{i+1}-\lambda_i} \\
0 \leqslant 2^{\lambda_{i+2}-\lambda_i} d_{i+2} \leqslant 2^{\lambda_{i+2}-\lambda_i+w} - 2^{\lambda_{i+2}-\lambda_i} \\
\cdots\cdots \\
0 \leqslant 2^{\lambda_{i+h}-\lambda_i} d_{i+h} \leqslant 2^{\lambda_{i+h}-\lambda_i+w} - 2^{\lambda_{i+h}-\lambda_i}
\end{cases}$$

Adding the above inequalities gives $0 \leqslant d'_i \leqslant 2^{\lambda_{i+h}-\lambda_i+w} + (2^{\lambda_{i+h-1}-\lambda_i+w} - 2^{\lambda_{i+h}-\lambda_i}) + \cdots + (2^w - 2^{\lambda_{i+1}-\lambda_i}) - 1 \leqslant 2^{\lambda_{i+h}-\lambda_i+w} - 1$. This finishes the proof. □

Theorem 1 indicates that if $h+1$ consecutive digits $d_i, ..., d_{i+h}$ are merged, the unknown bits increase from $(h+1)w$ to $\lambda_{i+h} - \lambda_i + w$. It is easy to check that $\lambda_{i+h} - \lambda_i + w = w + \sum_{j=1}^{h} (\lambda_{i+j} - \lambda_{i+j-1}) \geqslant h(w+1) + w$ and the equality holds if only if for all $1 \leqslant j \leqslant h, \lambda_{i+j} - \lambda_{i+j-1} = w+1$. Heuristically, from Equation (4) and (5), the smaller the number of unknown bits is, the larger the determinant of corresponding lattice is, and the easier we can recover the secret key. So those consecutive non-zero digits such that the distance of each adjacent two non-zero digits is $w+1$ are preferred to be merged. In this case, the number of unknown bits increase from $(h+1)w$ to $(h+1)w+h$. [2] In other word, the number of non-zero digits is decreased by h at the cost of h-bit information loss, which equivalently means the number of non-zero digits is decreased by 1 at the cost of 1-bit information loss. This is the best we can achieve.

From the above analysis, we propose a strategy of merging, in which only the consecutive non-zero digits such that the distance of each two adjacent non-zero digits is $w+1$ are merged. The concrete method of merging is introduced in Algorithm 3. By this strategy, we can reduce the number of non-zero digits to about a half meanwhile keep the number of unknown bits as small as possible. Here we give an illustrative example.

We generate a 257-bit ephemeral key k using OpenSSL ($w = 3$), and get its wNAF form like this (we only list the positions of non-zero digits):

3 7 12 18 22 28 34 39 43 48 52 56 64 69
73 77 82 88 92 96 100 106 112 117 122 126
131 135 139 144 150 154 161 165 169 173 177
181 185 189 194 198 202 207 214 219 226 233
238 244 248 252 257

We do the merging as follows:

3 7 | 12 | 18 22 | 28 | 34 | 39 43 | 48 52 56 | 64 | 69 73 77
7bits 3bits 7bits 3bits 3bits 7bits 11bits 3bits 11bits

82 | 88 92 96 100 | 106 | 112 | 117 | 122 126 | 131 135 139
3bits 15bits 3bits 3bits 3bits 7bits 11bits

144 | 150 154 | 161 165 169 173 177 181 185 189
3bits 7bits 31bits

194 198 202 | 207 | 214 | 219 | 226 | 233 | 238 | 244 248 252 | 257
11bits 3bits 3bits 3bits 3bits 3bits 3bits 11bits 3bits

[2] In fact, there are h bits among the $(h+1)w+h$ unknown bits being known, but we just do not know how to utilize it.

