A New Sieving Approach for Solving the HNP with One Bit of Nonce by Using Built-in Modulo Arithmetic

Yao Sun and Shuai Chang

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.

Abstract. The Hidden Number Problem (HNP) has been extensively used in the side-channel attacks against (EC)DSA and Diffie-Hellman. The lattice approach is a primary method of solving the HNP. In EUROCRYPT 2021, Albrecht and Heninger constructed a new lattice to solve the HNP, which converts the HNP to the SVP. After that, their approach became the state-of-the-art lattice method of solving the HNP. But Albrecht and Heninger's approach has a high failure rate for solving the HNP with one bit of nonce (1-bit HNP) because there are enormous vectors related to the modulus q shorter than the target vector in Albrecht and Heninger's Lattice.
To decrease the number of vectors that are shorter than the target vector and avoid the duplicated reduction, we introduce the modulo-q lattice, a residue class ring of the general lattice modulo q, where q is the modulus of the HNP. We present a new sieving algorithm to search for the shortest vectors in the modulo-q lattice. Our algorithm uses built-in modulo q arithmetic and many optimization techniques. As a result, we can solve a general 1-bit HNP ($q = 2^{120}$) within 5 days and solve a general 1-bit HNP ($q = 2^{128}$) within 17 days.

Keywords: Hidden Number Problem (HNP) · lattice · sieving algorithm · modulo arithmetic

1 Introduction

The Hidden Number Problem The Hidden Number Problem (HNP) was extensively used in the side-channel attacks against (EC)DSA and Diffie-Hellman. In [15], the adversaries are supposed to know some significant bits of random multiples of a secret integer modulo some known integer. This secret integer can be constrained by a set of integer-linear modulo equations.

Since the creative works of Bleichenbacher [13] and Howgrave-Graham and Smart [27], side-channel information about (EC)DSA nonces have been extensively used by solving the Hidden Number Problem, including [42, 34, 9, 46, 50, 47, 40, 51, 28]. The former approach deploys a combinatorial algorithm that can be cast as a variant of the BKW algorithm, [14, 4, 30, 23]. The latest improvement

is [10], which recovers a key from less than one bit of the nonce by Bleichenbacher's algorithm. Recently, the authors in [35] found the first practical attack scenario that was able to use Boneh and Venkatesan's [15] original application of the HNP to prime-field Diffie-Hellman key exchange.

The other efficient way of solving for the secret integer is based on lattice reduction. The Hidden Number Problem can be reformulated as a lattice and resolved by solving the Closest Vector Problem, also known as Bounded Distance Decoding [42]. This conversion takes advantage of the fact that the unknown parts of the random multiples of a secret integer are relatively smaller than the modulus number. Thus, it suffices to find a relatively small solution to a set of integer-linear modulo equations. In CVP, one must find a uniquely closest vector in the lattice to some given point.

The work in [7] provided a more effective way of solving the HNP by breaking the "lattice barrier" using a predicate. Albrecht and Heninger considered the expected squared norms of the target vectors instead of the upper bounds, and they also converted the HNP to the SVP by constructing a new lattice. Thus, with the help of efficient lattice reduction tools, e.g., [6, 19], they solve many HNPs which were considered infeasible previously. We will show more details about Albrecht and Heninger's method in Sec. 2.3.

Lattice algorithms Lattice reduction algorithms [33, 48, 49, 21, 39], are still the primary tools for solving the HNP practically. The efficiency of lattice reduction algorithms determines the magnitude of the HNP that can be solved. Lattice reduction algorithms also have numerous other applications in cryptanalysis, including factoring RSA keys with partial information about the secret key via Coppersmith's method [16, 41], the (side-channel) analysis of lattice-based schemes [36, 8, 26, 5, 17].

Given a lattice, two kinds of lattice algorithms are used to resolve the SVP.

