Another Lattice Attack Against ECDSA with the wNAF to Recover More Bits per Signature

Ziqiang Ma[1], Shuaigang Li[2], Jingqiang Lin[3(✉)], Quanwei Cai[2], Shuqin Fan[4], Fan Zhang[5,6], and Bo Luo[7]

[1] School of Information Engineering, Ningxia University, Yinchuan, China
maziqiang@nxu.edu.cn
[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
lishuaigang@iie.ac.cn
[3] School of Cyber Security, University of Science and Technology of China, Hefei, China
linjq@ustc.edu.cn
[4] State Key Laboratory of Cryptology, Beijing, China
fansq@sklc.org
[5] School of Cyber Science and Technology, College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China
fanzhang@zju.edu.cn
[6] Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, Hangzhou 310027, China
[7] Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, USA
bluo@ku.edu

Abstract. In the resource-constrained environment such as the Internet of Things, the windowed Non-Adjacent-Form (wNAF) representation is usually used to improve the calculation speed of the scalar multiplication of ECDSA. This paper presents a practical cache side channel attack on ECDSA implementations which use wNAF representation. Compared with existing works, our method exploits more information from the cache side channels, which is then efficiently used to construct

This work was supported in part by the Open Subject of the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences under Grant 2020-MS-08; in part by the Ningxia Natural Science Foundation of China under Grant 2021AAC03078; in part by the Key RD plan of Ningxia Hui Autonomous Region, China under Grant 2021BEB04047; in part by the National Key RD Plan of China under Grant 2020YFB1005803; in part by National Natural Science Foundation of China under Grant 62072398, by National Key Laboratory of Science and Technology on Information System Security, by State Key Laboratory of Mathematical Engineering and Advanced Computing, and by Key Laboratory of Cyberspace Situation Awareness of Henan Province.

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 111–129, 2023.
https://doi.org/10.1007/978-3-031-25538-0_7

lattice attacks in the ECDSA private key recovery. First, we additionally monitor the invert function which is related to the sign of the wNAF digits, and obtain a Double-Add-Invert chain through the Flush+Flush cache side channel. Then, we develop effective methods extracting 154.2 bits information of the ephemeral key per signature for 256-bit ECDSA from this chain, much more than the best known result which extracts 105.8 bits per signature. Finally, to efficiently use the extracted information, we convert the problem of recovering the private key to the Hidden Number Problem (HNP) and the Extended Hidden Number Problem (EHNP) respectively, which are solved by lattice reduction algorithms. We applied the attack on ECDSA with the secp256k1 curve in OpenSSL 1.1.0h. The experimental results show that only 3 signatures are enough to recover the private key. To the best of our knowledge, this work exploits the signs of the wNAF representation, along with the Double-Add chain against ECDSA, to recover the private key with *the least number of signatures.*

Keywords: ECDSA · windowed Non-Adjacent-Form · Lattice attack · Hidden number problem · Extended hidden number problem · Cache side channel

1 Introduction

The ECDSA [5] digital signature scheme based on the elliptic-curve cryptography (ECC), is widely used in lots of popular applications, such as OpenPGP [14], smartcard [27], TLS [30] and Bitcoin [21]. The scalar multiplication is the core operation of ECDSA and its speed determines the total efficiency of ECDSA. Exploiting the windowed Non-Adjacent Form (wNAF) [22,29] of the ephemeral key (the scalar) is a commonly used method to accelerate the scalar multiplication [1,2], especially in resource-constrained environments such as IoT. While this method needs a pre-computed table, which costs many memory resources. Several works [11,18,34] have been put forward to improve the efficiency of wNAF. One of the most effective ways is to exploit the invert function to convert the negative digits into positive digits during the calculation, which can save half of the pre-computed storage space [20].

However, side channel attacks can extract information on the ephemeral key with the wNAF representation. As long as some bits of the ephemeral key are leaked, the ECDSA private key can be fully recovered [24]. With the cache side channel attacks [10,17,33], practical attacks [4,6,9,26,32] are proposed to attack ECDSA. These works [6,9,26,32] observe the execution of ECDSA through cache side channels, and observe the ordered sequence of points addition and doubling during the ECDSA signing. Then, they extract and exploit the different information of the ephemeral key from this chain, and translate the problem of recovering the ECDSA private key to the Hidden Number Problem (HNP) or the Extended Hidden Number Problem (EHNP), which is then solved by lattice reduction algorithms. However, They infer the information of the ephemeral

key's wNAF representation *only* from the Double-Add chain, so that for one signature the number of bits extracted hits the ceiling. The method proposed by Fan [9] is believed to infer almost all the available bits from the Double-Add chain, which performs remarkably better than other attacks [6,26,32]. Although another attack proposed by Allan [4] additionally monitored the invert function of the scalar multiplication implementation, they exploited the analytical method of Van de Pol [26] and obtained 71.4 bits per signature. Naturally, more bits are obtained from the side channels, and fewer signatures are required for recovering the ECDSA private key. For example, existing methods [4,6,9,26,32] require 6, 4, 200, 13, and 85 signatures, respectively, to recover the private key.

In this paper, we propose a more efficient attack against the wNAF implementation with the invert function of ECDSA, which obtains on average 154.2 bits per signature for 256-bit ECDSA and requires only 3 signatures to recover the ECDSA private key. First, we monitor the invert function of the implementation of the scalar multiplication, along with points addition and doubling functions using the cache side channel, and then construct a Double-Add-Invert chain instead of the Double-Add chain [6,9,26,32]. The invert function is invoked only when the sign of the current non-zero digit of the ephemeral key's wNAF representation is opposite to the previous one in the scalar multiplication. Through this information, we can extract the signs of the non-zero digits. Then, taking full advantage of information obtained through the Double-Add-Invert chain, we construct the HNP and EHNP instances respectively, to recover the ECDSA private key. We apply our attacks to the secp256k1 curve in OpenSSL 1.1.0h. The Flush+Flush [10] attack is used to monitor the functions of double, add and invert, and construct the Double-Add-Invert chain. From this chain, we successfully determine whether each digit of the ephemeral key's wNAF representation is zero or not, and also the signs of the non-zero digits. We extract on average 154.2 bits from one signature through the perfect Double-Add-Invert chain for 256-bit ECDSA. With the HNP problem, we need about 248 signatures to recover the private key with a success probability of 1.5% (in Sect. 4). While using the EHNP problem, 3 signatures are enough with a success probability no less than 69.9% (in Sect. 5).