Algorithm 3: the Merging algorithm

Input: the double-and-add chain of the scalar multiplication of k, denoted as $SeqAD$, whose length is n

Output: the merged double-and-add chain of the scalar multiplication of k, denoted as $SeqMer$

```
1: i = 0
2: while i < n do
3:     numD = 0
4:     if SeqAD[i] = "A" then
5:         SeqMer[i] = SeqAD[i]
6:         i++
7:         while SeqAD[i] ≠ "A" do
8:             numD++
9:             SeqMer[i] = SeqAD[i]
10:            i++
11:        end while
12:        if numD = w then
13:            SeqMer[i] = "D"  // merge the "A" of the chain if there are
                                   w "D" before it, the merging is
                                   done by replacing "A" with "D"
14:        end if
15:    else
16:        SeqMer[i] ="D"
17:        i++
18:    end if
19: end while
```

Below is the positions of the merged non-zero digits and the corresponding window size.

3	12	18	28	34	39	48	64	69	82	88	106
7bits	3bits	7bits	3bits	3bits	7bits	11bits	3bits	11bits	3bits	15bits	3bits

112	117	122	131	144	150	161	194	207	214	219	226
3bits	3bits	7bits	11bits	7bits	7bits	31bits	11bits	3bits	3bits	3bits	3bits

233	238	244	257
3bits	3bits	11bits	3bits

Let l' be the number of non-zero digits after merging and for $1 \leqslant i \leqslant l'$, d'_i be the new merged digit, λ'_i be the position of d'_i (which is also the location of the new merged non-zero digit). From Equation (2) we have $k = \overline{k} + \sum_{i=1}^{l'} d'_i \cdot 2^{\lambda'_i+1}$, where $0 \leqslant d'_i \leqslant 2^{\mu_i} - 1$, μ_i is the current window size of d'_i. As stated in Section 3.1, the average number of non-zero digits in Equation (2) is 50.4. Roughly speaking, in half of the cases, the distance between two consecutive non-zero digits is $w + 1$, which means the number of non-zero digits after merging is about half of that before merging. We pick 800 random ephemeral keys for test (the **secp256k1** curve). The results show that the average number of non-zero digits after merging is 26.52 per signature, correspondingly the lattice dimension is reduced to $26.52u + u + 2 = 27.52u + 2$. Comparisons of the lattice dimensions before and after merging are listed in Table 2.

4.2 Recovering the MSDs

Intuitively, the more information of the ephemeral key k we have, the higher success probability of recovering the secret key will be. We seek for extracting more information from the double-and-add chain. We find that in some cases, the MSD of k and the sign of the second MSD can be recovered.

Table 2: Dimension comparisons before and after merging

Number of signatures	Lattice dimension of the basic method	Lattice dimension after merging
4	208	113
5	259	140
6	311	168
7	362	195

As observed that, the length of the double-and-add chain of a scalar (denoted as L_{AD}) does not always equal to the length of the binary string of the scalar (denoted as L_0). Indeed, there exist three cases that $L_{\mathrm{AD}} < L_0$, $L_{\mathrm{AD}} > L_0$ and $L_{\mathrm{AD}} = L_0$. When $L_{\mathrm{AD}} > L_0$ or $L_{\mathrm{AD}} = L_0$, the MSD of k and the sign of the second MSD can be recovered. As can be easily seen that $\lambda_l = L_{\mathrm{AD}} - 1$ and $2^{L_0-1} \leqslant k \leqslant 2^{L_0} - 1$.

We first introduce a lemma.

