Bias from Uniform Nonce: Revised Fourier Analysis-based Attack on ECDSA

Shunsuke Osaki[1][0000−0002−9221−8095] ⋆ and Noboru Kunihiro[2][0000−0003−1822−7476]

[1] NEC Corporation, Tokyo , Japan, shunsuke-osaki@nec.com
[2] University of Tsukuba, Ibaraki, Japan, kunihiro@cs.tsukuba.ac.jp

Abstract. The key finding problem for ECDSA can be reduced to the Hidden Number Problem (HNP) when nonce top bits leak with signatures and hashes. Two main HNP-solving methods exist: lattice-based attacks and Fourier analysis-based attacks. Bleichenbacher's Fourier analysis-based attack can recover keys even if the nonces error rate is high. Aranha et al. (CCS 2020) use a 4-list sum algorithm for linear combinations of samples, which is important in the Fourier analysis-based attack. They evaluated the required number of signatures by assuming a uniform sum distribution in their algorithm. However, the actual distribution was not uniform, leading to an underestimation of the number of signatures. In this study, we derive the exact sum distribution and propose an algorithm incorporating it. Additionally, we introduce a signature reduction algorithm utilizing previously unused pairs. Previous studies assumed biased top nonces bits values by discarding certain samples to bias the nonces intentionally. However, we demonstrate that even unbiased nonces, if their top bits are leaked, enable signatures reduction and secret key recovery without altering execution time using the same algorithm. We show that for any key length, the number of signatures is reduced by 1/2 for 1 bit leakage, and the number of signatures is reduced by 1/4 for 2 or more bits leakage, and we confirm this experimentally for 131-bit ECDSA.

Keywords: ECDSA · Bleichenbacher's Fourier-analysis based attack · 4-list sum algorithm · side-channel attack.

1 Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a digital signature algorithm used in SSH, SSL/TLS, Bitcoin, and other applications. Therefore, it is very important to evaluate how much secret information leakage affects its security. A nonce (Number used only ONCE) is random secret information generated at the time of signing and can be leaked by side-channel attacks. An attack has been proposed that attributes a situation in which a certain number of triples of some bits of the nonces, the corresponding hash value of the message,

⋆ This work was carried out while the first author was at University of Tsukuba.

and the signature are leaked to the Hidden Number Problem (HNP) [6]. Lattice-based and Fourier analysis-based attacks are known as methods for solving HNP.

Lattice-based attacks were considered to be able to solve HNP with a small number of signatures when the top few bits of nonces are known without error. If the key length is 160-bit, the attacker can solve the HNP if the 2 bits nonces are leaked [1, 11, 13], 3 bits for 256-bit [1, 13], 4 bits for 384-bit [1, 13], respectively, with tens to thousands of signatures, it is possible to recover the secret key in a few minutes. Lattice-based attacks require at least 2 bits of error-free nonces information, but do not require a large number of signatures and time. Recently, Geo et al. showed that HNP can be solved with a lattice-based attack against nonce with errors by using many signatures, computational resources and time [9]. They succeeded in recovering the secret key for 128-bit ECDSA with an error rate of 0.1 and 160-bit ECDSA with an error rate of 0.01.

On the other hand, the Fourier analysis-based attack can recover the secret key even if the nonces error rate are high. However, it requires a large amount of computing resources (e.g., workstations), and the computation time can range from several days to a week. In addition, the Fourier analysis-based attack uses the bias of the nonces to recover the secret key, and only the signatures corresponding to the biased nonces are used. The Fourier analysis-based attack is an attack method against DSA first proposed by Bleichenbacher [4]. Next, De Mulder et al. [7] introduced the method in detail and successfully recovered a secret key against 384-bit ECDSA when the top 5 bits of nonce are leaked. Next, Aranha et al. [2] successfully recovered a secret key for the first time for a 160-bit ECDSA when the top 1 bit of the nonces are leaked. Next, Takahashi et al. [14] successfully recovered a secret key for a 252-bit qDSA when the top 2 bits of the nonces are leaked. Recently, Aranha et al. [3] were the first to estimate the modular bias when the top 1 bit of the nonces are leaked with errors, and were the first to successfully recover a 192-bit ECDSA secret key. In addition, Osaki et al. [12] are the first to estimate the modular bias when the top multiple bits of the nonces are leaked with errors. They also show experimentally that the case where each of the 2 bits is leaked with an error rate of 0.1 requires fewer signatures than the case where the 1 bit are leaked with no errors.

In a Fourier analysis-based attack, the computation to take linear combinations of HNP samples is the most time-consuming. For the linear combination, small values should be obtained with a small number of computations. Aranha et al. use the 4-list sum algorithm for linear combinations [3]. They then attribute the problem of optimizing the number of signatures to a linear programming problem and solve it to find the parameters for optimizing the number of signatures, memory, and time required for the attack.

1.1 Our contributions

First, we modify the 4-list sum algorithm used in [3] to perform a linear combination of HNP samples. The sum distribution was assumed to be uniform, but in actuality it is not uniform, so the number of samples obtained was overestimated. In addition, since carry was not taken into account, half of the samples

that should have been obtained were discarded. We modify the algorithm to obtain all samples and estimate the accurate number of signatures, taking into account distribution and carry. In addition, we estimate the number of signatures that can be recovered with an error-free 1 bit leakage and an error rate of 0.11434 and a 2 bit leakage, with an equal number of signatures.

We then show that it is possible to recover the secret key using only HNP samples corresponding to uniform nonces, while all previous studies using Fourier analysis-based attacks used only samples of HNPs corresponding to *biased nonces*. We show that the number of signatures required is reduced to 1/2 for a 1 bit leakage and to 1/4 for a 2 bits leakage. We also show that the number of required signatures decreases to less than 1/4 for 3 bits or more.

Finally, we conduct experiments for uniform nonces and biased nonces. This experiment actually confirms that the number of signatures required to recover the secret key is reduced.

2 Preliminaries

2.1 ECDSA

ECDSA is a digital signature algorithm using elliptic curves. The solution set $(x, y) \in \mathbb{F} \times \mathbb{F}$ and the infinity point O is a commutative group derived from the chord-and-tangent rule. The signature generation algorithm is shown in Algorithm 1. The k randomly generated in the line 1 of Algorithm 1 is a nonce which is used only for signature generation, and a different value is used each time.

Algorithm 1 ECDSA signature generation

Input: Elliptic curve E, prime number q, secret key $\mathrm{sk} \in \mathbb{Z}_q$, message $\mathsf{msg} \in \{0,1\}^*$, base point on elliptic curve G, and cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_q$
Output: A valid signature (r, s)
1: $k \in [1, q-1]$ is selected at random
2: $R = (r_x, r_y) \leftarrow kG; r \leftarrow r_x \bmod q$
3: $s \leftarrow (H(\mathsf{msg}) + r \cdot \mathrm{sk})/k \bmod q$
4: **return** (r, s)

2.2 Hidden Number Problem

The function $\mathrm{MSB}_n(x)$ returns the top n bits of x for a positive integer x. Let b be a positive integer, let χ_b be a fixed distribution on $\{0,1\}^b$, and let the error bit sequence e be sampled from χ_b. The probabilistic algorithm $\mathrm{EMSB}_{\chi_b}(x)$ takes x, b as input and returns $\mathrm{MSB}_b(x) \oplus e$. For each $i = 1, \ldots, M$, let h_i, k_i be uniform random values on $\mathbb{Z}_q$ and z_i be $z_i = k_i - h_i \cdot \mathrm{sk} \bmod q$. HNP is the problem of finding sk that satisfies these given $h_i, z_i, \mathrm{EMSB}_{\chi_b}(k_i)$ obtained at $i = 1, \ldots, M$. HNP was introduced by Boneh et al [6].

The ECDSA signature (r, s) is described in Algorithm 1. The nonce $k \in \mathbb{Z}_q$ is chosen uniformly at random and satisfies $s \equiv (H(\mathsf{msg}) + r \cdot \mathrm{sk})/k \bmod q$.

Therefore, the equation $H(\mathsf{msg})/s \equiv k - (r/s) \cdot \mathrm{sk} \bmod q$ is obtained. As $h \coloneqq r/s \bmod q$ and $z \coloneqq H(\mathsf{msg})/s \bmod q$, we obtain

$$k \equiv z + h \cdot \mathrm{sk} \bmod q. \tag{1}$$

Let M be the number of signatures and $\{(h_i, z_i)\}_{i=1}^{M}$ is obtained. It is a system of simultaneous linear equations with $(M+1)$ unknowns and cannot be solved because it is indefinite, and the secret key sk cannot be obtained. If MSBs of k_i can be obtained by vulnerability, etc., then an instance of HNP is obtained.

2.3 Bleichenbacher's Fourier analysis-based attack

In this section, we explain the Fourier analysis-based attack, which is one of the methods for solving HNP. First, we explain the bias function, which is important in the Fourier analysis-based attack. Next, a naive algorithm using the bias function is presented. We then present a framework including linear combinations to search for secret keys efficiently.

We define the bias function needed to find the secret key in this attack.