Our contributions are summarized as follows:

- We present an efficient cache side channel attack to recover the private key of ECDSA with the invertible wNAF representation. First, we construct a Double-Add-Invert chain by additionally monitoring the invert function. Then, we propose two new lattice attacks to exploit the positions and signs of all non-zero digits in the ephemeral key's wNAF representation obtained through the Double-Add-Invert chain.
- We apply our methods to attack the secp256k1 curve in OpenSSL 1.1.0h. Through the cache side channel, 154.2 bits are obtained per signature on average. The experiments show that only 3 signatures are enough to recover the private key with a success probability no less than 69.9%.

The rest of this paper is organized as follows. Section 2 introduces the preliminaries. Section 3 shows how to improve the cache side channel to get more

information. Sections 4 and 5 construct the lattice attacks with HNP and EHNP, respectively. Section 6 compares our attacks with existing works. And, Sect. 7 draws the conclusion.

2 Preliminaries

In this section, we first present the related concepts about the ECDSA and the wNAF. Then, we describe the attack method of the cache side channels. Also, the hidden number problem and the extended hidden number problem are introduced to utilize the data obtained from cache side channels.

2.1 The Elliptic Curve Digital Signature Algorithm

ECDSA [5,15] is based on the intractability of the elliptic curve discrete logarithm problem in finite field. We define a prime p, and set E as the elliptic curve on the finite field $\mathbb{F}_p$. G is the generator of the group with order q, which is a fixed point on the curve. We set an integer α as the private key of ECDSA that should satisfy $0 < \alpha < q$. The point $Q = \alpha G$ is the corresponding public key. Also the information about the elliptic curve is public. Given a message m, the ECDSA signature is generated as follows:

1. Generate a random number k, $0 < k < q$, as the ephemeral key.
2. Compute $(x, y) = kG$, and then set $r = x \bmod q$; return to 1 if r equals 0.
3. calculate $s = k^{-1}(h(m) + r \cdot \alpha) \bmod q$, h is a hash function; return to 1 if s equals 0.

Thus, the computed ECDSA signature of m is (r, s).

2.2 The Scalar Multiplication Using wNAF Representation

Scalar multiplication kG is caculated in the second step of ECDSA signature generation. Several algorithms (e.g. Montgomery Ladder) can be used to implement the scalar multiplication. Among them, the window Non-Adjacent Form (wNAF) algorithm [29] is most commonly used due to its speed. In wNAF, the ephemeral key k is expressed as $k = \sum 2^i k_i$. That means k is represented as a sequence of digits k_i. Each digit is either zero or an odd number satisfying $-2^w < k_i < 2^w$. w is the window size. In this representation, any non-zero digit is required to follow at least w zero digits.

For the scalar multiplication using the wNAF algorithm, during the initialization phase, it need to choose a window size w. Then the points $\{\pm G, \pm 3G, ..., \pm(2^w - 1)G\}$ are precomputed and stored. The multiplication kG is calculated as described in Algorithm 1, after k is under the wNAF representation.

In Algorithm 1, the if-then block (Line 4) uses k_i to determine whether running into the branch. This is a vulnerability to cache attacks. An attacker can apply the cache attacks by using a spy process to monitor add and double function and get a Double-Add chain to determine whether k_i is zero or not.

Algorithm 1. Implementation of kG with wNAF

Input: Scalar k in wNAF: k_0, k_1, ..., k_{l-1}, precomputed points $\{\pm G, \pm 3G, ..., \pm(2^w - 1)G\}$
Output: kG

```
1: Q ← G
2: for i from l − 1 to 0 do
3:     Q ← 2 · Q
4:     if k_i ≠ 0 then
5:         Q ← Q + k_i G
6:     end if
7: end for
8: return Q
```

2.3 The Scalar Multiplication with Invert Function

Here, we adopt OpenSSL 1.1.0h as the example to describe the implementation of the wNAF with the invert function (called the invertible wNAF representation). The core computation of the function is shown in Algorithm 2.

In this function, if the digit is non-zero, it runs into the if-then block in Line 6, and then determines whether an invert function is needed. The invert function is to compute the inverse of a point (the internal value of kG here). If the sign of the non-zero digit k_i is opposite to the previous non-zero digit, the invert operation (`EC_POINT_invert()`) is performed (Line 13), which makes it only need to precompute and store the points $\{G, 3G, ..., (2^w - 1)G\}$, and saves half of the storage space.

From Algorithm 2, it can be found that two conditional branches are vulnerable. First is the double-addtion branch which is already used to determine whether each digit of k is zero or not in the attack. Second is the invert branch, which is related to the signs of the non-zero digits. Combining with the sequence of the double and addition operations, the information of the sign can be inferred.

2.4 Cache Side Channel Attacks

The cache side channel attack was firstly proposed in 2002 [25]. It exploited the fact that accessing data from caches is much faster than from memory. An attacker can use these time variations of cache hits or misses to infer the operations executed by the victim process and then extract the secret information. In this section, we introduce the Flush+Flush [10] attack as the example which is used in our work.