LEMMA 1. *For integer m satisfying $1 \leqslant m \leqslant l - 1$, we have*

$$|\sum_{i=1}^{m} k_i \cdot 2^{\lambda_i}| < 2^{\lambda_{m+1}-1}. \tag{7}$$

PROOF. From the fact that the non-zero digit k_i satisfies $-2^w < k_i < 2^w$ $(1 \leqslant i \leqslant l)$ and $\lambda_{j+1} - \lambda_j \geqslant w + 1$ $(1 \leqslant j \leqslant l - 1)$ we can easily prove Inequality (7) by induction. For $m = 1$, it is easily check $|k_1 \cdot 2^{\lambda_1}| < 2^{w+\lambda_1} \leqslant 2^{\lambda_2-1}$. Assume Inequality (7) holds for any $1 \leqslant m - 1 \leqslant l - 2$, i.e., $|\sum_{i=1}^{m-1} k_i \cdot 2^{\lambda_i}| < 2^{\lambda_m-1}$, we prove it also holds for m. In fact, $|\sum_{i=1}^{m} k_i \cdot 2^{\lambda_i}| = |k_m \cdot 2^{\lambda_m} + \sum_{i=1}^{m-1} k_i \cdot 2^{\lambda_i}|$ $\leqslant (2^w - 1) \cdot 2^{\lambda_m} + 2^{\lambda_m-1} < 2^{w+\lambda_m} \leqslant 2^{\lambda_{m+1}-1}$. This finishes the proof. □

From Lemma 1, we can easily get $k_l > 0$. Since if $k_l < 0$, $k < k_l \cdot 2^{\lambda_l} + 2^{\lambda_l - 1} = (1 + 2k_l)2^{\lambda_l - 1} < 0$, which is obviously a contradiction.

We have the following propositions.

PROPOSITION 1. *If $L_{\mathrm{AD}} > L_0$, then k_{l-1} is negative and the MSD of k is 1.*

PROOF. The condition that $L_{\mathrm{AD}} > L_0$ indicates that $k \leqslant 2^{\lambda_l} - 1$. On the other hand, k can be rewritten as $k = k_l \cdot 2^{\lambda_l} + \sum_{i=1}^{l-1} k_i \cdot 2^{\lambda_i}$. If $k_l \neq 1$, then $k_l \geqslant 3$. From Lemma 1, we have $k \geqslant 3 \cdot 2^{\lambda_l} + \sum_{i=1}^{l-1} k_i \cdot 2^{\lambda_i} > 3 \cdot 2^{\lambda_l} - 2^{\lambda_l - 1} > 2^{\lambda_l}$, which contradict the fact $k \leqslant 2^{\lambda_l} - 1$. So we must have $k_l = 1$.

It remains to prove that k_{l-1} is negative. Suppose $k_{l-1} \geqslant 1$ is positive. Rewrite $k = 2^{\lambda_l} + k_{l-1} \cdot 2^{\lambda_{l-1}} + \sum_{i=1}^{l-2} k_i \cdot 2^{\lambda_i}$, we can easily get that $k > 2^{\lambda_l} + 2^{\lambda_{l-1}} - 2^{\lambda_{l-1}-1} > 2^{\lambda_l}$ by Lemma 1, which contradict the fact $k \leqslant 2^{\lambda_l} - 1$ again. So we have k_{l-1} is negative. This finishes the proof. □

When $L_{\mathrm{AD}} = L_0$, we can also determine the value of MSD and the sign of the second of MSD.

PROPOSITION 2. *If $L_{\mathrm{AD}} = L_0$, then k_{l-1} is positive and the MSD of k is 1.*[3]

PROOF. The condition that $L_{\mathrm{AD}} = L_0$ indicates that $2^{\lambda_l} \leqslant k \leqslant 2^{\lambda_l + 1} - 1$.

Suppose $k_l \neq 1$, then $k_l \geqslant 3$. We have $k = k_l \cdot 2^{\lambda_l} + \sum_{i=1}^{l-1} k_i \cdot 2^{\lambda_i} \geqslant 3 \cdot 2^{\lambda_l} + \sum_{i=1}^{l-1} k_i \cdot 2^{\lambda_i} > 2^{\lambda_l + 1} + (2^{\lambda_l} - 2^{\lambda_l - 1}) > 2^{\lambda_l + 1}$, where the second inequality holds by Lemma 1, which contradicts the fact $k \leqslant 2^{\lambda_l + 1} - 1$. So we must have $k_l = 1$.