Enumeration Enumeration approaches, including [44, 29, 20, 49, 24, 22, 38, 3], search for vectors whose lengths are bounded by some given value R via lattice-vector enumeration. Specifically, enumeration algorithms are based on the fact that any vector in the lattice can be rewritten with respect to the Gram-Schmidt basis. Lattice-vector enumeration uses a depth-first tree search through a tree to find out all available vectors whose lengths are not longer than R, so the "pruning" strategy is the primary way to speed up enumeration algorithms.

Sieving Sieving approaches, including [31, 1, 43, 37, 12, 32, 11, 25, 18, 6, 19], take a list of vectors in the lattice as input and search for integer combinations of these vectors that are short. Sieves that consider k vectors at a time are called k-sieves. Specifically, given two vectors $\mathbf{u}$ and $\mathbf{v}$ in a lattice $\mathcal{L}$ with $\mathbf{u} \neq \pm\mathbf{v}$, 2-sieves consider the integer combinations of the form $\mathbf{u} \pm \mathbf{v}$. As the size of the list in the sieving algorithm is often very large (e.g., $> 2^{20}$), it is too expensive to consider all pairs of points in the list. Researchers tend to select some of the vectors in the list and put them into buckets. Next, only pairs of vectors in the

same bucket are considered in practical computations, which saves much more time than checking all pairs in the list. Thus, the techniques of bucketing play an important role in the sieving approaches.

1-bit HNP We call the HNP with only one bit of nonce the 1-bit HNP. Some theoretical analyses on 1-bit HNP were done in [2]. In [10], Aranha et al. recovered a key from less than one bit of the nonce by Bleichenbacher's algorithm. This method relies on a search for heavy Fourier coefficients. "However, those heavy Fourier coefficients only reveal the secret key in an HNP instance where the most significant bits of nonces are constant (say identically zero)" [50]. Besides, compared with the lattice approach, Aranha et al.'s method requires a large number of nonces, e.g., at least 2^{20}, while the lattice approach only needs a few nonces.

At present, there are no effective lattice approaches that can solve the 1-bit HNP. Albrecht and Heninger's method is the state-of-the-art lattice method of solving the HNP, but their method fails to solve the 1-bit HNP. We believe the main reason is that *there are a vast number of vectors shorter than the target vector in the lattice they constructed*. Here the target vector is a vector constructed by Albrecht and Heninger's method, and the HNP will be solved once the target vector is found in the lattice. Given m nonces of the 1-bit HNP with the modulus $q = 2^{128}$, we can construct the lattice by Albrecht and Heninger's method and calculate the value obtained by the Gaussian heuristic and the expected squared lengths of target vectors. The data are shown in Fig. 1. The target vector is supposed to be the shortest if m is 170, where the value obtained by the Gaussian heuristic becomes larger than the expected squared length of the target vector. Unfortunately, the target vector will not be the shortest in Albrecht and Heninger's lattice, no matter how many nonces are used. This is because there is a special kind of vectors, say $\mathbf{q}_i = (0, \ldots, q, \ldots, 0)$, existing in Albrecht and Heninger's lattice, where q is the modulus of the HNP. For convenience, we call the vectors $\mathbf{q}_i$'s and $\sum t_i \cdot \mathbf{q}_i$ as q-vectors, where t_i's are integers. If m is 170, there should be $\binom{169}{3} \cdot 2^3 + \binom{169}{2} \cdot 2^2 + \binom{169}{1} \cdot 2 = 6379074$ q-vectors shorter than the expected squared length of the target vector, which makes the target vector very difficult to be found.

Main idea and contribution Our key observation is that *if there are only a few vectors, e.g., no more than* 10^4*, that are shorter than the target vector, the target vector will be found by lattice algorithms with a high success rate*. Thus, to solve 1-bit HNP by lattice approach, we need to decrease the number of vectors that are shorter than the target vector. Hence, the q-vectors should be removed first of all.