The spy and the victim processes have shared momeory, which is the basis of the Flush+Flush [10] attack. Then the attacker can apply a spy process to detect the state of the specific memory lines in the caches of the victim process. The execution time of the `clflush` instruction is measured in Flush+Flush attack to infer if the specific cache is padding or not. If the cache is padding, the execution time of `clflush` is longer, and if the cache is empty, the execution time is shorter. The Flush+Flush attack obtaining information involves three steps:

Algorithm 2. The Implementation of the Scalar Multiplication in OpenSSL

Input: Scalar k in wNAF $\{k_0, k_1, ..., k_{l-1}\}$, precomputed points $\{G, 3G, ..., (2^w-1)G\}$
Output: kG

```
1: r ← 0, is_neg ← 0, r_is_inverted ← 0
2: for i from l − 1 to 0 do
3:     if r ≠ 0 then
4:         EC_POINT_dbl(r)   // double
5:     end if
6:     if k_i ≠ 0 then
7:         is_neg ← (k_i < 0)
8:         if is_neg then
9:             k_i ← −k_i
10:        end if
11:        if is_neg ≠ r_is_inverted then
12:            if r ≠ 0 then
13:                EC_POINT_invert(r)   // invert
14:            end if
15:            r_is_inverted ←!r_is_inverted
16:        end if
17:        if r = 0 then
18:            r ← EC_POINT_copy(k_iG)
19:        else
20:            r ← EC_POINT_add(r, k_iG)   // add
21:        end if
22:    end if
23: end for
24: return r
```

- **Flush:** The attacker flushes out the specific target memory lines from the caches by the `clflush` instruction.
- **Idle:** In this step, the attacker waits the victim running for a little time slot.
- **Flush:** The attacker executes the `clflush` instruction again and measures the instruction execution time. The shorter time means that the victim does not access the target memory lines. Otherwise, it means that the victim accesses the target memory lines.

Therefore, the attacker infers the victim's memory activities by the execution time, and further inferences the secret information.

2.5 The (Extended) Hidden Number Problem and Lattice Attack

The attacker cannot obtain all the private key information from the cache side channel attack against the wNAF algorithm. So the lattice attack is always needed to infer the complete private key. The attacker can use the partial information obtained to construct the (Extended) Hidden Number Problem and solve it through the lattice reduction algorithm to recover the private key.

The Hidden Number Problem (HNP) is presented to recover the secret key of Diffie-Hellman key exchange [7], DSA [13] and ECDSA [24], when some consecutive bits of the ephemeral key are known. We have a positive l, a prime number q, and randomly chose $t_1, t_2, ..., t_d$ in $\mathbb{F}_q$. The number pairs (t_i, u_i) are known, and satifsy

$$v_i = |\alpha t_i - u_i|_q \leq q/2^{l+1}, \quad 1 \leq i \leq d.$$

The $\alpha \in \mathbb{F}_q$ is the unknown number which is called the hidden number. Thus the HNP is to recover the unknown α. In this expression, $|\cdot|_q$ denotes the reduction modulo q into the range $[-q/2, ..., q/2)$.

The Extended Hidden Number Problem (EHNP) introduced in [12] also can be used to recover the private key [9] of ECDSA. We have a prime number N and u congruences

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

In these congruences, $0 < x < N, \beta_i, a_{i,j}, c_i, l_i$ and $\varepsilon_{i,j}$ are all known. $k_{i,j}$ and x are unknown and they satisfy $0 \leq k_{i,j} \leq 2^{\varepsilon_{i,j}}$. The EHNP is to recover the unknown x satisfying the congruences above.

For the ECDSA algorithm, attackers take advantage of the signature equation $s = k^{-1}(h(m) + r \cdot \alpha) \bmod q$. Based on the partial information related to the ephemeral key obtained by the attackers through the cache side channels, they transform this equation to satisfy the form of HNP or EHNP. Then the private key as the hidden number can be calculated and recovered by solving the SVP/CVP problem in lattice converted from HNP or EHNP using the lattice reduction algorithm such as LLL [16] or BKZ [28].

3 Improving Cache Side Channel Attack on Invertible wNAF Representation

This section proposes how to get more bits of the ephemeral key with the wNAF representation through the cache side channel. First, we analyze the invert function implemented in the wNAF representation in ECDSA and show how to use the Double-Add-Invert chain obtained from the cache side channel to extract the information about the ephemeral key. Then we implement the Flush+Flush attack against the secp256k1 curve in OpenSSL 1.1.0h to obtain the cache side channel information.

3.1 Attacking Invertible wNAF Through the Cache Side Channel

As shown in Algorithm 2, each digit of k performs a double function. If the digit is not zero and the sign of the digit is opposite to the prior one, the invert function is called. Finally, the addition function is called to add the internal value of kG with a precomputed point indexed by the absolute value of the digit.

We use the vulnerability that the invert function is called conditionally to improve the attack to obtain more valid data. We use a spy process to monitor the double, add and invert functions during computing the scalar multiplication. The time is divided into slots, and in each slot, the spy determines whether the three functions are performed or not by monitoring the cache hits/misses. Then we obtain a Double-Add-Invert chain. According to the Double and Add in this chain, we determine whether each digit of k is zero or not, as done in previous works. Then, based on the Invert in this chain, we infer the sign of each non-zero digit.

When we use the Double-Add-Invert chain to extract the digits of k, the Double represents the double function is called, and the Add represents both double and add are called. The Invert represents the invert function is called. Therefore, the appearance of Double means that k_i is zero and the Add means that k_i is not zero. We use the Invert to determine the sign of k_i. The sign of k_i is related to the previous non-zero digit. First, the Invert comes out together with Add. If the Invert appears, it represents that the sign of k_i is opposite to the previous non-zero digit. While, if the Invert does not appear when Add comes out, it represents that the sign of this digit is the same as the previous non-zero one. In the wNAF representation, the most significant digit is always positive. Thus, we can determine the sign of all non-zero digits. In this way, we obtain both the positions and signs of all non-zero digits.