Next we prove that k_{l-1} is positive. Suppose $k_{l-1} \leqslant -1$ is negative. Then $k = 2^{\lambda_l} + k_{l-1} \cdot 2^{\lambda_{l-1}} + \sum_{i=1}^{l-2} k_i \cdot 2^{\lambda_i} < 2^{\lambda_l} - 2^{\lambda_{l-1}} + 2^{\lambda_{l-1}-1} < 2^{\lambda_l}$, where the first inequality holds by Lemma 1, which contradicts the fact $k \geqslant 2^{\lambda_l}$. So we have k_{l-1} is positive. This finishes the proof. □

In order to have an intuitive understanding of Proposition 1 and Proposition 2, we give two examples:

Example 1: wNAF representations of scalar when $L_{\mathrm{AD}} > L_0$, $w = 3$

497 $\Longrightarrow$	1	0	0	0	0	-1	0	0	0	1	wNAF form
$\Longrightarrow$	A	D	D	D	D	A	D	D	D	A	"AD"chain
$\Longrightarrow$		1	1	1	1	1	0	0	0	1	binary string

Example 2: wNAF representations of scalar when $L_{\mathrm{AD}} = L_0$, $w = 3$

610 $\Longrightarrow$	1	0	0	0	3	0	0	0	1	0	wNAF form
$\Longrightarrow$	A	D	D	D	A	D	D	D	A	D	"AD" chain
$\Longrightarrow$	1	0	0	1	1	0	0	0	1	0	binary string

From Example 1, we can see that if $L_{\mathrm{AD}} > L_0$, then $k_l = 1$ and $k_{l-1} = -1 < 0$, while in Example 2, it is $L_{\mathrm{AD}} = L_0$ with $k_l = 1$ and $k_{l-1} = 3 > 0$.

According to Proposition 1 and Proposition 2, if $L_{\mathrm{AD}} \geqslant L_0$, we can determine the exact value of MSD and whether the digit k_{l-1} is positive or not. We give a statistical result, which states that there are 597 signatures that satisfy $L_{\mathrm{AD}} \geqslant L_0$ among randomly chosen 1000 signatures. In another word, for about 59.7% signatures, we can recover the MSD of the ephemeral key k and the sign of k_{l-1}.

Next we will show how we will use the results of Proposition 1 and Proposition 2 to improve the lattice attack. We only take the case of $L_{\mathrm{AD}} = L_0$ as an example, the case of $L_{\mathrm{AD}} > L_0$ can be dealt with similarly so we will omit for simplicity.

Assuming for some ephemeral key k, we have $k_l = 1$ and $1 \leqslant k_{l-1} \leqslant 2^w - 1$. Then we have $k = \sum_{j=1}^{l} k_j \cdot 2^{\lambda_j} = 2^{\lambda_l} + k_{l-1} \cdot 2^{\lambda_{l-1}} + \sum_{j=1}^{l-2} k_j \cdot 2^{\lambda_j}$. Let $k_j = 2(d_j - 2^{w-1}) + 1$, where $1 \leqslant j \leqslant l-2$, $0 \leqslant d_j \leqslant 2^w - 1$. Let $k_{l-1} = 1 + 2\overline{k_{l-1}}$, where $0 \leqslant \overline{k_{l-1}} < 2^{w-1}$. So we further have

$$k = \overline{k_{l-1}} \cdot 2^{\lambda_{l-1}+1} + \sum_{j=1}^{l-2} d_j \cdot 2^{\lambda_j + 1} + (2^{\lambda_{l-1}} + 2^{\lambda_l} + \sum_{j=1}^{l-2} 2^{\lambda_j} - \sum_{j=1}^{l-2} 2^{w+\lambda_j}) .$$