The introduction of $\mathbf{q}_i$ to the lattice is to simulate the "modulo q" operation in the HNP, such that general lattice tools, e.g., [6, 19], developed for general lattices can be used to search for shortest vectors over integers. We realized that if we could develop a sieving algorithm with built-in modulo q operation, then

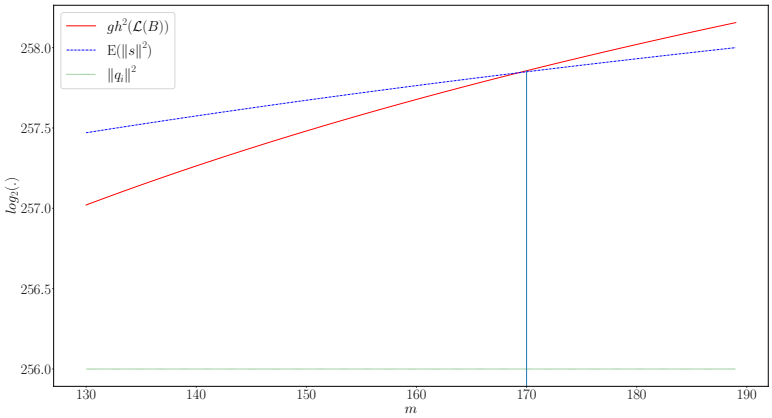

Fig. 1. Albrecht and Heninger's method of solving 1-bit HNP ($q = 2^{128}$).

$\mathbf{q}_i$'s are no longer needed. Hence, the number of vectors that are shorter than the target vector will decrease significantly.

Therefore, to use the built-in modulo q operation, we introduce the modulo-q lattice, which is a residue class ring of the general lattice modulo the model generated by $\mathbf{q}_i$'s. We designed a sieving algorithm to search for short vectors in the modulo-q lattice, including an efficient bucketing method in the modulo-q lattice. All appeared integers in this algorithm are treated by the modulo q operation automatically. This algorithm has been implemented by C++/CUDA. As a result, we can solve a general 1-bit HNP ($q = 2^{120}$) within 5 days and solve a general 1-bit HNP ($q = 2^{128}$) within 17 days.

Relations to Albrecht and Heninger's work [7] and Aranha et al.'s work [10] Our work is improved based on Albrecht and Heninger's work. We first construct Albrecht and Heninger's lattice in our approach and then convert it from the general lattice to the modulo-q lattice. Albrecht and Heninger's target vector also lies in the corresponding modulo-q lattice. We develop a new sieving algorithm to search for the target vector in the modulo-q lattice. Compared with Albrecht and Heninger's method, our approach has two advantages for solving the 1-bit HNP.

- q-vectors become zero vectors in the modulo-q lattice, so the number of vectors that are shorter than the target vector decreases, which enables us to find the target vector easier.
- The vectors in the modulo-q lattice are all residue class of integer vectors, such that many distinct vectors in the general lattice become the same in the modulo-q lattice. That is, the sieving algorithm in modulo-q lattice only

needs to perform reduction to a vector $\mathbf{v}$ and can avoid all reduction to the vectors $\mathbf{v} + \sum t_i \cdot \mathbf{q}_i$, which saves much time to find the target vector.

Compared with Aranha et al.'s work, which is based on Bleichenbacher's algorithm, our lattice approach only needs a few nonces. Specifically, Aranha et al.'s method requires more than 2^{20} nonces to solve the 1-bit HNP with $q > 2^{100}$, while less than 2^{10} nonces are enough for our approach.

Open Source Code The codes of the algorithms reported in this work will be released at GitHub publicly once this paper is published somewhere.

Outline We introduce some preliminaries on the lattice, the HNP, and Albrecht and Heninger's work to solve the HNP in Sec. 2. We give the definition of the module-q lattice and the sieving algorithm in Sec. 3. Experiments come in Sec. 4, and we conclude this paper in Sec. 5.

2 Preliminaries

2.1 Lattice

Let $\mathcal{Z}$ be the ring of integers, and $q \in \mathcal{Z}$ be a modulus. Then $\mathcal{Z}_q = \mathcal{Z}/(q\mathcal{Z})$ is the residue class ring of $\mathcal{Z}$ modulo q. In this paper, we also regard elements in $\mathcal{Z}_q$ as integers if no confusions occur, so we have $\mathcal{Z}_q \subset \mathcal{Z}$.