3.2 The Implementation of Flush+Flush Attack

We use the Flush+Flush technique instead of the Flush+Reload to monitor the functions. First, the cost of Flush is about 160 cycles [10] less than the cost of Reload. Second, the measurement stage and flush stage are merged into one stage, because in the measurement stage the execution of `clflush` also plays the role of flushing the cache. Thus, the precision is much higher than the Flush+Reload. Moreover, compared with Allan's method [4] , using Flush+Flush does not additionally degrade the performance of the system, nor does it increase the risk of being detected by the victim.

We launched the Flush+Flush attack on an Acer Veriton T830 running Ubuntu 16.04. The machine has an Intel Core i7-6700 processor with four execution cores and an 8 MB LLC. The attacking target is the ECDSA implemented in OpenSSL 1.1.0h, which uses wNAF representation in the scalar multiplication. We attack the 256-bit curve secp256k1 for the experiments.

Threshold. We monitor the time to execute the `clflush` instruction. For each address of the monitored functions, we record the time of flushing the cache 1000 times, and take the time larger than 99 percent of samples plus 6 cycles as the threshold for this address. The thresholds are recalculated every time before the attack is mounted.

Time Slot. For the attack, time slots are set approximately 2000 cycles. In each slot, the spy process flushes the memory lines of the add, double and invert functions (`EC_POINT_dbl()`, `EC_POINT_add()` and `EC_POINT_invert()`) out of the caches.

Experimental Results. Figure 1 shows a fragment of outputs by the spy process when performing the ECDSA with the secp256k1 curve. In this figure, □, ◊ and △ represent "double", "add" and "invert" respectively. From this fragment, three operations are clearly distinguished, so we can easily obtain the Double-Add-Invert chain.

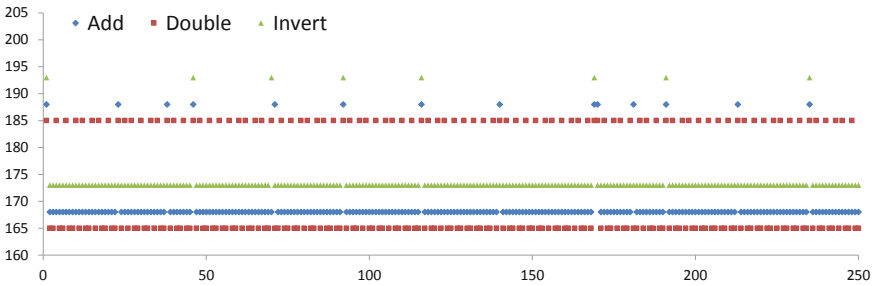

Fig. 1. A fragment of the output of the Flush+Flush attack.

4 Recover the ECDSA Private Key with HNP

After we obtain the information from the cache side channel, the most intuitive idea is that first recovering some consecutive bits of the ephemeral key k from the Double-Add-Invert chain and then constructing the HNP or EHNP with these bits. So we first recover the consecutive bits at the position of every non-zero digits of k. Then we construct the HNP problem using these bits and solve it by the approximate CVP/SVP Problem with the lattice reduction algorithm to recover the ECDSA private key.

4.1 Recovering Consecutive Bits

First, we denote the wNAF representation of k as $k = \sum k_i 2^i$, and the binary representation as $k = \sum b_i 2^i$. When we know the information about whether k_i is zero and the sign of the non-zero k_i, we can simply determine some bits of k. For example, if we obtain the sign of the least non-zero k_j, we can infer that b_j is one and b_i is zero for $0 \leq i < j$. But for arbitrary non-zero digits, it can not directly determine whether the bit corresponding to the position of the digit is zero or one.

Set m and $m + n$ as the positions of two consecutive non-zero digits of the wNAF representation, and w be the window size. That is, $k_m, k_{m+n} \neq 0$ and

$k_{m+i} = 0$ for all $0 < i < n$. We analyse the transformation method between the binary and wNAF representation, getting the following result:

$$b_{m+n} = \begin{cases} 0, & k_m < 0 \\ 1, & k_m > 0 \end{cases}, \quad b_{m+i} = \begin{cases} 0, & k_m > 0 \\ 1, & k_m < 0 \end{cases}, \quad w \le i \le n-1 \tag{1}$$

And if m is the position of the least non-zero digit of k,

$$b_i = \begin{cases} 1, & i = m \\ 0, & 0 \le i < m \end{cases}. \tag{2}$$

In this way, at the position of every non-zero digit we can obtain $n - w + 1$ consecutive bits of k except at the position of the least non-zero digit being $m + 1$. For the wNAF representation, the average number of non-zero digits of k is approximately $(\lceil \log_2 q \rceil + 1)/(w + 2)$. While the average distance between consecutive non-zero digits is $w + 2$, i.e. on average $n = w + 2$, meaning that we can obtain 3 consecutive bits on average at every non-zero digit (except the least one). Thus, on average we can obtain approximately $3(\lceil \log_2 q \rceil + 1)/(w+2)$ bits of the ephemeral key k in total. Meanwhile, the minimal value of n is $w+1$, so the minimal length of the consecutive bits is 2. This illustrates that all the sequences of consecutive bits obtained (except the least) are no less than 2 bits.

For the secp256k1 curve implemented in OpenSSL, $\lceil \log_2 q \rceil + 1 = 257$, $w = 3$. So the total number of bits per signature we obtain is $3(\lceil \log_2 q \rceil + 1)/(w + 2) = 154.2$. In theory, two signatures would be enough to recover the 256-bit private key as $2 \times 154.2 = 308.4 > 256$.

4.2 Constructing the Lattice Attack with HNP

In this section, we transform the problem of recovering the private key to the HNP instance, and further convert it to the CVP/SVP instance in a lattice. Our method is based on the analysis from [23]. And we make the following improvements to it. First, the length of the consecutive bits used to construct the lattice is variable while the prior work fixes the length, which may lose some information. Second, in our method the position of consecutive bits is arbitrary in the ephemeral key and does not need to be fixed, while the prior work needs all the consecutive bits at the same position. Finally, from one signature we obtain multiple sequences of consecutive bits, and all of them can be used for constructing the lattice as long as the length of the sequence is satisfied, while the prior work only generates one sequence of consecutive bits for one signature.