It is easy to see that when we use the above representation of k to build the lattice, the dimension can be reduced by 1 each time. Meanwhile, since the unknown $\overline{k_{l-1}}$ is in the interval $[0, 2^{w-1})$, so the window size of $\overline{k_{l-1}}$ is decreased from w to $w-1$. Even when the $(l-1)$-th non-zero digit is merged after using the method of merging, say being merged to a new digit $d'_{l'}$, the window size of $d'_{l'}$ is decreased by 1 if we determine the sign of k_{l-1} beforehand. The decrease of the window size will increase the determinant of lattice, which means that the ratio of $\|\vec{w}\|/GH(L)$ is decreased. By this method, we can remarkably increase the success probability of secret key recovery.

4.3 Enumeration of the MSD

In this subsection, we further investigate the possible values of MSD for the remaining 40.3% signatures satisfying $L_{\mathrm{AD}} < L_0$, so that we can use enumeration to increase success probability with the cost of a little more time.

PROPOSITION 3. *If $L_{\mathrm{AD}} < L_0$, then*

$$2^{L_0 - L_{\mathrm{AD}}} < k_l < 2^{L_0 - L_{\mathrm{AD}} + 1}.$$

PROOF. Rewrite k as $k = k_l \cdot 2^{\lambda_l} + \sum_{i=1}^{l-1} k_i \cdot 2^{\lambda_i}$, where $\lambda_l = L_{\mathrm{AD}} - 1$. From Lemma 1, we have $k_l \cdot 2^{\lambda_l} - 2^{\lambda_l - 1} < k < k_l \cdot 2^{\lambda_l} + 2^{\lambda_l - 1}$, which means that $(2k_l - 1)2^{L_{\mathrm{AD}} - 2} < k < (2k_l + 1)2^{L_{\mathrm{AD}} - 2}$. Since $2^{L_0 - 1} \leqslant k \leqslant 2^{L_0} - 1$, there must be $(2k_l + 1)2^{L_{\mathrm{AD}} - 2} > 2^{L_0 - 1}$ and $(2k_l - 1)2^{L_{\mathrm{AD}} - 2} < 2^{L_0} - 1$, which will separately give that $k_l > 2^{L_0 - L_{\mathrm{AD}}}$ and $k_l < 2^{L_0 - L_{\mathrm{AD}} + 1}$ since k_l is an odd. This finishes the proof. □

From Proposition 3, if $L_{\mathrm{AD}} < L_0$, the number of all the possible cases of the MSD is $2^{L_0 - L_{\mathrm{AD}} - 1}$, which is no greater than 2^{w-2} since $L_0 - L_{\mathrm{AD}} \leqslant w - 1$. So we can enumerate all the possible values of MSD if the number of signatures are small.

Suppose $w = 3$. If $L_0 - L_{\mathrm{AD}} = 1$, we can determine $k_l = 3$; if $L_0 - L_{\mathrm{AD}} = 2$, the only possible values of MSD are $k_l = 5, 7$. According to our statistical results on the **secp256k1** curve, about 45% among all the cases of $L_{\mathrm{AD}} <$

[3] In this proposition we only consider the wNAF representation. While in the modified wNAF representation, if $k_l = 1$, $k_{l-1} < 0$, and $\lambda_l - \lambda_{l-1} = w+1$, the position of k_l is modified from λ_l to $\lambda_l - 1$ to make $L_{\mathrm{AD}} = L_0$. If this case happens, the distance between k_l and k_{l-1} is w, which can be easily detected. So if we find that $L_{\mathrm{AD}} = L_0$, but $\lambda_l - \lambda_{l-1} = w$, then $k_l = 1$ and $k_{l-1} < 0$.