Definition 1. *Let $\boldsymbol{K}$ be a random variable on $\mathbb{Z}_q$. The modular bias $B_q(\boldsymbol{K})$ is defined as*

$$B_q(\boldsymbol{K}) = \boldsymbol{E}\left[\exp\left((2\pi \boldsymbol{K}/q)\,\mathrm{i}\right)\right].$$

Where $\boldsymbol{E}(\boldsymbol{K})$ denotes the mean and i *is the imaginary unit. The sample bias of the set of points $K = \{k_i\}_{i=1}^{M}$ on $\mathbb{Z}_q$ is defined as*

$$B_q(K) = \frac{1}{M}\sum_{i=1}^{M} \exp\left((2\pi k_i/q)\,\mathrm{i}\right). \tag{2}$$

The bias function is the average of these vectors when the nonces are considered to be vectors over the unit circumference. The larger the number of top bits fixed in the nonce, the larger the absolute value of the average of the vectors. The bias function satisfies the properties shown in Lemma 1 [7, 14]. For a positive integer l and a λ-bit integer q, it is proved that when the top l bits of $\boldsymbol{K}$ are fixed to some constant and the remaining $(\lambda - l)$ bits are random, the following equation is proved in [14].

$$\lim_{q\to\infty} |B_q(\boldsymbol{K})| = \frac{2^l}{\pi} \cdot \sin\left(\frac{\pi}{2^l}\right) \tag{3}$$

It can be seen that $\lim_{l\to\infty}\lim_{q\to\infty}|B_q(\boldsymbol{K})| = 1$ in equation (3). If all bits are not fixed and are random, the absolute value of bias is $1/\sqrt{M}$ [14].

Aranha et al. show a modular bias when only the MSB contains errors in Lemma 4.2 [3]. Extending their Lemma 4.2, the modular bias when multiple bits of nonces contain errors is shown in Theorem 1 in [12]. In addition, the absolute value of the modular bias when multiple bits leak with errors is expressed as follows in [12].

Lemma 1. *Let ε_j be the error rate of the top j-th bit, the absolute value of the modular bias is given by*

$$\prod_{j=1}^{l} \sqrt{\left(1-2\varepsilon_j\right)^2 \sin\frac{\pi}{2^j} + \cos^2\frac{\pi}{2^j}} \left|B_q\left(\boldsymbol{K}\right)\right|. \tag{4}$$

We will describe a naive key search method with the bias function. The inputs are HNP samples $\{(h_i, z_i)\}_{i=1}^{M}$ corresponding to nonces whose top bits are biased. We compute $K_w = \{z_i + h_i w \bmod q\}_{i=1}^{M}$ for each $w \in [1, q-1]$ which is a candidate secret key. Since the nonces are biased, the w which maximizes $|B_q(K_w)|$ is the secret key. This method is inefficient because it requires searching all candidates.

The Fourier analysis-based attack algorithm of [3] is shown in Algorithm 2. In this algorithm, linear combinations of samples are used to search for the secret key efficiently, and this phase is called collision search. Except for the [3] and [4], this phase is called range reduction. Collision search has two constraints: small linear combinations, which reduces the value by taking linear combinations, and sparse linear combinations, which limits the number of linear combinations.

The first constraint is small linear combinations. In this constraint, it should be satisfied that $h'_j = \sum_i \omega_{i,j} h_i < L_{\mathrm{FFT}}$ where $\omega_{i,j} = \{-1, 0, 1\}$. This constraint reduces the search range from q to L_{FFT}. Let M' be the number of samples after linear combination, and the linear combinations generate $\left\{\left(h'_j, z'_j\right) = \left(\sum_i \omega_{i,j} h_i, \sum_i \omega_{i,j} z_i\right)\right\}_{j=1}^{M'}$ for each of h, z. Now, from the values of $\omega_{i,j}$, $\Omega_j \coloneqq \sum_i |\omega_{i,j}|$ is the number of linear combinations in h'_j.

The second constraint is sparse linear combinations. This constraint prevents Ω_j, the number of linear combinations, from becoming large. Small linear combinations can be easily obtained by taking a large number of linear combinations. If all the coefficients of linear combinations are restricted to $\{-1, 0, 1\}$, the peak bias decreases exponentially with the L_1 norm of the coefficient vector.

Thus, the absolute value of the bias decreases exponentially as $|B_q(K)|^{\Omega_j}$. Therefore, since the peak can be observed if it is larger than $1/\sqrt{M'}$, the average of the absolute value of the bias, we impose the constraint $|B_q(K)|^{\Omega_j} \gg 1/\sqrt{M'}$. Bleichenbacher [4] and De Mulder et al. [7] pointed out that the peak value of the modular bias, and Takahashi et al. [14] show in the next lemma that it decreases exponentially.

Lemma 2. *Let $\boldsymbol{K}_i$ be a random variable with uniform distribution on the interval $\left[0, \left\lfloor (q-1)/2^l \right\rfloor\right]$ corresponding to nonces k_i. If $i_1 \neq i_2$, then $\boldsymbol{K}_{i_1}, \boldsymbol{K}_{i_2}$ are independent,*

$$\left| B_q\left(\sum_i \omega_i \boldsymbol{K}_i\right)\right| = \left|B_q\left(\boldsymbol{K}\right)\right|^{\Omega}.$$

If the peak width is extended by linear combinations, it is impossible to find the entire secret key in a single Fourier analysis-based attack. Let λ' be the

Algorithm 2 Bleichenbacher's attack framework

Input: $\{(h_i, z_i)\}_{i=1}^{M}$: HNP samples on $\mathbb{Z}_q$ with biased nonces, M': Number of linear combinations to be found, L_{FFT}: FFT table size
Output: Top bits of sk
1: **Collision Search**
2: Generate $\left\{\left(h'_j, z'_j\right)\right\}_{j=1}^{M'}$, for $j \in [1, M']$, the coefficients $\omega_{i,j} \in \{-1, 0, 1\}$, and the linear combination pairs are denoted as $\left(h'_j, z'_j\right) = \left(\sum_i \omega_{i,j} h_i, \sum_i \omega_{i,j} z_i\right)$.

(1) Small: $0 \leq h'_j < L_{\mathrm{FFT}}$
(2) Sparse: $|B_q(\boldsymbol{K})|^{\Omega_j} \gg 1/\sqrt{M'}$ for all $j \in [1, M']$, where $\Omega_j \coloneqq \sum_i |\omega_{i,j}|$.

3: **Bias Computation**
4: $Z \coloneqq (Z_0, \ldots, Z_{L_{\mathrm{FFT}}-1}) \leftarrow (0, \ldots, 0)$
5: **for** $j = 1$ to M' **do**
6: $\quad Z_{h'_j} \leftarrow Z_{h'_j} + \exp\left(\left(2\pi z'_j/q\right)\mathrm{i}\right)$
7: **end for**
8: Let $w_i = iq/L_{\mathrm{FFT}}$,$\{B_q(K_{w_i})\}_{i=0}^{L_{\mathrm{FFT}}-1} \leftarrow \mathrm{FFT}(Z)$
$= \left(B_q(K_{w_0}), B_q(K_{w_1}), \ldots, B_q\left(K_{w_{L_{\mathrm{FFT}}-1}}\right)\right)$
9: Find i that maximizes $|B_q(K_{w_i})|$
10: **return** Top $\log L_{\mathrm{FFT}}$ bits of w_i

number of recovered bits of the secret key sk in a single Fourier analysis-based attack, where $\lambda' = \log L_{\mathrm{FFT}}$. According to [3], $\log L_{\mathrm{FFT}} - 4$ bits can always be recovered empirically. By repeating this attack, the entire secret key can be obtained. For details, see Section 5.2 in [3].

Bias computation is performed using FFT with a time complexity of $O(L_{\mathrm{FFT}}(\log L_{\mathrm{FFT}}))$. Therefore, collision search is the most time-consuming computation in a Fourier analysis-based attack. So, it is very important to improve the collision search algorithm and to make accurate estimates.

2.4 4-list sum algorithm

Let the birthday problem be the problem of choosing random n bits elements $x_1 \in \mathcal{L}_1$ and $x_2 \in \mathcal{L}_2$ from 2 lists $\mathcal{L}_1$ and $\mathcal{L}_2$ that satisfy $x_1 \oplus x_2 = 0$. In addition, given $\mathcal{K}$ lists with n bits values, the problem of selecting 1 of elements from each list and finding $\mathcal{K}$-tuples of values for which the XOR of those $\mathcal{K}$ values is 0. This problem is called Generalized Birthday Problem (GBP). Wagner [15] noted similarities between GBP and Bleichenbacher's attack [4].

From the two constraints of the Fourier analysis-based attack, it is desirable to take small values with fewer linear combinations efficiently. As a method to take such an efficient linear combination, Aranha et al. adopt the 4-list sum algorithm [3]. 4-list sum algorithm is for $\mathcal{K} = 4$ in GBP.

The 4-list sum algorithm is shown in Algorithm 3. In (a) in the algorithm, the number of elements in each of $\{\mathcal{L}_i\}_{i=1}^{4}$ is 2^a, and since the distribution of $x_1 + x_2$ is uniform, the number of samples output to $\mathcal{L}'_1, \mathcal{L}'_2$ is given by $2^a \cdot 2^a \cdot 2^{-a} = 2^a$ regardless of c value.

Algorithm 4 shows the algorithm that repeats Algorithm 3. The r variable represents the number of iterations of the 4-list sum algorithm. Using this algorithm, $\Omega_j = 4^r$ can be expressed in Algorithm 2. In this study, we will only experiment with the case $r = 2$.

Algorithm 3 Parameterized 4-list sum algorithm based on Howgrave-Graham–Joux

Input: $\{\mathcal{L}_i\}_{i=1}^{4}$: sorted lists of uniform random samples of λ-bit of length 2^a, n: number of top bits to be discarded in each round, $v \in [0, a]$: parameter
Output: $\mathcal{L}'$: list of $(\lambda - n)$ bits samples
1: For each $c \in [0, 2^v)$:

(a) Find pairs, $(x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2$, satisfying $\mathrm{MSB}_a(x_1 + x_2) = c$. Output a new sorted list $\mathcal{L}'_1$ with $2^a \cdot 2^a \cdot 2^{-a} = 2^a$ elements of $x_1 + x_2$. Similarly, for $\mathcal{L}_3, \mathcal{L}_4$, the sorted list $\mathcal{L}'_2$ is obtained.
(b) Find pairs, $(x'_1, x'_2) \in \mathcal{L}'_1 \times \mathcal{L}'_2$, satisfying $\mathrm{MSB}_n(|x'_1 - x'_2|) = 0$. Output a new sorted list $\mathcal{L}'$ with $|x'_1 - x'_2|$ as $2^a \cdot 2^a \cdot 2^{-(n-a)} = 2^{3a-n}$ elements.