To construct an HNP instance using arbitrary consecutive bits, we use the standard analysis from [23]. Assuming that we have the l consecutive bits of k with the value of a, starting at some known position j. So k is represented as $k = 2^j a + 2^{l+j} b + c$ for $0 \le a \le 2^l - 1$, $0 \le b \le q/2^{l+j}$ and $0 \le c < 2^j$. We determine the following values

$$\begin{cases} t = \lfloor r\lambda s^{-1} \rceil_q \\ u = \lfloor (2^j a - s^{-1} h(m))\lambda \rceil_q \end{cases}, \tag{3}$$

where $\lfloor\cdot\rfloor_q$ denotes the reduction modulo q into range $[0, ..., q)$. where (r, s) is the ECDSA signature, λ satisfies that $|\lambda|_q < q2^{-j-l/2}$ and $|\lambda 2^{j+l}|_q \leq q/2^{j+l/2}$, and $\lfloor\cdot\rfloor_q$ denotes the reduction modulo q into range $[0, ..., q)$.

It satisfies that

$$|\alpha t - u|_q < q/2^{(l/2-1)}. \tag{4}$$

This way, an HNP instance is constructed.

In practice, OpenSSL uses $k + q$ as the ephemeral key. So the Eq. 3 remains the same, but the Inequality 4 turns into

$$|\alpha t - u|_q < q/2^{(l/2-\log_2 3)}. \tag{5}$$

Note that, the Eq. 5 represents that the $l/2 - \log_2 3 - 1$ most significant bits of $\lfloor\alpha t\rfloor_q$ is u, based on the definition of the HNP. So it should satisfy that $l/2 - \log_2 3 - 1 \geq 1$, i.e. $l > 7$. That means the length of the consecutive bits used to construct the HNP instance should be larger than 7, although we could use all the sequences of the consecutive bits of the ephemeral key in theory.

Next we turn the HNP instance into the lattice problem. We use d triples (t_i, u_i, l_i) to construct a $d + 1$ dimensional lattice $L(B)$ spanned by the rows of the following matrix:

$$B = \begin{pmatrix} 2^{l_1+1}q & 0 & \cdots & 0 & 0 \\ 0 & 2^{l_2+1}q & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 2^{l_d+1}q & 0 \\ 2^{l_1+1}t_1 & \cdots & \cdots & 2^{l_d+1}t_d & 1 \end{pmatrix}.$$

Let the vector $\mathbf{x} = (2^{l_1+1}\alpha t_1 \bmod q, ..., 2^{l_d+1}\alpha t_d \bmod q, \alpha)$, and the vector $\mathbf{u} = (2^{l_1+1}u_1, ..., 2^{l_d+1}u_d, 0)$. It can be proved that the vector $\mathbf{x}$ is one of the closest vectors to $\mathbf{u}$. Inputing B and $\mathbf{u}$, and solving the approximate CVP problem, the vector $\mathbf{x}$ is revealed. hence the private key α is recovered.

The approximate CVP problem can be transformed to an approximate SVP instance. d triples (t_i, u_i, l_i) can construct a lattice $L(B')$ with the dimension $d + 2$ spanned by the rows of the following matrix

$$B' = \begin{pmatrix} B & 0 \\ \mathbf{u} & q \end{pmatrix}.$$

Similarly, the vector $\mathbf{x}' = (2^{l_1+1}(\alpha t_1 - u_1) \bmod q, ..., 2^{l_d+1}(\alpha t_d - u_d) \bmod q, \alpha, -q)$ belonging to the lattice $L(B')$ is a very short vector. But this lattice also contains another vector $(-t_1, ..., -t_d, q, 0) \cdot B = (0, ..., 0, q, 0)$, which also is a very short vector in the lattice. Therefore we expect the two shortest vectors in a reduced basis of the lattice contain $\mathbf{x}'$ with a suitably lattice reduction algorithm. Then we acquire the secret key α.

Table 1. The success probability of solving SVP with different parameters.

Dimension	Block size					
	$l \geq 9$		$l \geq 10$		$l \geq 11$	
	10	20	10	20	10	20
50	0	0	0	0	0	0
60	0	0	0	0	3.0	7.5
70	0	0	0.5	1.0	29.5	30.5
80	1.0	1.5	4.0	8.5	49.0	65.0
100	0.5	4.0	21.0	33.5	70.5	83.5
120	2.0	4.5	21.5	35.0	77.5	92.0
140	3.0	8.0	29.0	46.0	88.5	99.0
160	2.5	13.0	27.5	49.0	94.0	99.5
180	3.5	11.5	37.0	57.0	97.0	100.0
200	9.5	26.0	46.0	66.0	99.0	100.0
220	15.0	29.0	51.0	72.5	100.0	100.0

4.3 Lattice Attack on Secp256k1

We apply the lattice attack to the curve secp256k1 and assume the Flush+Flush attack is perfect, which means we correctly obtain the Double-Add-Invert chain and recover all the information about the digits of the ephemeral key it contains.

In the experiments, we use the BKZ algorithm implemented in `fplll` [31] to solve the SVP problem converted from the HNP problem. We denote the success probability as the amount of successfully recovering the private key divided by the total number of the lattice attacks. We want to find the optimal strategy for our attack in terms of the following parameters:

- the minimal value of l (length of the consecutive bits of k)
- the block size of BKZ
- the lattice dimension

Table 1 shows the success probability for different dimensions and block sizes of solving the SVP instance. In each case, we run 200 experiments and compute the success probability. Although in Sect. 4.2 the HNP problem introduced requires that l should be larger than 7. But in our experiments, when $l = 8$, the private key can not be successfully recovered. We can successfully recover the private key of a 256-bit ECDSA only need 60 sequences of consecutive bits with a success probability of 7.5%. These 60 sequences come from up to 60 signatures. That means we just need about 60 signatures satisfied the length requirement to successfully recover the ECDSA private key.