L_0 satisfies that $L_0 - L_{\mathrm{AD}} = 1$, while the remaining satisfies $L_0 - L_{\mathrm{AD}} = 2$. The average time of using the enumeration method is prolonged by $2^{0.403\times 0.55u} \approx 1.167^u$ times, where u is the number of signatures used to recover the secret key. If $u = 4, 5$ and 6, the time of using enumeration is separately prolonged by about 1.849, 2.156 and 2.514 times.

As we can see that, the use of enumeration can decrease the number of non-zero digits of k, thus reducing the dimension of the corresponding lattice with just a little cost of time.

5. EXPERIMENT RESULTS

We applied our attack to the **secp256k1** curve, whose window size is $w = 3$. All executions were performed on an Intel Core i7-3770 CPU running at 3.40GHz in single thread. We use the BKZ algorithms with blocksize 20 and 25 implemented in the NTL library to solve the SVP instances. All of our codes are written in C++. As stated above, our attack is applied on the basic assumption that a perfect double-and-add chain of the ephemeral key k can be obtained. As the success probability of the original method in Section 3.2 is too low, we only give the experiment results using the improved methods. Given the number of signatures $u = 4, 5, 6, 7$, various combinations of improved attack methods, and different lattice reduction algorithms, we execute 200 experiments each time to recover the secret key, with the success probability computed by the ratio of the number of success attacks and 200. We list the results of attacks in Table 3. As we can see, only 4 signatures are enough to recover the secret key, while using 5 signatures we can succeed with probability of 37.5%, using 6 signatures with probability of 90% and using 7 signatures with probability of 94%.

According to [26], it is quite easy to identify 581 full double-and-add chain among 1000 random Flush+Reload attack results, containing only 4 error chains. So it is possible to get a perfect double-and-add chain with probability being about 57.7%. As we can succeed recovering the secret key with 4, 5, 6 and 7 signatures with perfect double-and-add chain, the actual number of signatures we need is about 7, 9, 11 and 13. Note that the success probability using 4 signatures is not high enough, we believe that if our attack use other powerful reduction algorithm, like the BKZ 2.0 algorithm [6], the success probability will be increased.

Our attack can be applied to other curves implemented by wNAF method like the **secp521r1** curve, and can also be extended to any cryptographic implementation using the wNAF representation, like DSA, SM2, etc.

6. CONCLUSIONS

In this paper, we demonstrated that ECDSA implemented by OpenSSL using the wNAF representation can be broken using the lattice attack with the help of the Flush+Reload attack. We develop a new efficient method of extracting and utilizing side-channel information to mount a lattice attack to ECDSA. After that, the methods of elimination, merging, MSD recovering and enumeration are used to remarkably increase the success probability of key recovery. Our attack is mounted to the **secp256k1** curve in experiments. It is shown that given results of perfect Flush+Reload attack, 4 signatures would be enough to recover the secret key.

7. ACKNOWLEDGEMENTS

This work is supported by the National Basic Research Program of China (973 Program) granted No. 2013CB338003. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

8. REFERENCES

[1] The openssl project. OpenSSL – cryptography and SSL/TLS toolkit. http://www.openssl.org.

[2] O. Aciiçmez, Ç. K. Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, ASIACCS 2007, pages 312–320, New York, NY, USA, 2007. ACM.

[3] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom. "Ooh aah... ,just a little bit": A small amount of side channel can go a long way. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded System – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 75–92. Springer Berlin Heidelberg, 2014.

[4] B. B. Brumley and R. M. Hakala. Cache-timing template attacks. In M. Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 667–684. Springer Berlin Heidelberg, 2009.

[5] B. B. Brumley and N. Tuveri. Remote timing attacks are still practical. In V. Atluri and C. Diaz, editors, *Computer Security – ESORICS 2011: 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, pages 355–371. Springer Berlin Heidelberg, 2011.

[6] Y. Chen and P. Nguyen. BKZ2.0: better lattice security estimates. In *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2011.

[7] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation. In *Advances in Cryptology – Proceedings of ICICS 1997*, volume 1334 of *Lecture Notes in Computer Science*, pages 282–290. Springer Berlin Heidelberg, 1997.