2: **return** $\mathcal{L}'$ with $M' = 2^{3a+v-n}$ elements

Algorithm 4 shows the algorithm that repeats Algorithm 3. The r variable represents the number of iterations of the 4-list sum algorithm. Using this algorithm, $\Omega_j = 4^r$ can be expressed in Algorithm 2. In this study, we will only experiment with the case $r = 2$.

Algorithm 4 Iterative HGJ 4-list sum algorithm

Input: $\mathcal{L}$: list of $M = 4 \times 2^a$ uniformly random λ-bit samples, $\{n_i\}_{i=0}^{r-1}$: number of top bits to be discarded in each round, $\{v_i\}_{i=0}^{r-1}$: parameters $v_i \in [0, a_i]$
Output: $\mathcal{L}'$: list of $\left(\lambda - \sum_{i=0}^{r-1} n_i\right)$ bits in the sample whose length is 2^{m_r}
1: Let $a_0 = a$
2: For each $i = 0, \ldots, r-1$:

(a) Divide $\mathcal{L}$ into four lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4$ of length 2^{a_i} and sort each list.
(b) Apply Algorithm 3 to $\{\mathcal{L}_i\}_{i=1}^{4}$ with parameters n_i and v_i. Obtain a single list $\mathcal{L}'$ of length $2^{m_{i+1}} = 2^{3a_i + v_i - n_i}$. Let $\mathcal{L} \coloneqq \mathcal{L}'$. Let $a_{i+1} = m_{i+1}/4$.

3: **return** $\mathcal{L}'$

In Theorem 4.1 of [3], it is proved that the following holds as $N = 2^n, T = 2^t, M = 2^m = 2^{a+2}$.

$$2^4 M' N = T M^2 \tag{5}$$

$$m' = 3a + v - n \tag{6}$$

Using this relationship, they formulate a linear programming problem that optimizes time, memory, and number of signatures, respectively, and summarized

in Table 2 in [3]. In addition, they used this linear programming problem for optimization to successfully recover the secret key.

3 Modification of the 4-list sum algorithm and estimation of the number of its outputs

The number of elements of $\mathcal{L}'_1, \mathcal{L}'_2$ is not 2^a in (a) of Algorithm 3 in [3]. This is caused by the fact that the analysis is based on the assumption that the distribution of the sum is uniform, even though it is not. First, we propose a modification algorithm to make the distribution of the sums uniform. However, this modification method has the problem of consuming a large amount of computer resources. Therefore, since the distribution does not necessarily need to be uniform, we propose a modification in Section 3.6 that takes advantage of the non-uniform distribution, based on the algorithm of [3].

3.1 Technical Issues with the algorithm in [3]

Aranha et al. [3] used the algorithm introduced by Dinur [8]. Dinur's algorithm computes as a xor $\oplus$, while Aranha et al.'s algorithm computes as an addition $+$. Due to this difference, the algorithm of [3] does not give the estimated results. In this section, we first point out the problems in the algorithm of [3], and then the following section shows a simple modification method and the problems of this modification method.

Consider the following small example of a 4-list sum algorithm.

Example 1. Consider the Algorithm 3. Let $l = 5, n = 4, a = 2$ and then let $x_1 = 17\,(1\,0001), x_2 = 18\,(1\,0010), x_3 = 15\,(1111), x_4 = 17\,(1\,0001)$. In (a), $x'_1 = 35\,(10\,0011), x'_2 = 32\,(10\,0000)$, and $\mathrm{MSB}_2\,(x'_1) = \mathrm{MSB}_2\,(x'_2) = 2\,(10)$. In (b), $\mathrm{MSB}_4\,(|x'_1 - x'_2|) = 0$ and $|x'_1 - x'_2| = 3\,(11)$.

Since $l - n = 1$, the output result is expected to be less than 1 bit, but it turns out to be 2 bits. Also, the $\mathrm{MSB}_4\,(|x'_1 - x'_2|)$ is trying to return the top 4 bits, while $|x'_1 - x'_2|$ is 2 bits, and the behavior of the function $\mathrm{MSB}_n\,(x)$ is ambiguous.

3.2 Modifying definition of function $\mathrm{MSB}_n\,(x)$

In [3], $\mathrm{MSB}_n\,(x)$ is defined to return the top n bits of x. In the 4-list sum algorithm we are considering now, x includes values of less than n bits, so the top n bits of x contain ambiguity. In addition, there is a problem that the algorithm may output a value with an unintended bit length. To solve these problems, we introduce a new function $\mathrm{MSB}_{l,n}\,(x)$, which returns the $(l - n)$ bit logical right shifted value of x.

The function $\mathrm{MSB}_{l,n}\,(x)$ introduced above solves the problem considered in Example 1. With $\mathrm{MSB}_{l,n}\,(x)$, in (a) the result is $\mathrm{MSB}_{5,2}\,(x'_1) = \mathrm{MSB}_{5,2}\,(x'_2) = 4\,(100)$. In (b), the result of the computation is $\mathrm{MSB}_{5,4}\,(|x'_1 - x'_2|) = 1\,(1) \neq 0$, so the 1 bit output is obtained. Since the output result is not a combination of solutions that satisfy the condition, it is excluded from the candidates.

3.3 Output range of 4-list sum algorithm

As can be seen from Example 1, a value of 2 bits is obtained when the value should be less than 1 bit. This is because the addition of a bits may result in $(a+1)$ bits. The following theorem shows the probability that the sum of a digits or less in basis b is $(a+1)$ digits.

Theorem 1. *Let a and $b \geq 2$ be positive integers and n be a positive integer such that $\lceil b^a/2 \rceil \leq n-1 < b^a$. The probability that the sum of $x_1, x_2 \in [0, n-1]$ in basis b is $(a+1)$ digits is given by*

$$\frac{(2n - b^a - 1)(2n - b^a)}{2n^2}$$

Proof. Let $x_1, x_2 \in [0, n-1]$. When x_1 is fixed, the condition for $x_1 + x_2$ to be $(a+1)$ digits is $x_2 \geq b^a - x_1$. The interval of x_2 satisfying the condition is $[b^a - x_1, n-1]$ and there are $(n - b^a + x_1)$ as x_2. Also, the smallest x_1 at which the carry occurs is $b^a - n + 1$. Therefore,

$$\sum_{x_1 = b^a - n + 1}^{n-1} (n - b^a + x_1).$$

Since there are n^2 possible combinations of adding x_1, x_2, the probability of $(a+1)$ digits is

$$\frac{\sum_{x_1 = b^a - n + 1}^{n-1} (n - b^a + x_1)}{n^2} = \frac{(2n - b^a - 1)(2n - b^a)}{2n^2}.$$

In Theorem 1, if $n-1$ is the maximum of a digits, i.e., $n - 1 = b^a - 1$, then $(1 - 1/b^a)/2 = (1 - 1/n)/2$ is obtained, showing that it is carried forward with about half probability. From this, the size of $\mathcal{L}'_1, \mathcal{L}'_2$ in (a) of Algorithm 3 is $2^a \cdot 2^a \cdot 2^{-(a+1)} = 2^{a-1}$. The number of elements output to $\mathcal{L}'$ in (b) is $2^{a-1} \cdot 2^{a-1} \cdot 2^{-(n-a)} = 2^{3a+n-2}$, and $M' = 2^{3a+v-n-2}$. Therefore, the number of elements in the output is $1/4$ compared to the estimate in [3].

Considering Theorem 1, the range of v in Algorithm 3 can be modified from $v \in [0, a]$ to $v \in [0, a+1]$. This makes $M' = 2^{4a-n-1}$ only when $v = a+1$, and the number of output elements becomes $1/2$ compared to the estimation in [3].

3.4 Number of outputs of 4-list sum algorithm

In Theorem 4.1 of the [3], the equations (5) and (6) are presented. In the proof, (a) of Algorithm 3 assumes that it is possible to find 2^a pairs whose sum is c for each c. However, in reality, only $c+1$ pairs can be obtained when $v \in [0, a]$. Furthermore, $\mathrm{MSB}_{l,a}(x_1 + x_2)$ can be $(a+1)$ bit as considered in Theorem 1. Let $v \in [0, a+1]$, allowing $(a+1)$ bits in c, there are $2^{2a+1} - c - 1$ pairs of combinations for which the sum in the range $c \in \left[2^a, 2^{a+1} - 1\right)$ is c. The following theorem shows the number of combinations in these two cases.

Theorem 2. *For a non-negative integer A, let x_1, x_2 be non-negative integers $x_1, x_2 \leq A$. For $c \leq A$, there are $c+1$ pairs of (x_1, x_2) such that $x_1 + x_2 = c$. In the case of $A \leq c \leq 2A-1$, there are $2A-c-1$ pairs of (x_1, x_2) such that $x_1 + x_2 = c$.*

Proof. Consider the case $c \leq A$. From $x_1 + x_2 = c$, there always exists one x_2 corresponding to x_1. Since $x_1, x_2 \in [0, c]$, there are $c+1$ pairs of x_1, x_2 such that $x_2 = c - x_1$. Therefore, there are $c+1$ pairs of (x_1, x_2) whose sum is c.