We analysed 10000 signatures. 32.33% signatures contain the sequence of consecutive bits that $l \geq 9$, 17.44% signatures contain the sequence that $l \geq 10$ and 9.08% signatures contain the sequence that $l \geq 11$. So, obtaining 60

satisfying signatures ($l \geq 11$) needs 661 signatures totally. However, when using 80 signatures that satisfy $l \geq 9$, the total number of signatures needed is the least, just 248.

5 Recover the ECDSA Private Key with EHNP

We can see that the restriction on the length of the bits has a great influence on the number of signatures we need to observe. Although we have exploited the sign information of the wNAF representation, the result is not as expected, worse than Allan's attack [4], which only needs 6 signatures. So, in this section, we directly exploiting the wNAF representation to construct EHNP problem with the known information from the Double-Add-Invert chain to recover the ECDSA private key. The final result shows this method only needs 3 signatures, better than existing results.

5.1 Extracting More Information

Suppose in the obtained Double-Add-Invert Chain, the numbers of A is l, whose positions are $\lambda_i (1 \leq i \leq l)$, separately. So we can easily have

$$k = \sum_{i=1}^{l} k_i^{'} 2^{\lambda_i}, \qquad k_i^{'} \in \{-7, -5, -3, -1, 1, 3, 5, 7\}. \tag{6}$$

On the other hand, from the Invert chain, we can easily know that the $k_i^{'}$ is a positive integer or a negative integer. Suppose $k_i^{'} = (-1)^{h_i} k_i^*$, where $(-1)^{h_i}$ is the sign of $k_i^{'}$ which is known, $k_i^* \in \{1, 3, 5, 7\}$. Write $k_i^* = 2k_i + 1$, where $k_i \in \{0, 1, 2, 3\}$. Then we have

$$k = \sum_{i=1}^{l} (-1)^{h_i} k_i^* 2^{\lambda_i} = \sum_{i=1}^{l} (-1)^{h_i} (2k_i + 1) 2^{\lambda_i} = \bar{k} + \sum_{i=1}^{l} (-1)^{h_i} k_i 2^{\lambda_i + 1} \tag{7}$$

where $\bar{k} = \sum_{i=1}^{l} (-1)^{h_i} 2^{\lambda_i}$, h_i, λ_i are known and the only unknowns are $k_i \in \{0, 1, 2, 3\}$.

For the secp256k1 curve implemented in OpenSSL, from Eq.(7), there are approximately 51.4*2= 102.8 bits being unknown, which means the number of the known bits is about $257 - 102.8 = 154.2$ on average. It is about 1.5 times of the number of bits in [9]. In theory, we just need two signatures to recover the 256-bit private key, because the least integer m is 2 such that $m \cdot 154.2 > 256$.

5.2 Find the Target Vector with New Lattice

Similar to [9], we translate to the EHNP problem to recover ECDSA private key. Then we further construct a lattice and solve the approximate SVP using lattice reduction algorithm.

Reduction to EHNP Problem. Transforming the expression of the signature and we can have the equation

$$\alpha r - sk + H(m) = 0 \mod q. \tag{8}$$

Substitue (8) with Eq. (7), we get Eq. (9)

$$\alpha r - \sum_{i=1}^{l}((-1)^{h_i}2^{\lambda_i+1}s)k_i - (s\bar{k} - H(m)) + hq = 0. \tag{9}$$

where $\alpha, k_i (1 \le i \le l)$ and $h \in \mathbb{Z}$ is unknown.

If we have u signatures (r_i, s_i) of different messages $m_i (1 \le i \le u)$ with the same private key α. From (9) we can easily have

$$\begin{cases} \alpha r_1 - \sum\limits_{j=1}^{l_1}((-1)^{h_{1,j}}2^{\lambda_{1,j}+1}s_1)k_{1,j} - (s_1\bar{k}_1 - H(m_1)) + h_1^{'}q = 0 \\ \qquad\qquad \vdots \\ \alpha r_i - \sum\limits_{j=1}^{l_i}((-1)^{h_{i,j}}2^{\lambda_{i,j}+1}s_i)k_{i,j} - (s_i\bar{k}_i - H(m_i)) + h_i^{'}q = 0 \\ \qquad\qquad \vdots \\ \alpha r_u - \sum\limits_{j=1}^{l_u}((-1)^{h_{u,j}}2^{\lambda_{u,j}+1}s_u)k_{u,j} - (s_u\bar{k}_u - H(m_u)) + h_u^{'}q = 0 \end{cases} \tag{10}$$

the number of non-zero digits is l_i. $(-1)^{h_{i,j}}k_{i,j}$ is the j-th non-zero digit in the i-th signature, $\lambda_{i,j}$ is its position, $\bar{k}_i = \sum\limits_{j=1}^{l_i}(-1)^{h_{i,j}}2^{\lambda_{i,j}} \mod q$ and $\alpha, h_i^{'}, k_{i,j}$ are unknown elements in the equations.

Given Eq. (10), find $0 < \alpha < q$ and $0 \le k_{i,j} \le 2^{w-1} - 1$. We denote this problem DSA-EHNP.

Constructing the Target Vector. Next we will translate the DSA-EHNP problem to the problem of finding the short vector of a related lattice, which uses the same way of [9].

Notice that we have $k_{i,j} \in \{0, 1, \cdots, 2^{w-1}-1\}$. For $1 \le i \le u$ and $1 \le j \le l_i$, denote $c_{i,j} = (-1)^{h_{i,j}+1}2^{\lambda_{i,j}+1}s_i \mod q$, $\beta_i = H(m_i) - s_i\bar{k}_i \mod q$. The lattice L is spanned by the following matrix B in Eq. (11).