[8] P. FIPS. 186-4 digital signature standard (DSS). *National Institude of Standards and Technology (NIST)*, 2013.

[9] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In N. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008: 27th Annual International Conference on the Theory and Application of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 31–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[10] M. Hlaváč and T. Rosa. Extended hidden problem and its cryptanalytic applications. In E. Biham and A. M. Youssef, editors, *Selected areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 114–133. Springer Berlin Heidelberg, 2007.

[11] A. Hollosi, G. Karlinger, T. Rossler, M. Centner, and et al. Die österreichische bürgerkarte, 2008.

[12] N. Howgrave-Grahm and N. P. Smart. Lattice attacks

Table 3: Experiment Results (A. MSD recovering; B. Enumeration; C. Merging and Elimination) ("p" denotes the success probability of the attack, "time" denotes the average time the algorithm cost)

Number of Signatures	Reduction Algorithm	method	p(%)	time(min)
7	BKZ-20	C	24	34.5
		A+C	62	28.9
	BKZ-25	C	68	238.8
		A+C	94	226
6	BKZ-20	C	22	18
		A+C	28	9.48
		A+B+C	51	43.18
	BKZ-25	C	35	103.2
		A+C	61	80
		A+B+C	90	193.58
5	BKZ-20	C	1	18
		A+C	4.5	12.17
	BKZ-25	C	4	36
		A+C	17	41.73
		A+B+C	37.5	102.6
4	BKZ-25	A+C	1.5	40.9
		A+B+C	8	88.8

on digital signature schemes. *Designs, Codes and Cryptography*, 23:283–290, 2001.

[13] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar. Lucky 13 strikes back. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 85–96, New York, NY, USA, 2015. ACM.

[14] D. Johnson, A. Menezes, and S. A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1:36–63, 2001.

[15] P. C. Kocher, J. Jaff, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer Berlin Heidelberg, 1999.

[16] K. Koyama and Y. Tsuruoka. Speeding up elliptic curve cryptosystems using a signed binary windows method. In *Advances in Cryptology - CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 345–357. Springer Berlin Heidelberg, 1992.

[17] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[18] M. Liu and P. Q. Nguyen. Solving BDD by enumeration: An update. In E. Dawson, editor, *Topics in Cryptology – CT-RSA 2013: The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer Berlin Heidelberg, 2013.

[19] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[20] P. Q. Nguyen and I. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology*, 15:151–176, 2002.

[21] P. Q. Nguyen and I. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30:201–217, 2003.

[22] D. Page. Theoretical use of cache memory as a cryptanalytic side-channel. *IACR Cryptology ePrint Archive*, 2002:169, 2002.

[23] C.-P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. In *Fundamentals of Computation Theory – FCT 1991*, volume 529 of *Lecture Notes in Computer Science*, pages 68–85. Springer Berlin Heidelberg, 1991.

[24] J. Solinas. Efficient arithmetic on Koblitz curves. *Design, Codes and Cryptography*, 19(2):195–249, 2000.

[25] E. Tromer, D. A. Osvik, and A. Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[26] J. van de Pol, N. P. Smart, and Y. Yarom. Just a little bit more. In K. Nyberg, editor, *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 3–21. Springer International Publishing, 2015.

[27] S. Vanstone. Responses to NIST's proposal. *Communications of the ACM*, 35:50–52, 1992.

[28] Y. Yarom and N. Benger. Recovering OpenSSL ECDSA nonces using the Flush+Reload cache side-channel attack. *IACR Cryptology ePrint Archive*, 2014:140, 2014.

[29] Y. Yarom and K. Falkner. Flush+Reload: a high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 2014)*, pages 719–732, San Diego, CA, Aug. 2014. USENIX Association.

[30] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS 2012, pages 305–316, New York, NY, USA, 2012. ACM.