Next, consider the case $A \leq c \leq 2A-1$. Since $x_1+x_2 = c$, there is exactly one corresponding x_2, if there exists an x_1 satisfying the condition. Since $x_1, x_2 \in [c-A, A]$, there exist $A - (c-A) + 1 = 2A - c + 1$ pairs of x_1, x_2 such that $x_2 = c - x_1$. Therefore, there are $2A - c + 1$ pairs of (x_1, x_2) that sum to c.

When applying Theorem 2 to Algorithm 3, we can set $A = 2^a - 1$. In [3], the number of elements in the list does not depend on the parameter c, but this theorem shows that it does depend on c. In Algorithm 3, [3] claims that 2^{20} pairs are output for any c when $a = 20$, but only 1 pair is output when $c = 0$.

Based on Theorem 2, Figure 1 shows the distribution of elements output in (a) of Algorithm 3 for the case $a = 5$. In [8], which was referred to in applying the 4-list sum algorithm in [3], xor is used instead of addition. In the case of xor, the distribution is uniform as shown in Figure 2. In [3], the estimate was based on the assumption that the distribution would be Figure 2, but the actual distribution is not uniform as in Figure 1. The number of collisions in (a) is shown in Appendix A.

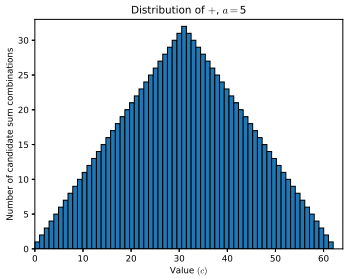

Fig. 1. Distribution of combinations of sums in the case $a = 5$.

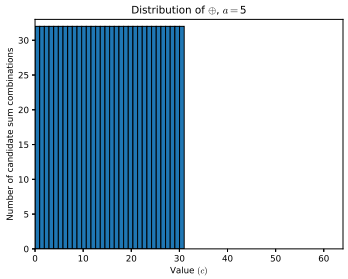

Fig. 2. Distribution of combinations of xor in the case of $a = 5$.

3.5 Simple modification of the algorithm

In this section, we show the modification to make the distribution uniform so that the carry does not occur and the sum does not become $(a+1)$ bits. GBP perform addition on $\mathbb{Z}_q$ instead of $\oplus$ in [5, 10], which also results in a uniform distribution.

In (a) of the algorithm, the computation as modulo q allows pairs not used by carry to be included in the $\mathcal{L}'_1, \mathcal{L}'_2$ as elements satisfying the condition, and they are uniformly distributed. This modification does not affect bias computation in Fourier analysis-based attacks. A simple modification is to output the element $(x_1 + x_2) \bmod q$ in list $\mathcal{L}'_1$ with $\mathrm{MSB}_a\left((x_1 + x_2) \bmod q\right) = c$ as the condition to be satisfied in (a) of the Algorithm. Similarly for $\mathcal{L}_3, \mathcal{L}_4$. This modification allows the algorithm to output 2^a elements in (a) without $(a+1)$ bits due to carry. The distribution of the output is uniform, and for $a = 5$, the distribution is as shown in Figure 2. Therefore, the linear programming problem presented by Aranha et al. can be applied [3].

3.6 Improved 4-list sum algorithm using the distribution of sums

In the implementation of the 4-list sum algorithm in [3], $x_1 + x_2, x_3 + x_4$ are not output to L'_1, L'_2 in order to save memory. Their implementation does not keep all the bits in each list, which allows it to run with less memory. First, only the top 127 bits of the original x_i are output to $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4$, and the original samples are stored in ROM to reduce memory consumption. Next, in (a) of the algorithm, find pairs (x_1, x_2) satisfying $\mathrm{MSB}_{127,a}\left(x_1 + x_2\right) = c$ and store the triplet consisting of the indices of x_1 and x_2 and the value of $x_1 + x_2$ in $\mathcal{L}'_1$. Similarly, in (b) of the algorithm, find pairs $\left(\left(x_1 + x_2, \mathrm{index}_{x_1}, \mathrm{index}_{x_2}\right), \left(x_3 + x_4, \mathrm{index}_{x_3}, \mathrm{index}_{x_4}\right)\right) \in \mathcal{L}'_1 \times \mathcal{L}'_2$ satisfying $\mathrm{MSB}_{127,n}\left(|x'_1 - x'_2|\right) = 0$ and store the quadruplet indices $(\mathrm{index}_{x_1}, \mathrm{index}_{x_2}, \mathrm{index}_{x_3}, \mathrm{index}_{x_4})$ in $\mathcal{L}'$. Finally, the samples are read from ROM and the result of the linear combinations are computed from the indices list $\mathcal{L}'$. This implementation keeps only 127 bits of each value of h in memory while searching for collisions, no matter how long the key length is. In addition, a linear combination of h and z must be taken simultaneously, but using a list of indices, only h needs to be kept in RAM during collision search.

Algorithm for simple modifications requires that each value of x_1, x_2, x_3, x_4 be kept in all-bit memory for the modulo calculation in (a) in the algorithm. Therefore, the next section considers an algorithm to increase the number of samples after linear combinations without modulo to save memory.

In Section 3.3, we confirmed that the number of samples output is not uniform depending on the value of c. The range of c in [3]'s Algorithm 3 with c selected is shown in Figure 3. Since their algorithm assumes uniformity, c is selected in the range from 0 to $2^v - 1$, and the figure clearly shows that they select a small range. In this section, we seek the optimal interval of c that maximizes output count, while maintaining a width of 2^v for non-uniformity.

For $c \in [p, p + 2^v)$, the number of samples output after one round of 4-list sum algorithm is

$$\sum_{i=p}^{\min(2^a-1,\,p+2^v-1)} (i+1) + \sum_{i=2^a-1}^{\max(2^a-1,\,p+2^v-1)} \left(-i + 2^{a+1} - 1\right).$$

Finding p that maximizes the above equation. $p = 2^a - 1 - 2^{v-1}, 2^a - 2^{v-1}$ maximize it to $2^{v-1}\left(2^{a+1} - 2^{v-1}\right)$. This is also clear from Figure 4, which shows that it is sufficient to take an interval from the center, 2^a, to the left and right.

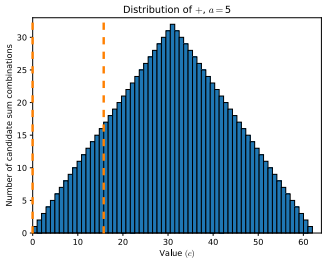

Fig. 3. Range of c in [3] ($v = 4$).

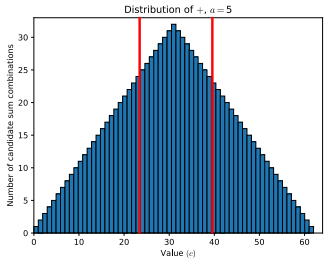

Fig. 4. Range of optimal c ($v = 4$).

From the range of c obtained, the number of samples output after one round of the 4-list sum algorithm is less than the estimate of [3] as follows.

$$\begin{aligned} M' &= \left(2 \sum_{c=2^a-2^{v-1}}^{2^a-2} (c+1)^2 + \left(2^a - 2^{v-1}\right)^2\right) 2^{-(n-a)} + 2^{3a-n} \\ &= \left(2^{2a+v} - 2^{a+2v-1} + \frac{2^{3v-2}}{3} - 2^{2v-2} + \frac{7 \cdot 2^v}{6}\right) 2^{-(n-a)} \\ &< 2^{3a+v-n} \end{aligned} \tag{7}$$

Algorithm 5 shows the modified algorithm based on these modifications. From Theorem 2, let the function $f(c)$ which returns the number of combinations of sums be

$$f(c) = \begin{cases} c+1 & \text{if } c \leq 2^a - 1 \\ 2^{a+1} - c - 1 & \text{otherwise} \end{cases}$$

Note that in (b), $\{f(c)\}^2 \cdot 2^{-(n-a)} < 1$ and the number of elements output to $\mathcal{L}'$ may be 0 depending on the possible range of c.

3.7 Experiments and evaluations on distribution

We experimented to confirm that the number of samples output differs depending on the range chosen for c. Comparison experiments are performed by applying the original Algorithm 3 and the improved Algorithm 5 to 60-bit ECDSA. Table 1 shows the parameters and the experimental results. To check the distribution, it is not necessary to recover the key, and it is sufficient to confirm that the number of samples output does not depend on a. Therefore, the key length can be any number of bits, and in this experiment, we will use 60-bit.

Algorithm 5 Modified: Parameterized 4-list sum algorithm based on Howgrave-Graham–Joux

Input: $\{\mathcal{L}_i\}_{i=1}^{4}$: sorted lists of uniform random samples of λ of length 2^a, n: number of top bits to be discarded in each round, $v \in [0, a+1]$: parameter, λ: parameter
Output: $\mathcal{L}'$: list of $(\lambda - n)$ bits in the sample

1: **if** $v \neq a+1$ **then**
2: $\quad \gamma = 2^a - 2^{v-1} - 2, \delta = 2^a + 2^{v-1} - 1$
3: **else**
4: $\quad \gamma = 0, \delta = 2^v$
5: **end if**
6: For each $c \in [\gamma, \delta)$:

(a) Find pairs $(x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ satisfying $\mathrm{MSB}_{\lambda,a}(x_1 + x_2) = c$. Out put a new sorted list $\mathcal{L}_1'$ with $(c+1)$ elements of $x_1 + x_2$. Similarly, for $\mathcal{L}_3, \mathcal{L}_4$, the sorted list $\mathcal{L}_2'$.
(b) Find pairs $(x_1', x_2') \in \mathcal{L}_1' \times \mathcal{L}_2'$ satisfying $\mathrm{MSB}_{\lambda,n}(|x_1' - x_2'|) = 0$. Out put a new sorted list $\mathcal{L}'$ with $|x_1' - x_2'|$ as $\{f(c)\}^2 \cdot 2^{-(n-a)}$ elements.