It is easy to check there exists

$$\begin{aligned} \mathbf{w} &= (h_1^{'}, \cdots, h_\mu^{'}, \alpha, k_{1,1}, \cdots, k_{1,l_1}, \cdots, k_{u,1}, \cdots, k_{u,l_u}, -1)\mathbf{B} \\ &= (0, \cdots, 0, \tfrac{\alpha}{q}\delta - \tfrac{\delta}{2}, \tfrac{k_{1,1}}{4}\delta - \tfrac{\delta}{2}, \cdots, \tfrac{k_{1,l_1}}{4}\delta - \tfrac{\delta}{2}, \cdots, \\ &\quad \tfrac{k_{u,1}}{4}\delta - \tfrac{\delta}{2}, \cdots, \tfrac{k_{u,l_u}}{4}\delta - \tfrac{\delta}{2}, -\tfrac{\delta}{2}) \in L(\mathbf{B}), \end{aligned}$$

and the Euclid norm of the vector $\mathbf{w}$ satisfies $\|\mathbf{w}\| \le \frac{\delta}{2}\sqrt{n-u}$, where n is the dimension of the lattice, i.e., $n = \sum\limits_{i=1}^{u} l_i + u + 2$.

Table 2. The success probability for solving the EHNP problem. (merge: elimination and merging, MSD: recovering the MSDs, SMSD: enumeration of the MSD.)

Number of signatures	Optimization methods	Success probability (%)	Time (s)
3	n_merge_n_MSD_n_SMSD	69.9	341
	n_merge_MSD_n_SMSD	87.5	242
	n_merge_MSD_SMSD	89.1	273
4	merge_n_MSD_n_SMSD	99.8	497
	merge_MSD_n_SMSD	100	300
	merge_MSD_SMSD	100	320

$$\mathbf{B} = \begin{pmatrix} q & & & & & & & & & & & \\ & \ddots & & & & & & & & & & \\ & & q & & & & & & & & & \\ r_1 & \cdots & r_u & \frac{\delta}{q} & & & & & & & & \\ c_{1,1} & & & & \frac{\delta}{4} & & & & & & & \\ \vdots & & & & & \ddots & & & & & & \\ c_{1,l_1} & & & & & & \frac{\delta}{4} & & & & & \\ & \ddots & & & & & & \ddots & & & & \\ & & c_{\mu,1} & & & & & & \frac{\delta}{4} & & & \\ & & \vdots & & & & & & & \ddots & & \\ & & c_{\mu,l_\mu} & & & & & & & & \frac{\delta}{4} & \\ \beta_1 & \cdots & \beta_u & \frac{\delta}{2} & \frac{\delta}{2} & \cdots & \frac{\delta}{2} & \cdots & \frac{\delta}{2} & \cdots & \frac{\delta}{2} & \frac{\delta}{2} \end{pmatrix} \tag{11}$$

The determinant of lattice $L(\mathbf{B})$ is $\|L\| = \frac{1}{2}q^{u-1} \cdot \delta^{n-u}(\frac{1}{4})^{n-u-2}$. The target vector $\mathbf{w}$ may not be the shortest vector, however, if we choose a appropriate value of δ, the target vector $\mathbf{w}$ which will be a pretty short vector which can be found by lattice reduction algorithm ([3,8,16,28]), thus, the secret key α can be recovered.

5.3 Attacking the Secp256k1

We apply this lattice attack to the curve secp256k1 and also assume that the Flush+Flush attack is perfect. In the experiments, we use the BKZ with block size 30 to solve the SVP problem converted from the EHNP problem.

Fan [9] proposed three optimization methods: elimination and merging, recovering the MSDs, and enumeration of the MSD. In our experiments, we measure the success probability of the lattice attack with these optimization methods.

Table 2 shows the success probability for different number of signatures and optimization selections. We can successfully recover the private key of a 256-bit ECDSA only need 3 signatures for at least 69.9%. We also tried to use 2 signatures to recover the ECDSA private key, but it can not succeed. Although we already reach the best result, the same as in [19], theoretically we only need 2 signatures to recover the private key. That means we can still make improvements on it to reach the theoretical optimal number.

Table 3. Comparison with previous attack methods

Methods	Exploited information	HNP or EHNP	# of bits	# of signatures
Benger et al. [6]	LSB	HNP	2	200
Van de Pol et al. [26]	Half positions of non-zero digits	HNP	47.6	13
Wang et al. [32]	Positions of two non-zero digits and the length of the wNAF representation of k	HNP	$\geq$2.99	85
Allan et al. [4]	Half positions and signs of non-zero digits	HNP	71.4	6
Fan et al. [9]	All positions of non-zero digits	EHNP	105.8	4
Micheli et al. [19]	All positions of non-zero digits	EHNP	105.8	3
Ours	All positions and signs of non-zero digits	HNP	154.2	248
		EHNP	154.2	3

6 Comparison with Other Lattice Attacks

In this section, we compared our method with the previous attacks. Generally, through cache side channels attackers obtain the Double-Add or Double-Add-Invert chain and extract partial information about k. Then, the private key recovery from incomplete information of k is transformed into a problem that can be solved by lattice reduction, such as the HNP or EHNP problem. Finally, through being converted to the CVP/SVP problem in the lattice, the ECDSA private key is recovered.

With the HNP problem, several works have been proposed. They all used the Flush+Reload attack to obtain the Double-Add chain. Then they used the different information extracted from the Double-Add chain, especially the least significant bits (LSBs) of the ephemeral key [6], half of the consecutive non-zero digit [26], and the positions of two non-zero digits and the length of the wNAF representation [32]. These methods extracted restricted bits from the Double-Add chain, so that they need many signatures to recover the private key.

Besides, Allan et al. [4] first used the Flush+Reload attack to monitor the invert function and used a performance degradation attack to increase the attack accuracy. Then they exploited Van de Pol's method to construct the HNP problem with the extra information to recover the ECDSA private key. This method

can extract 71.4 bits per signature on average for the secp256k1 curve and recover the private with 6 signatures.