In this work, all vectors are denoted by bold lowercase letters, and matrices are denoted by bold capital letters. A lattice $\mathcal{L}$ is a discrete subgroup of $\mathcal{R}^d$, where $\mathcal{R}$ is the rational number field, and d is the dimension of the vectors. When the rows $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$ of a matrix $\mathbf{B}$ are linearly independent over the field $\mathcal{R}$, we regard $\mathbf{B}$ as the basis of the lattice $\mathcal{L}(\mathbf{B}) = \{\sum v_i \cdot \mathbf{b}_i \mid v_i \in \mathcal{Z}\}$. That is, row representations for matrices are used in this paper.

The Gram-Schmidt orthogonalization of $\mathbf{B}$ is $\mathbf{B}^* = (\mathbf{b}_0^*, \ldots, \mathbf{b}_{d-1}^*)$, where the Gram-Schmidt vector $\mathbf{b}_i^*$ is $\pi_i(\mathbf{b}_i)$ and $\pi_i : \mathcal{R}^d \to \mathrm{span}(\mathbf{b}_0, \ldots, \mathbf{b}_{i-1})$ for $i = 0, \ldots, d-1$. Then $\mathbf{b}_0^* = \mathbf{b}_0$ and $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=0}^{i-1} \mu_{i,j} \cdot \mathbf{b}_j^*$ for $i = 1, \ldots, d-1$ and $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$. The **length** of a vector $\mathbf{v}$ is the Euclidean norm of $\mathbf{v}$, denoted by $\|\mathbf{v}\|$. The **volume** of a lattice $\mathcal{L}(\mathbf{B})$ is $\mathrm{Vol}(\mathcal{L}(\mathbf{B})) = \prod_i \|\mathbf{b}_i^*\|$, that is an invariant of the lattice. The first minimum of a lattice $\mathcal{L}$ is the length of the shortest non-zero vector, denoted by $\lambda_1(\mathcal{L})$. For simplification, we use $\mathrm{Vol}(\mathbf{B}) = \mathrm{Vol}(\mathcal{L}(\mathbf{B}))$ and $\lambda_1(\mathbf{B}) = \lambda_1(\mathcal{L}(\mathbf{B}))$ in this work.

The Gaussian heuristic predicts that the number $|\mathcal{L} \cap \mathcal{B}|$ of lattice points inside a measurable body $\mathcal{B} \subset \mathcal{R}^n$ is approximately equal to $\mathrm{Vol}(\mathcal{B})/\mathrm{Vol}(\mathcal{L})$. Applied to Euclidean d-balls, it leads to the following prediction of the length of the shortest non-zero vector in a lattice.

Definition 1. *We denote by* $\mathrm{gh}(\mathcal{L})$ *the expected first minimum of a lattice* $\mathcal{L}$ *according to the Gaussian heuristic. For a full rank lattice* $\mathcal{L} \subset \mathcal{R}^d$*, it is given*

by:

$$\mathrm{gh}(\mathcal{L}) \approx \sqrt{\frac{d}{2\pi e}} \cdot \mathrm{Vol}(\mathcal{L})^{1/d}.$$

One of the most critical problems on lattices is to find the shortest vector in a lattice.

Definition 2 (Shortest Vector Problem (SVP)). *Given a lattice basis* $\mathbf{B}$*, find the shortest non-zero vector in* $\mathcal{L}(\mathbf{B})$*.*

In this work, we frequently meet a special kind of vectors in $\mathcal{Z}^d$. Let $q \in \mathcal{Z}$ be a modulus, and we denote $\mathbf{q}_i = (0, \ldots, q, \ldots, 0) \in \mathcal{Z}^d$ which is a one-hot vector with the ith coefficient being q. We call the vectors $\mathbf{q}_i$'s and $\sum_{i=0}^{d-1} t_i \cdot \mathbf{q}_i$ as **q-vectors** if no confusions occur, where $t_i \in \mathcal{Z}$ for $0 \leq i < d$.