7: **return** $\mathcal{L}'$ with $M' = \left\{2\sum_{c=2^a-2^{v-1}}^{2^a-2}(c+1)^2 + \left(2^{2a} + 2^v - 2^{a+v}\right) + 2^{2a}\right\} 2^{-(n-a)}$.

Table 1. Parameters and results of the experiment

Parameter	a_0	v_0	n_0	a_1	v_1	n_1	Original M'	M' after modification
$l = 1, \varepsilon = 0$	8	5	14	14	2	16	0	$2^{29.43}$
$l = 2, \varepsilon = 0.1$	8	5	15	14	2	15	0	$2^{27.34}$

In the original algorithm, the secret key could not be recovered due to the $M' = 0$, but it could be recovered in the modified algorithm. The reason for the small number of samples in the original algorithm is that $v_1 = 2$, which means that only the sums of combinations $0, 1, 2, 3$ are considered, and the number of pairs satisfying these conditions is small. Thus, the number of samples obtained by the algorithm of [3] becomes 0 when the value of v is extremely small with respect to a. The modified algorithm succeeded in recovering the secret key because a sufficient number of samples were obtained.

3.8 Equivalent ε for $l = 2$ to $l = 1, \varepsilon = 0$

Consider the case where nonce is leaking with multi-bit errors. An estimate if the 2 bits leaked with an error rate of 0.1 could recover the secret key with fewer signatures than if the 1 bit nonce was leaked without error was obtained. [12]. We can find the error rate in Table 2 in [3] so that $m_r = 2(\log \alpha - 4^r \log(|B_q(\boldsymbol{K})|))$ are equal. Details on α are given in Appendix C of [3], and if 1 bit contain error, $\alpha = \sqrt{2\ln(2L_{\mathrm{FFT}}/\varepsilon)}$. If l bits contain errors, $\alpha = \sqrt{2\ln\left(2L_{\mathrm{FFT}}/\left(1-(1-\varepsilon)^l\right)\right)}$.

Consider the case $r = 2$. The error rate that makes m_r equal to $l = 1, \varepsilon = 0$ is $\varepsilon = 0.11$ for $l = 2$. In both cases, $m_2 \approx 27.0740$. Here, since the error rate for each bit is 0.11, we have Note that the error rate at the 2 bit is $1 - (1 - 0.11)^2 \approx 0.21$.

4 New Fourier analysis-based attack for uniform nonces

All previous studies [2, 3, 7, 12, 14] on Fourier analysis-based attacks, while avoiding all searches by linear combinations, used only biased nonces, and only the number of signatures actually used in the attacks were given as estimates. However, when the nonces are generated uniformly at random, several times as many signatures and nonces leak are required to obtain signatures corresponding to biased nonces if the previously proposed methods are applied. Therefore, the signatures that could be used for the attack would be a fraction of the total, and many signatures would have been wasted. For example, if only samples in which the top 2 bits of the nonces are 01 are used when 2 bits are leaked, then samples in which the remaining top 2 bits are $00, 10, 11$ are not used. In other words, $3/4$ of the signatures collected are not used, so a successful attack requires 4 times the number of signatures to be collected.

In this section, we show that by using all signatures without wasting, the number of signatures decreases to $1/2$ in the case of a 1 bit leak and $1/4$ in the case of a 2 bits leak in the same execution time as before, and the secret key is successfully recovered. Furthermore, we show that when 3 or more l bits are leaked, the time remains the same when the number of signatures is reduced to $1/4$, while the number of signatures required decreases by spending more time when the number of signatures decreases by $1/2^{(l+6)/4}$.

4.1 Trick of our new attacks

The absolute value of the bias function increases when there is a bias in $\{k_i\}_{i=1}^{M}$. Fourier analysis-based attacks can reduce the number of key searches by linear combinations. Linear combinations generate $\left\{\left(h'_j, z'_j\right)\right\}_{j=1}^{M'}$ form $\{(h_i, z_i)\}_{i=1}^{M}$. In this case, linear combinations are also performed for the corresponding set of nonces $\{k_i\}_{i=1}^{M}$, and there exist triplets $\left\{\left(k'_j, h'_j, z'_j\right)\right\}_{j=1}^{M'}$. Here, we have a new problem of solving HNP for the samples. That is, if $\left\{\mathrm{MSB}_l\left(k'_j\right)\right\}_{j=1}^{M'}$ are biased, it can be solved by a Fourier analysis-based attack. In the Algorithm 2, this is done by bias computation after linear combination. Therefore, the nonces $\{k_i\}_{i=1}^{M}$ of the HNP samples itself need not necessarily be biased; it is sufficient if $\left\{k'_j\right\}_{j=1}^{M'}$ corresponding to the samples after linear combinations are biased.

The next theorem shows that the peak value is identical when the constraint that $\boldsymbol{K}$ is uniformly distributed over some interval is removed in the Lemma 2.

Theorem 3. *Let l be the number of leakage bits and b the positive integer $b \in \left[0, 2^l - 1\right]$. Let $k_{b,i}$ be the nonces whose top l bits are b and $\boldsymbol{K}_{b,i}$ be a random variable uniform distributed overt the interval $\left[b\lfloor(q-1)/2^l\rfloor, (b+1)\lfloor(q-1)/2^l\rfloor\right]$ corresponding to nonce $k_{b,i}$. Let $\boldsymbol{K}$ be a uniformly distributed random variable on the interval $\left[0, \lfloor(q-1)/2^l\rfloor\right]$. Assuming that $\omega_{b,i}$ takes only $\{-1, 0\}$ or $\{0, 1\}$*

values depending on the value of b, then

$$\left|B_q\left(\sum_b\sum_i \omega_{b,i}\boldsymbol{K}_{b,i}\right)\right| = \left|B_q\left(\boldsymbol{K}\right)\right|^{\Omega}.$$

Proof. Let $i_b^+ \in \{i|\omega_{b,i}=1\}, i_b^- \in \{i|\omega_{b,i}=-1\}$ and $\Omega \coloneqq \sum_b\sum_i|\omega_{b,i}|$, then

$$B_q\left(\sum_b\sum_i \omega_{b,i}\boldsymbol{K}_{b,i}\right) = B_q\left(\sum_b\sum_{i_b^+}\boldsymbol{K}_{b,i^+} - \sum_b\sum_{i_b^-}\boldsymbol{K}_{b,i_b^-}\right)$$

$$= B_q\left(\sum_b\sum_{i_b^+}\boldsymbol{K}_{b,i_b^+}\right)\overline{B_q\left(\sum_b\sum_{i_b^-}\boldsymbol{K}_{b,i_b^-}\right)} = \prod_b\prod_{i_b^+}B_q\left(\boldsymbol{K}_{b,i_b^+}\right)\prod_b\prod_{i_b^-}\overline{B_q\left(\boldsymbol{K}_{b,i_b^-}\right)}$$

$$\left|B_q\left(\sum_b\sum_i \omega_{b,i}\boldsymbol{K}_{b,i}\right)\right| = \left|\prod_b\prod_{i_b^+}B_q\left(\boldsymbol{K}_{b,i_b^+}\right)\prod_b\prod_{i_b^-}\overline{B_q\left(\boldsymbol{K}_{b,i_b^-}\right)}\right| = \left|B_q\left(\boldsymbol{K}\right)\right|^{\Omega}$$

Theorem 3 shows that the peak value after linear combinations is equal to Lemma 2, even when the top bits of nonces are uniform.

4.2 Distribution after linear combinations

Previous studies have not discussed the distribution after linear combinations. In this section, we discuss the distribution after linear combinations. Then, we will show the graphical representations in the next section.

Generalize for leaks of more than 2 bits. Let l be the number of leakage bits and $b \in \left[0, 2^l-1\right]$. Let $\boldsymbol{K}_b$ be a random variable that is uniformly distributed on the interval $\left[b\lfloor(q-1)/2^l\rfloor, (b+1)\lfloor(q-1)/2^l\rfloor\right]$ corresponding to nonces. Let $\alpha \in \left[0, 2^l-1\right]$ and $\boldsymbol{K}_\alpha^0, \ldots, \boldsymbol{K}_\alpha^{2^l-1}$ be independent random variables. In the previous studies, the linear combinations were taken only for random variables in equal intervals, so the random variable after the linear combinations is $\boldsymbol{K}' = \boldsymbol{K}_\alpha^0+\boldsymbol{K}_\alpha^1-\boldsymbol{K}_\alpha^2-\boldsymbol{K}_\alpha^3$. The combinations of random variables in different intervals are $\boldsymbol{K}'' = \boldsymbol{K}_{\alpha_0}+\boldsymbol{K}_{\alpha_1}-\boldsymbol{K}_{\alpha_2}-\boldsymbol{K}_{\alpha_3}$, where $\alpha_0, \ldots, \alpha_{2^l-1} \in \left[0, 2^l-1\right]$, as $\alpha_i \neq \alpha_j$ when $i \neq j$. Next, we show that the ranges of $\boldsymbol{K}$ and $\boldsymbol{K}''$ are equal in range and the distribution is shifted by a constant amount. As $\zeta_0, \ldots, \zeta_{2^l-1} \in \left[0, 2^l-1\right]$, from the interval, $\boldsymbol{K}_{\alpha_0} \equiv \boldsymbol{K}_\alpha^0 + \zeta_0\lfloor(q-1)/2^l\rfloor \bmod q$ can be expressed as

$$\begin{aligned}\boldsymbol{K}'' &\equiv \left(\boldsymbol{K}_\alpha^0 + \boldsymbol{K}_\alpha^1 - \boldsymbol{K}_\alpha^2 - \boldsymbol{K}_\alpha^3\right) + \left(\zeta_0+\zeta_1-\zeta_2-\zeta_3\right)\lfloor(q-1)/2^l\rfloor \bmod q \\ &= \boldsymbol{K}' + \left(\zeta_0+\zeta_1-\zeta_2-\zeta_3\right)\lfloor(q-1)/2^l\rfloor \bmod q\end{aligned}$$

Thus, the distributions are off by a constant amount and the ranges are equal. From the above and Theorem 3, it can be seen that the case of taking a linear combination with only $\boldsymbol{K}$ biased before the linear combination and the case of taking a linear combination that is unbiased before the linear combination and

biased after the linear combination, the peak values are equal, and the intervals of the distribution are also equal. Therefore, it is not necessary that the set of nonce $\{k_i\}_{i=1}^{M}$ is biased before the linear combination. However, itt is sufficient that $\{k'_j\}_{j=1}^{M'}$ corresponding to the sample after the linear combination is biased.