While with the EHNP problem, Fan et al. [9] extracted all positions of digits from the Flush+Reload attack and took advantage of them to construct an EHNP instance. They managed to obtain on average 105.8 bits per signature for the secp256k1 curve. With some optimization only 4 signatures are needed to recover the private key with the probability being 8%. Subsequently, Micheli et al. [19] improved Fan's work and optimized the lattice to recover the ECDSA private by only 3 signatures.

We compare these attacks with ours in detail in four aspects as shown in Table 3. Our method exploits both the signs and the positions of the non-zero digits of the ephemeral key k achieved from the Flush+Flush attack. It extracts the largest amount of information, on average 156.2 bits per signature for the secp256k1 curve. We use both the HNP and EHNP problems to recover the ECDSA private key. Although the using HNP problem needs 248 signatures, using EHNP problem the number of signatures needed to recover the private key is only 3 with the success probability being not less than 69.9%.

7 Conclusion

In this paper, we demonstrate a practical attack on the ECDSA algorithm implemented with the scalar multiplication using the wNAF representation. We improve the cache side channel attack by using the Flush+Flush technique and adding an extra monitor to the invert function. Through them, we get the extra information, i.e., the signs of all non-zero digits of the ephemeral key. Then, we construct the HNP and EHNP instances respectively, to utilize the extracted information for the ECDSA private key recovery.

This work obtains the information about the signs of the non-zero digits of the ephemeral key without using performance degradation and uses the least number of signatures to recover the ECDSA private key. We applied the Flush+Flush attack to the secp256k1 curve in OpenSSL 1.1.0h to verify the availability of monitoring the invert function. from the Double-Add-Invert chain we extract on average 154.2 bits of information per signature. If the obtained Double-Add-Invert chain is perfect, 3 signatures are enough to recover the ECDSA private key by using the EHNP problem with a success probability no less than 69.9%. This result reaches the best one as ever known with higher success probability.

References

1. Cryptlib Encryption Toolkit. https://www.cs.auckland.ac.nz/pgut001/cryptlib/ (2020)
2. The Legion of the Bouncy Castle (2020). http://www.bouncycastle.org/
3. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 717–746. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_25

4. Allan, T., Brumley, B.B., Falkner, K., van de Pol, J., Yarom, Y.: Amplifying side channels through performance degradation. In: Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC), pp. 422–435 (2016)
5. American National Standards Institute: ANSI X9.62-2005, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA) (2005)
6. 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: 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pp. 75–92 (2014)
7. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 129–142. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_11
8. Chen, Y., Nguyen, P.Q.: BKZ 2.0: better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_1
9. 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 (CCS), pp. 1505–1515 (2016)
10. Gruss, D., Maurice, C., Wagner, K., Mangard, S.: Flush+ Flush: a fast and stealthy cache attack. In: 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 279–299 (2016)
11. Hai, H., Ning, N., Lin, X., Zhiwei, L., Bin, Y., Shilei, Z.: An improved wnaf scalar-multiplication algorithm with low computational complexity by using prime precomputation. IEEE Access **9**, 31546–31552 (2021)
12. Hlaváč, M., Rosa, T.: Extended hidden number problem and its cryptanalytic applications. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 114–133. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74462-7_9
13. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. Des. Codes Cryptogr. **23**(3), 283–290 (2001)
14. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP Message Format (RFC 4880) (2007)
15. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001)
16. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**(4), 515–534 (1982)
17. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: IEEE Symposium on Security and Privacy, S&P 2015, pp. 605–622 (2015)
18. Liu, S., Qi, G., Wang, X.A.: Fast and secure elliptic curve scalar multiplication algorithm based on a kind of deformed fibonacci-type series. In: 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pp. 398–402 (2015)
19. De Micheli, G., Piau, R., Pierrot, C.: A tale of three signatures: practical attack of ECDSA with wNAF. In: Nitaj, A., Youssef, A. (eds.) AFRICACRYPT 2020. LNCS, vol. 12174, pp. 361–381. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51938-4_18
20. Möller, B.: Algorithms for multi-exponentiation. In: International Workshop on Selected Areas in Cryptography, pp. 165–180 (2001)

21. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
22. Nguyen, P.Q.: The dark side of the hidden number problem: lattice attacks on DSA. In: Cryptography and Computational Number Theory, pp. 321–330 (2001)
23. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the digital signature algorithm with partially known nonces. J. Cryptol. **15**(3), 151–176 (2002)
24. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Des. Codes Cryptogr. **30**(2), 201–217 (2003)
25. Page, D.: Theoretical use of cache memory as a cryptanalytic side-channel. IACR Cryptol. ePrint Arch. **2002**, 169 (2002)
26. van de Pol, J., Smart, N.P., Yarom, Y.: Just a little bit more. In: The Cryptographers' Track at the RSA Conference (CT-RSA), pp. 3–21 (2015)
27. Rankl, W.: Smart card applications: design models for using and programming smart cards (2007)
28. Schnorr, C.P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Program. **66**(1), 181–199 (1994)
29. Solinas, J.A.: Efficient arithmetic on koblitz curves. Des. Codes Cryptogr. **19**(2), 195–249 (2000)
30. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246) (2008)
31. The FPLLL development team: fplll, a lattice reduction library (2016). https://github.com/fplll/fplll
32. Wang, W., Fan, S.: Attacking OpenSSL ECDSA with a small amount of side-channel information. Sci. China Inf. Sci. **61**(3), 032105:1–032105:14 (2017)
33. 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 (2014)
34. Zhao, S.l., Yang, X.Q., Liu, Z.W., Yu, B., Huang, H.: An improved wnaf scalar-multiplication algorithm with low computational complexity. Acta Electonica Sinica, 1–7 (2022)