2.2 The Hidden Number Problem

To show our approach, we first give a mathematical definition of the Hidden Number Problem (HNP), which was proposed in [15]. The Hidden Number Problem can be regarded as 1-dimensional LWE [45]. The notations in the following definition will be used throughout this paper.

Definition 3 (the Hidden Number Problem (HNP)). *Let* q *be a modulus with* $2^{n-1} < q \leq 2^n$*, and* x *be a secret integer in* $\mathcal{Z}_q = \mathcal{Z}/(q\mathcal{Z}) = \{0, \ldots, q-1\}$*. Assume there are* m *pairs of* $(\alpha_i, \beta_i) \in \mathcal{Z}_q^2$*, such that*

$$\beta_i + k_i = \alpha_i \cdot x \bmod q, \tag{1}$$

where $0 \leq k_i < 2^l$ *for* $0 \leq i < m$*, and* l *is an integer smaller than* n*. In case that* (α_i, β_i)*'s are known and* k_i*'s,* x *are unknown, find an integer* $x' \in \mathcal{Z}_q$ *to make Eq. (1) hold for all* $0 \leq i < m$*.*

Generally, to ensure the found x' is just the secret integer x, the constraint $m > n$ is required in the HNP. For convenience, we call the pair $(\alpha_i, \beta_i) \in \mathcal{Z}_q^2$ in the above definition as *nonce data* in the rest of this paper.

By saying **1-bit HNP** in this work, we mean the Hidden Number Problem with $l = n - 1$ in Def. 3. That is, in 1-bit HNP, only one (most significant) bit of nonce is known.

2.3 Solving the HNP by Lattice Approach

One important improvement on solving the Hidden Number Problem by lattice is given in [7]. In that work, Albrecht and Heninger constructed a lattice in the following way to convert the HNP in Def. 3 to the SVP in the lattice. Let $\tilde{\beta}_i = \beta_i + 2^{l-1}$ and $k'_i = k_i - 2^{l-1}$, then Eq. (1) becomes

$$\beta_i + k_i = \tilde{\beta}_i + k'_i = \alpha_i \cdot x \bmod q. \tag{2}$$

Note that $0 \leq k_i < 2^l$, thus $-2^{l-1} \leq k'_i < 2^{l-1}$ holds.

Assume α_0 and q are coprime[1], then there exists a number α_0^{-1}, such that $\alpha_0^{-1} \cdot \alpha_0 = 1 \bmod q$. Thus, we can get $\alpha_0^{-1} \cdot (\tilde{\beta}_0 + k'_0) = x \bmod q$, which can be used to eliminate x in the other equations. Finally, the following equations are obtained:

$$\beta'_i + k'_i = \alpha'_i \cdot k'_0 \bmod q, \tag{3}$$

where $\beta'_i = \tilde{\beta}_i - \alpha_i \cdot \alpha_0^{-1} \cdot \tilde{\beta}_0$ and $\alpha'_i = \alpha_i \cdot \alpha_0^{-1}$ for $0 < i < m$.

Denote $w = 2^{l-1}$, Albrecht and Heninger constructed an $(m+1)$-dimensional lattice $\mathcal{L}(\mathbf{B})$, which is generated by

$$\mathbf{B} = \begin{bmatrix} q & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & q & 0 & \cdots & 0 & 0 & 0 \\ & \vdots & & \ddots & & & \\ 0 & 0 & 0 & \cdots & q & 0 & 0 \\ \alpha'_1 & \alpha'_2 & \alpha'_3 & \cdots & \alpha'_{m-1} & 1 & 0 \\ \beta'_1 & \beta'_2 & \beta'_3 & \cdots & \beta'_{m-1} & 0 & w \end{bmatrix}. \tag{4}$$

The target vector is $\mathbf{s} = (-k'_1, -k'_2, \ldots, -k'_{m-1}, -k'_0, w)$, where $-w \leq k'_i < w$ for $0 \leq i < m$. So the length of $\mathbf{s}$ is bounded by $(m+1) \cdot w^2$.