4.3 Visual explanation of the distribution after linear combination

We show conceptual illustrations of the distribution described in the previous section. First, Figure 5 shows a conceptual illustration of the distribution of conventional nonces that are all biased to 0 at the time of 1 bit leakage. Figure 6 shows a conceptual illustration of the distribution when the top 1 bit are $1 + 1 - 0 - 0$ and linear combinations are taken with uniform nonces. These figures show that the distributions after linear combinations are the same when the 1 bit is leaked, whether the nonces are biased or not.

Next, the distributions for the 2 bits leakage case are shown. Figure 7 shows a conceptual illustration of the distribution of nonces that are all biased to 01 at the time of 2 bits leakage case. Figure 8 shows a conceptual illustration of the distribution when linear combinations are taken with uniform nonces at 2 bits leakage. Here, the top 2 bits are $0 + 1 - 2 - 3$. These figures show that the distributions after linear combinations are the same. The distributions for 3 bits and the other case of 1 bit are shown in Appendix B. Furthermore, we explain that the distributions of Fig 1 and a part of Fig 5 are the same in Appendix B.

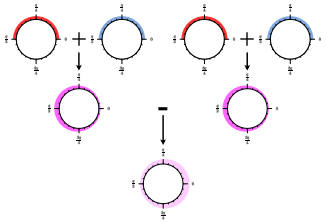

Fig. 5. Distribution by linear combinations with top 1 bit biased towards 0.

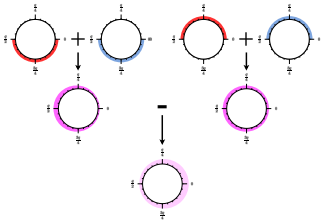

Fig. 6. Distribution by linear combinations with unbiased top 1 bit.

4.4 Algorithm for precomputation resulting in bias

In order to achieve the above, when dividing the list in (a) of Algorithm 5, it is sufficient to make the nonces corresponding to each element of the list $\{\mathcal{L}_i\}_{i=1}^{4}$ biased. In other words, in the case of 2 bits, the HNP samples corresponding to the above 4 random variables and the corresponding nonces should be assigned to the list and given as input. Algorithm 6 shows the precomputation for the general l bits case. In the case of 1 bit, since 2 lists are returned, each list is

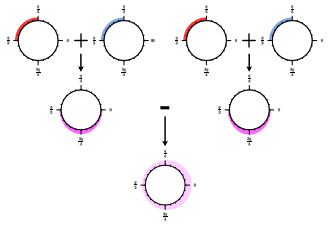

Fig. 7. Distribution by linear combinations with top 2 bits biased towards 01.

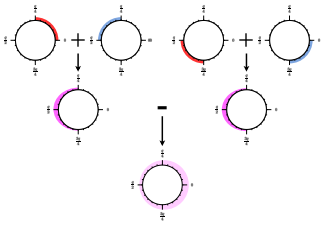

Fig. 8. Distribution by linear combinations with unbiased top 2 bits.

divided into 2 lists to make 4 lists for use in the 4-list sum algorithm. For 2 bits, 4 lists are returned, and for 3 bits or more, 8 or more lists are returned.

Up to this section, we have shown that it is sufficient if k' corresponding to the samples after linear combinations are biased, and the algorithms achieve this. In the following, we will estimate the number of signatures to be collected to recover the secret key when those algorithms are executed.

Algorithm 6 Prepcomputation when uniform nonces are leaked

Input: $\{(\mathrm{MSB}_l(k_i)), h_i, z_i\}_{i=1}^{M}$: HNP saples with leaage to l bits of nonces
Output: List $\{\mathcal{L}_i\}_{i=1}^{2^l}$
1: **for** $i = 1$ to M **do**
2: Add (h_i, z_i) to $\mathcal{L}_{\mathrm{MSB}_l(k_i)+1}$.
3: **end for**
4: **return** $\{\mathcal{L}_i\}_{i=1}^{2^l}$

When 1 bit of nonces are leaked, there are 2 possible values for the top bits are 0 or 1. For the 4 lists $\{\mathcal{L}_i\}_{i=1}^{4}$ in the 4-list sum algorithm, there are two classification methods for the combinations of lists that MSB assigns as elements of lists: $\{0,0,1,1\}$ and $\{1,0,1,0\}$. $\{0,0,1,1\}$ and $\{1,1,0,0\}$, and $\{1,0,1,0\}$ and $\{0,1,0,1\}$ yield equal results, respectively. By classifying the lists as described above, the number of signatures required is $1/2$, since linear combinations can be taken with signature pairs in which the top 1 bit of nonces are both 0 and 1.

When 2 bits are leaked, there are 4 ways in which the top bits are $\{00,01,10,11\}$. For the 4 lists $\{\mathcal{L}_i\}_{i=1}^{4}$ in the 4-list sum algorithm, there are two classification methods for the combinations of lists that MSB assigns as elements of lists: $\{00,01,10,11\}$, $\{00,10,01,11\}$ and $\{00,11,01,10\}$. By classifying and listing, the number of signatures required is $1/4$ since linear combinations can be taken using all signatures where the top 2 bits are $\{00,01,10,11\}$.

When $l \geq 3$ bits are leaked, it is possible to reduce the number of signatures to at least $1/4$ of the previous method by using any 4 lists. In the following, we consider the case where all lists are used. There are two possible ways to obtain biased linear combinations using all the 2^l lists obtained. One is to use

the 2^l-list sum algorithm. Dinur also shows in the generalization process linear combinations with 8-list [8]. Similarly, it is possible for more than 3 bits.

Another way to get biased linear combinations is to perform the first round of 4-list sum algorithm multiple times. Consider the case of 3-bit leakage. $\boldsymbol{K}_0+\boldsymbol{K}_2-\boldsymbol{K}_4-\boldsymbol{K}_6=\boldsymbol{K}_1+\boldsymbol{K}_3-\boldsymbol{K}_5-\boldsymbol{K}_7$, and since the bias ranges are equal, after first round of 4-list sum algorithm for each $\{\boldsymbol{K}_0,\boldsymbol{K}_2,\boldsymbol{K}_4,\boldsymbol{K}_6\}$ and $\{\boldsymbol{K}_1,\boldsymbol{K}_3,\boldsymbol{K}_5,\boldsymbol{K}_7\}$, all the samples obtained in the first round can be run together from the second round. Generalizing at 3 or more l bits leakage, we need to do the following combination of round 1 4-list sum algorithm 2^{l-2} times.

$$\{(\boldsymbol{K}_i,\boldsymbol{K}_{2^{l-2}+i},\boldsymbol{K}_{2^{l-1}+i},\boldsymbol{K}_{2^{l-1}+2^{l-2}+i})\}_{i=0}^{2^{l-2}-1}$$

We will now estimate the specific number of signatures. Using all the uniform signatures obtained at l bits leakage from Equation (7) as $M=2^{a+2+l}$ as linear combinations of biased samples, we obtain $M'\cdot 2^l$ samples in round 1. Because, for $M=2^{a+2}$, M' samples are obtained, and this is repeated 2^l times, so the number of samples output is $M'\cdot 2^l$. Applying the algorithm where each of the 4 lists is different range of lists, the number of signatures in the input is 2^{a+l}, and the number of samples after linear combinations in the output is $M'\cdot 2^{l-2}$. Thus, the required number of samples after linear combinations can be reduced by a factor of $1/2^{l-2}$. Therefore, in the case of $l=2$, an equal number of samples M' can be obtained with a signature count of $1/4$. With respect to the parameter v, M' is minimum for $v=0$ and $\left(2^{a-1}+2\right)2^{-(n-a)}$, and maximum for $v=a+1$ and $\left(2^{3a+1}/3-2^{2a}+7\cdot 2^{a+1}/6\right)2^{-(n-a)}$. If we take the value of M as $1/2$, the number of samples obtained is at most about $1/4$ and at least about $1/16$. If we take η as a positive integer and multiply the value of M by $1/2^\eta$, the number of samples obtained is at least about $1/2^{4\eta}$ times, so the number of samples is about $M'\cdot 2^{l-2}\cdot 2^{-4\eta}=M'\cdot 2^{l-4\eta-2}$. In order to obtain the same number of samples as in the original algorithm, we need only $l-4\eta-2\geq 0$, so we obtain $\eta\geq(l-2)/4$. Thus, the number of signatures required can be at least $1/2^{(l-2)/4+2}=1/2^{(l+6)/4}$ times that of previous studies using only biased samples, or at most $1/2^{(l+2)/2}$ times, so the number of signatures required decreases as l increases.