Clearly, the target vector $\mathbf{s}$ is in $\mathcal{L}(\mathbf{B})$, because of Eq. (3). Conversely, if a vector $\mathbf{v} = (v_0, \ldots, v_{m-1}, w) \in \mathcal{L}(\mathbf{B})$ is found such that $-w < v_i \leq w$ for $0 \leq i < m$, then let $k'_0 = -v_{m-1}$, we can recover an integer x' by the equation $\alpha_0^{-1} \cdot (\tilde{\beta}_0 + k'_0) = x' \bmod q$, where α_0 and $\tilde{\beta}_0$ are already known. It is not hard to verify that x' could make Eq. (1) hold for $0 \leq i < m$, since $-w < v_i \leq w$. If $m > \log_2(q)$, we should have $\mathbf{s} = \mathbf{v}$.

If the length of the target vector $\mathbf{s}$ is not large, it can be found by lattice algorithms with a high success rate [7]. The vector $\mathbf{s}$ is supposed to be the shortest in $\mathcal{L}(\mathbf{B})$ when the dimension of $\mathcal{L}(\mathbf{B})$ is large enough. According to the Gaussian heuristic, the shortest vector in the lattice should have the length of

$$\mathrm{gh}(\mathcal{L}(\mathbf{B})) \approx \sqrt{\frac{m+1}{2\pi e}} \cdot \mathrm{Vol}(\mathcal{L}(\mathbf{B}))^{1/(m+1)} = \sqrt{\frac{m+1}{2\pi e}} \cdot (n^{m-1} \cdot w)^{1/(m+1)}. \tag{5}$$

Instead of using the upper bound $(m+1) \cdot w^2$ of $\mathbf{s}$, Albrecht and Heninger considered the *expected squared length* of $\mathbf{s}$, i.e.

$$\mathbf{E}\left[\|\mathbf{s}\|^2\right] = \mathbf{E}\left[(\sum_{i=0}^{m-1} k'^2_i) + w^2\right] = m \cdot w^2/3 + m/6 + w^2. \tag{6}$$

Albrecht and Heninger observed that the squared lengths of target vectors are close to the expected squared length, and if $\|\mathbf{s}\| < \mathrm{gh}(\mathcal{L}(\mathbf{B}))$, the vector $\mathbf{s}$ is the shortest vector in $\mathcal{L}(\mathbf{B})$ with high probability. Besides, their method in [7] also works for $\|\mathbf{s}\| \geq \mathrm{gh}(\mathcal{L}(\mathbf{B}))$, but we found in our experiments that the success rate

[1] This condition is not hard to meet, since there are m α_i's.

of Albrecht and Heninger's method drops quickly when $\|\mathbf{s}\|$ becomes far away from $\mathrm{gh}(\mathcal{L}(\mathbf{B}))$.

Assumption 1([7, 18]) *When a 2-sieve algorithm terminates, it outputs a database L containing all vectors with norm $\leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$.*

Theorem 1 ([7]). *Let $\mathcal{L} \subset \mathcal{R}^d$ be a lattice containing a vector $\mathbf{v}$ such that $\|\mathbf{v}\| \leq R = \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$. Under Assumption 1, Algorithm 1 is expected to find the minimal $\mathbf{v}$ satisfying $f(\mathbf{v}) = 1$ in $2^{0.292d+o(d)}$ steps and $(4/3)^{d/2+o(d)}$ calls to $f(\cdot)$, where $f(\cdot)$ is a function to check whether $\mathbf{v}$ is the target vector.*

Since the target vector is $\mathbf{s} = (-k'_1, -k'_2, \ldots, -k'_{m-1}, -k'_0, w)$, where $-w \leq k'_i < w$ for $0 \leq i < m$. Thus, in the following algorithm, $f(\mathbf{v}) = 1$ only if $\mathbf{v} = (v_0, \ldots, v_m, w)$ and $-w < v_i \leq w$ for $0 \leq i < m$.