4.5 New Fourier analysis-based Algorithm for uniform nonces

We will describe an algorithm using the 4-list sum algorithm for uniform nonces, which is shown in Appendix C. First, lists are constructed from the leaked nonces values so that the nonces are biased by using Algorithm 6 then obtain $\mathcal{L}'$. Next, the 4-list sum algorithm is applied to the lists so that the lists become biased after linear combinations.

When biasing while performing the 4-list sum algorithm, there is a slight difference between the 1 bit leakage case and other cases. In the 1 bit leakage case, there are only 2 lists because they are biased toward 0 or 1. Therefore, the 4-list sum algorithm is executed after the list is split into 4 lists. When the leakage is 2 bits or more, there are more than 4 lists. For 3 bits or more, there are more than 8 lists, and each linear combination must be biased to the

same extent. Therefore, we classify them into biased pairs and run the 4-list sum algorithm once on each of them. Since the samples obtained after that are the same biased, the normal 4-list sum algorithm is performed after the 2nd round.

5 Comparison between previous and our proposed algorithms for uniform nonces

In this section, we compare the method proposed in Section 4 with existing methods. The 4-list sum algorithm uses the algorithm proposed in Section 3. In the case of error, we use the error rate obtained in Section 3.8.

Under the following four conditions, experiments were performed on 131-bit ECDSA with and without nonces bias. (1) 1 bit leakage with no error (2) 2 bits leakage with no error (3) 2 bits each with an error rate of about 0.11 leakage (4) 3 bits leakage with no error. In the case of 1 bit leakage, we experimented with different combinations of lists. In the case of 3 bits leakage, we performed experiments on half of the 4 lists and on all 8 lists. A total of 10 experiments were performed. Table 2 shows the parameters used when the 4-list sum algorithm is applied. Table 3 shows the parameters for the algorithm shown in 4.4. The number of signatures in the input is $M = 2^{a_0+3}$, since the 2nd round is performed after the 1st round is run 2 times. Experiments were performed on Ubuntu 20.04 LTS, Intel Xeon Silver 4214R × 2, 24 cores, 48 threads, DDR4 256GB.

Table 2. Parameters of the experiment

Condition	l	ε	a_0	v_0	n_0	a_1	v_1	n_1
(1)	1	0	21	13	56	19	15	47
(2)	2	0	21	13	56	19	12	47
(3)	2	0.11	21	13	56	19	15	47
(4)	3	0	15	15	38	20	11	65

Table 3. Parameters for using all when uniform 3 bits are leaked

Condition	a_0	v_0	n_0	a_1	v_1	n_1
(4)	13	13	35	18	12	68

The attack on the 1 bit leakage with no error was experimented on 4 lists in 2 cases where the top bits are $\{1, 1, 0, 0\}$ and $\{1, 0, 1, 0\}$, respectively. For the attack on 2 bits leakage with no error, we conducted experiments on 4 lists in which the top 2 bits are $\{00, 01, 10, 11\}$. The attack on 2 bits leakage with errors was performed with the same sorting as for 2 bits leakage with no error, with an error rate of about 0.11, which was estimated to be recoverable with the same number of samples after linear combination as for 1 bit leakage with no error. In the experiment with 3 bits leakage with no error for only 4 lists, the lists were selected so that $\{000, 010, 101, 001\}$. In the experiment with 3 bits leakage with no error for only 8 lists, we performed the 4-list sum algorithm 1 time for each list selected so that $\{000, 010, 100, 110\}$ and $\{001, 011, 101, 111\}$, then summarized the outputs and using the 4-list sum algorithm again.

Source Code. Our source code is available in GitHub repository https://github.com/ooshun/bias-from-uniform-nonce.

5.1 Experimental result

Table 4 shows the experimental results when the conventional top bits are biased, and Table 5 shows the results when the top bits are unbiased. Computational time is required for the 4-list sum algorithm. In all conditions, there was no difference in the number of recovered bits, peak values, or average noise excluding the peak values between the unbiased and biased cases. These results confirm that even when the nonces are not biased, they can be biased by linear combinations, and as a result, the secret key can be recovered. Thus, in the case of uniform leakage, it is possible to recover the secret key with a smaller number of signatures, which is shown as M in the tables than the previous method. Using all signatures with 3 bits leakage increased the time.

Table 4. Experimental results with bias

l	ε	M	M'	Time (sec.)	recovered bits	Average noise	Peak value
1	0	2^{24}	$2^{26.90}$	1186	29	$7.9 \cdot 10^{-5}$	$4.6 \cdot 10^{-4}$
2	0	2^{25}	$2^{23.99}$	504	29	$2.1 \cdot 10^{-4}$	$1.2 \cdot 10^{-1}$
	0.11	2^{25}	$2^{26.89}$	1201	29	$7.9 \cdot 10^{-5}$	$1.9 \cdot 10^{-3}$
3	0	2^{20}	$2^{7.93}$	90	29	$5.7 \cdot 10^{-2}$	$4.4 \cdot 10^{-1}$

Table 5. Experimental results with no bias

l	ε	Combinations of lists top l bits	M	M'	Time (sec.)	Recovered bits	Average noise	Peak value
1	0	$\{0,0,1,1\}$	2^{23}	$2^{26.90}$	1210	29	$7.9 \cdot 10^{-5}$	$4.5 \cdot 10^{-4}$
		$\{1,0,1,0\}$	2^{23}	$2^{26.90}$	1223	29	$7.9 \cdot 10^{-5}$	$4.3 \cdot 10^{-4}$
2	0	$\{00,01,10,11\}$	2^{23}	$2^{23.98}$	530	29	$2.1 \cdot 10^{-4}$	$1.2 \cdot 10^{-1}$
	0.11	$\{00,01,10,11\}$	2^{23}	$2^{26.89}$	1190	29	$7.9 \cdot 10^{-5}$	$1.9 \cdot 10^{-3}$
3	0	$\{000,010,101,001\}$	2^{18}	$2^{7.80}$	87	29	$5.9 \cdot 10^{-2}$	$4.4 \cdot 10^{-1}$
		$\{000,010,100,110,$ $001,011,101,111\}$	2^{16}	$2^{7.77}$	829	29	$6.0 \cdot 10^{-2}$	$4.6 \cdot 10^{-1}$

6 Conclusion

First, we modified 4-list sum algorithm in Section 3 . Issues related to carry and distribution were pointed out and we derived the exact distribution and made improvements incorporating that distribution. Second, we proposed an attack method for the unbiased nonces in Section 4. We showed that when the nonces are uniformly distributed, the number of signatures required in the case of a uniform distribution can be reduced to 1/2 in the case of a 1 bit leakage and 1/4 in the case of 2 bits and more leakage. In addition, we showed that the number of signatures can be reduced to $1/2^{(l+6)/4}$ times for l over 3 bits. The conventional collision search plays the role of reducing the number of secret key searches. In this study, we show that it can also reduce the number of signatures used by playing the role of biasing the secret key at the same time. Finally, we performed comparison experiments between uniform and biased distributions in Section 5 and confirmed that the number of required signatures could decrease.

Acknowledgements This work was supported by JST CREST Grant Number JPMJCR2113 and JSPS KAKENHI Grant Number 23K21667.

References

1. Albrecht, M.R., Heninger, N.: On bounded distance decoding with predicate: Breaking the "lattice barrier" for the hidden number problem. In: EUROCRYPT 2021. LNCS, vol. 12696, pp. 528–558. Springer (2021)
2. Aranha, D.F., Fouque, P., Gérard, B., Kammerer, J., Tibouchi, M., Zapalowicz, J.: GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In: ASIACRYPT 2014. LNCS, vol. 8873, pp. 262–281. Springer (2014)
3. Aranha, D.F., Novaes, F.R., Takahashi, A., Tibouchi, M., Yarom, Y.: Ladderleak: Breaking ECDSA with less than one bit of nonce leakage. In: CCS '20: 2020 ACM SIGSAC. pp. 225–242. ACM (2020)
4. Bleichenbacher, D.: On the generation of one-time keys in dl signature schemes. In: Presentation at IEEE P1363 working group meeting. p. 81 (2000)
5. Boneh, D., Joux, A., Nguyen, P.Q.: Why textbook elgamal and RSA encryption are insecure. In: ASIACRYPT 2000. LNCS, vol. 1976, pp. 30–43. Springer (2000)
6. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In: CRYPTO '96. LNCS, vol. 1109, pp. 129–142. Springer (1996)
7. De Mulder, E., Hutter, M., Marson, M., Pearson, P.: Using bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ecdsa: Extended version. Journal of Cryptographic Engineering **4**(1), 33–45 (2014)
8. Dinur, I.: An algorithmic framework for the generalized birthday problem. Des. Codes Cryptogr. **87**(8), 1897–1926 (2019)
9. Gao, Y., Wang, J., Hu, H., He, B.: Attacking ecdsa with nonce leakage by lattice sieving: Bridging the gap with fourier analysis-based attacks. Cryptology ePrint Archive, Paper 2024/296 (2024)
10. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: EUROCRYPT 2010. LNCS, vol. 6110, pp. 235–256. Springer (2010)
11. Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: An update. In: CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer (2013)
12. Osaki, S., Kunihiro, N.: Extended attacks on ECDSA with noisy multiple bit nonce leakages. In: ICISC 2023. LNCS, vol. 14561, pp. 163–184. Springer (2023)
13. Sun, C., Espitau, T., Tibouchi, M., Abe, M.: Guessing bits: Improved lattice attacks on (EC)DSA with nonce leakage. IACR TCHES **2022**(1), 391–413 (2022)
14. Takahashi, A., Tibouchi, M., Abe, M.: New bleichenbacher records: Fault attacks on qdsa signatures. IACR TCHES **2018**(3), 331–371 (2018)
15. Wagner, D.A.: A generalized birthday problem. In: CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer (2002)

A The number of collisions

Consider the number of collisions assumed in [3] for (b) of the algorithm Algorithm 3, the algorithm before modification, and the number of collisions in (b) of Algorithm 5, the algorithm after modification. Figure 9 and Figure 10 show

the collision counts for the case where $a = 5$. It can be seen that the number of collisions is reduced in the modified algorithm. This also indicates that the range of c should be taken as in Algorithm 5.