Algorithm 1: Sieving with Predicate [7]

Input: Lattice basis $\mathbf{B} = \{\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}\}$; Predicate $f(\cdot)$.
Output: $\mathbf{v}$ such that $\|\mathbf{v}\| \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}(\mathbf{B}))$ and $f(\mathbf{v}) = 1$ or $\perp$.

```
begin
    r ⟵ ⊥
    Run sieving algorithm on b_0, ..., b_{d-1} and denote output list as L
    for v in L do
        if f(v) = 1 and (r = ⊥ or ||v|| < ||r||) then
            r ⟵ v
    return r
```

3 Modulo-q Lattice

Given the number m which is the number of pairs (α_i, β_i)'s, the expected squared length of the target vector $\mathbf{s}$ in Albrecht and Heninger's lattice is $\mathbf{E}[\|\mathbf{s}\|^2] = m \cdot w^2/3 + m/6 + w^2$, where $w = 2^{l-1}$. Particularly, for 1-bit HNP, we have $w = q/4$. Note that

- the squared length of $\pm\mathbf{q}_i$ is $q^2 = 16 \cdot w^2$;
- the squared length of $\pm\mathbf{q}_i \pm \mathbf{q}_j$ where $i \neq j$ is $2 \cdot q^2 = 32 \cdot w^2$;
- the squared length of $\pm\mathbf{q}_i \pm \mathbf{q}_j \pm \mathbf{q}_k$ where $i \neq j \neq k$ is $3 \cdot q^2 = 48 \cdot w^2$;
- ....

That is, if $144 \leq m < 192$, then there are $\binom{m-1}{1} \cdot 2 + \binom{m-1}{2} \cdot 2^2 + \binom{m-1}{3} \cdot 2^3$ q-vectors shorter than the target vector $\mathbf{s}$ whose squared length is $\mathbf{E}[\|\mathbf{s}\|^2]$, which will make the target vector very difficult to be found successfully.

To remove the q-vectors from the lattice, we introduce the modulo-q lattice in Sec. 3.1. We show how to solve the HNP by the modulo-q lattice in 3.2. We present a sieving algorithm to search for the target vectors in Sec. 3.3. We discuss Albrecht and Heninger's "breaking the 'lattice barrier' " technique in Sec. 3.4.

3.1 The Definition of the Modulo-q Lattice

In fact, the modulo-q lattice is a residue class ring of the general lattice modulo q, where $q \in \mathcal{Z}$ is a modulus. For simplification, we use the operator $[\cdot]_q$ to denote the modulo q operator in this paper.

Definition 4. *Let $q \in \mathcal{Z}$ be a modulus and a be an integer in $\mathcal{Z}$. We denote*

$$[a]_q = a \bmod q.$$

Thus, we have $[a]_q \in \mathcal{Z}_q = \mathcal{Z}/(q\mathcal{Z})$. Similarly, let $\mathbf{v} = (v_0, v_1, \ldots, v_{d-1}) \in \mathcal{Z}^d$. Then $[\mathbf{v}]_q$ is defined as

$$[\mathbf{v}]_q = ([v_0]_q, [v_1]_q, \ldots, [v_{d-1}]_q) \in \mathcal{Z}_q^d.$$

If no confusions occur, we always regard $[a]_q$ as an integer in $\{0, \ldots, q-1\}$ in this paper, i.e. $\mathcal{Z}_q \subset \mathcal{Z}$.

Let $q \in \mathcal{Z}$ be a modulus and $a, b, t \in \mathcal{Z}$. It is easy to verify that

$$[a]_q + [b]_q = [a+b]_q, [a]_q - [b]_q = [a-b]_q, t \cdot [a]_q = [t \cdot a]_q.$$

Similarly, let $\mathbf{a}, \mathbf{b} \in \mathcal{Z}^d$ and $t \in Z$, we have

$$[\mathbf{a}]_q + t \cdot [\mathbf{b}]_q = [\mathbf{a} + t \cdot \mathbf{b}]_q.$$

The major difference between the general lattice and the modulo-q lattice is the definition of the lengths of vectors. The motivation of the following definition is to ensure the target vector has the same length in both the general lattice and the modulo-q lattice.