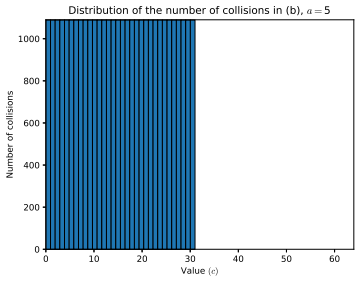

Fig. 9. Number of collisions in (b) assumed in [3].

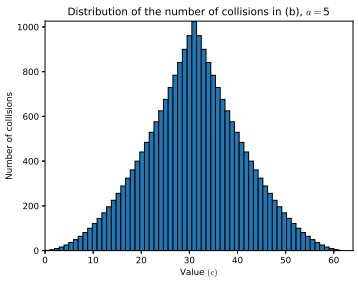

Fig. 10. Number of collisions in (b) of Algorithm 5.

B Distributions after linear combinations

We showed the distribution when nonces are unbiased and top 1 bit of lists are $\{1, 1, 0, 0\}$ in Fig 6. Figure 11 is distribution when nonces are unibiased and top 1 bit of lists are $\{0, 1, 0, 1\}$.

We described the distribution for unibiased 3 bits leakage. Figure 12 is distribution by linear combinations of the first round when all elements are used when the top 3 bits are unbiased.

In Section 4.3, we said the distributions of Fig 1 and a part of Fig 5 are the same. Figures 13 and 14 show how this is illustrated. It means that the sum of uniform distributions is biased. We can say that each distribution in Fig 13 is attached to the unit circle in Fig 14.

C Collision search for unbiased nonces

Algorithm 7 shows an algorithm for linear combinations using the 4-list sum algorithm when the nonces are uniform and the top bits are leaked. First, lists are created from the leaked nonces values so that the nonces are biased. Next, the 4-list sum algorithm is applied to the lists so that the lists become biased after linear combinations. The resulting list can then be used to compute the secret key by performing bias computation in Algorithm 2. Figure 12 shows lines 7 through 9 of Algorithm 7.

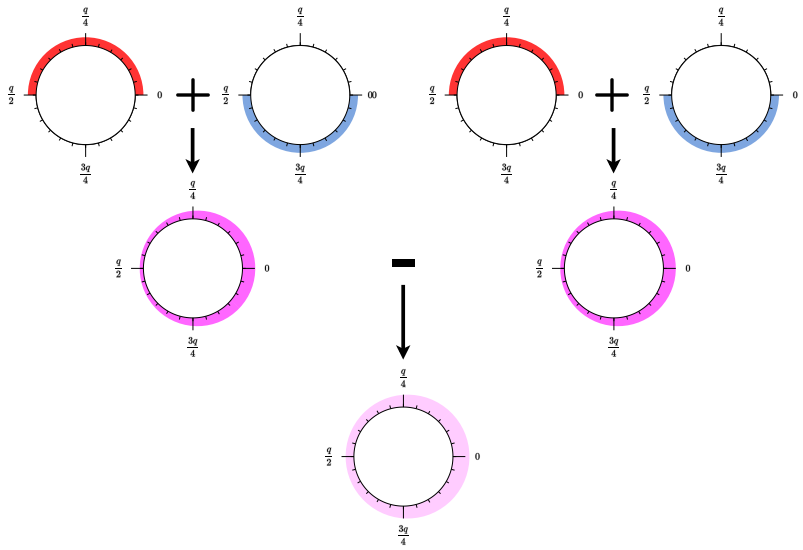

Fig. 11. Distribution after linear combinations with unbiased top 1 bit as $\{0, 1, 0, 1\}$.

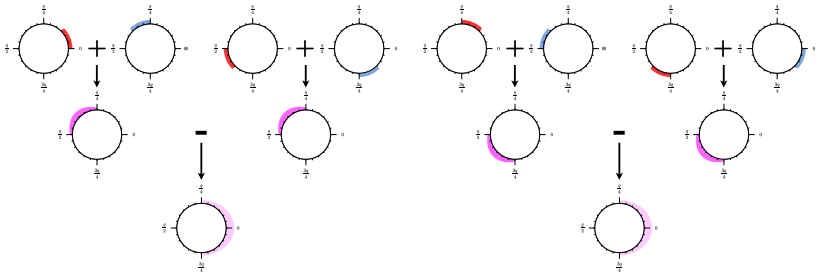

Fig. 12. Distribution after linear combinations of the first round when all elements are used when the top 3 bits are unbiased.

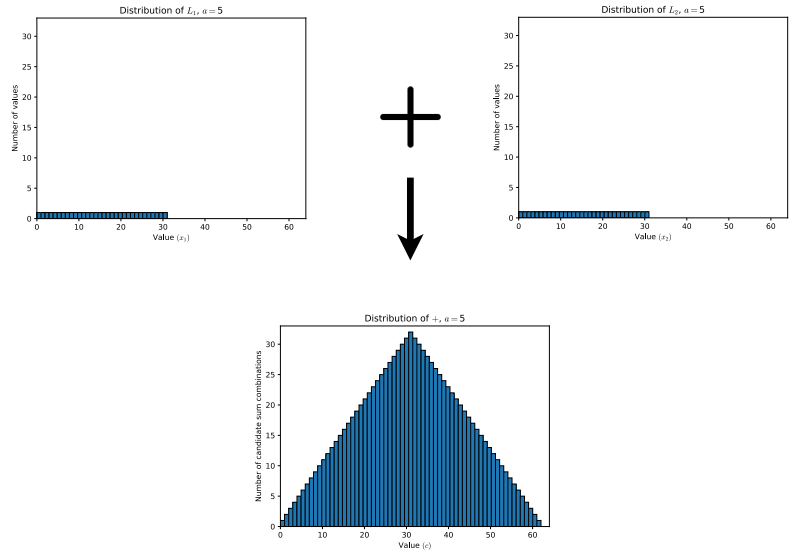

Fig. 13. Distribution after linear combinations of histogram.

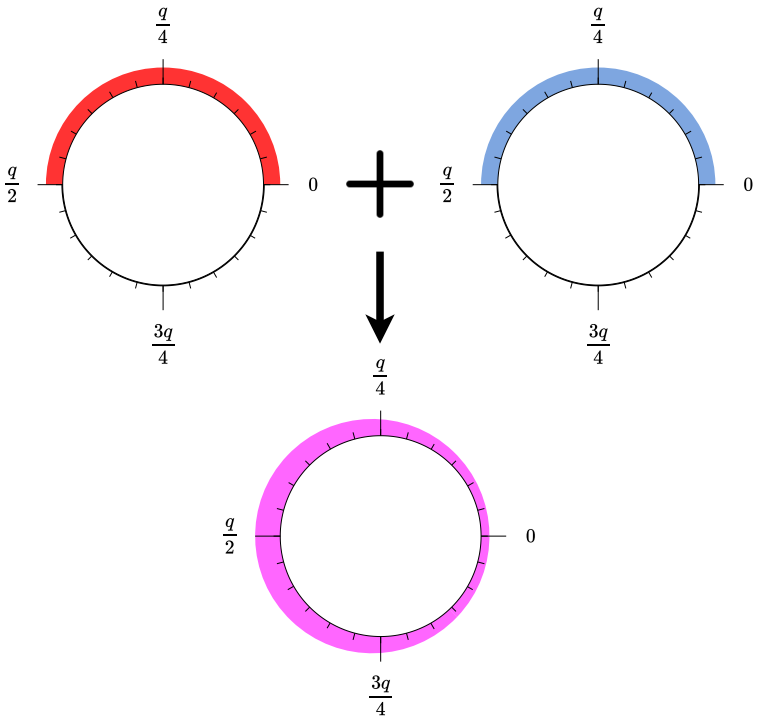

Fig. 14. Distribution after linear combinations on the unit circle.

Algorithm 7 Collision search using 4-list sum algorithm for uniform nonces

Input: $\{(\mathrm{MSB}_{\lambda,l}(k_i), h_i, z_i)\}_{i=1}^{M}$: HNP sample on $\mathbb{Z}_q$ corresponding to uniform nonces with top l bits leaked, M': number of linear combinations to be found, L_{FFT}: FFT table size.

Output: $\mathcal{L}'$

Precomputation
Execute Algorithm 6 and obtain $\{\mathcal{L}_i\}_{i=1}^{2^l}$.
Collision Search
if $l = 1$ **then**
  Split half of the elements of $\mathcal{L}_1$ and $\mathcal{L}_2$ into $\mathcal{L}_3$ and $\mathcal{L}_4$ respectively and perform 4-list sum algorithm
else
  for $i = 0$ to $2^{l-2} - 1$ **do**
    Execute round 1 of the 4-list sum algorithm for the quadruplets $\left\{\boldsymbol{K}_i, \boldsymbol{K}_{2^{l-2}+i}, \boldsymbol{K}_{2^{l-1}+i}, \boldsymbol{K}_{2^{l-1}+2^{l-2}+i}\right\}$ and add the result to $\mathcal{L}'$.
  end for
  Run $(r-1)$ rounds of 4-list sum algorithm for $\mathcal{L}'$.
  return $\mathcal{L}'$
end if