Definition 5. *Let $q \in \mathcal{Z}$ be a modulus, and $\mathbf{v} = (v_0, v_1, \ldots, v_{d-1}) \in \mathcal{Z}^d$. The length of the vector $[\mathbf{v}]_q$ is defined as*

$$\|[\mathbf{v}]_q\| = \sqrt{\sum_{[v_i]_q \le q/2} ([v_i]_q)^2 + \sum_{[v_i]_q > q/2} (q - [v_i]_q)^2}. \tag{7}$$

Here $[v_i]_q$ is an integer in $\{0, \ldots, q-1\}$, and the operators "$\cdot$" and "$+$" in Eq. (7) are conducted over $\mathcal{Z}$ without modulo q.

For example, let $q = 16$ and $\mathbf{v} = (-3, 25, 8, 11, 16)$. Then we have

$$\begin{aligned} \|[\mathbf{v}]_q\|^2 &= (16 - [-3]_{16})^2 + (16 - [25]_{16})^2 + ([8]_{16})^2 + (16 - [11]_{16})^2 + ([16]_{16})^2 \\ &= 9 + 49 + 64 + 25 + 0 \\ &= 147. \end{aligned}$$

Since $q - [v_i]_q < [v_i]_q$ if $[v_i]_q > q/2$, we have $\|[\mathbf{v}]_q\| \le \|\mathbf{v}\|$. Besides, we also have $\|[\mathbf{v}]_q\|^2 \le d \cdot (q/2)^2$ for all $\mathbf{v} \in \mathcal{Z}^d$.

Definition 6 (Modulo-q lattice). *Let $q \in \mathcal{Z}$ be a modulus and $\mathbf{B} = \{\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}\} \subset \mathcal{Z}^d$. The modulo-q lattice generated by $\mathbf{B}$ is*

$$\mathcal{L}_q(\mathbf{B}) = \{\sum_{i=0}^{d-1} (v_i \cdot [\mathbf{b}_i]_q) \mid v_i \in \mathcal{Z}\} \subset \mathcal{Z}_q^d. \tag{8}$$

The dimension of the modulo-q lattice $\mathcal{L}_q(\mathbf{B})$ is d.

Lattice vs. modulo-q lattice The modulo-q lattice $\mathcal{L}_q(\mathbf{B})$ is a residue class ring of $\mathcal{L}(\mathbf{B})$ modulo the model generated by $\{(q,0,\ldots,0),\ (0,q,\ldots,0),\ \ldots,\ (0,0,\ldots,q)\}$ over $\mathcal{Z}$. If we abuse the notation that $\mathcal{Z}_q \subset \mathcal{Z}$, we have $\mathcal{L}_q(\mathbf{B}) \subset \mathcal{L}(\mathbf{B})$.

As a result, let $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ and $\mathbf{q}_i = (0,\ldots,q,\ldots,0)$, then $\mathbf{v}$ and $\mathbf{v}+\mathbf{q}_i$ are distinct vectors in $\mathcal{L}(\mathbf{B})$, but $[\mathbf{v}]_q = [\mathbf{v}+\mathbf{q}_i]_q$ in $\mathcal{L}_q(\mathbf{B})$. Besides, $\sum t_i \cdot [\mathbf{q}_i]_q = (0,\ldots,0) \in \mathcal{L}_q(\mathbf{B})$ for all $t_i \in \mathcal{Z}$.

Note that the number of distinct vectors in the modulo-q lattice is limited, because $\mathcal{L}_q(\mathbf{B}) \subset \mathcal{Z}_q^d$ and $|\mathcal{Z}_q^d| = q^d$, while there are infinite vectors in $\mathcal{L}(\mathbf{B})$.

Since all vectors in the modulo-q lattice are in $\mathcal{Z}_q^d$ instead of $\mathcal{R}^d$, float operations are not needed in the computations in the modulo-q lattice. Moreover, the orthogonal projection π_i used in [18, 6, 19] as well as SubSieve algorithms cannot be applied to the modulo-q lattice. We need to develop a new algorithm for computing shortest vectors in the modulo-q lattice, which will be shown in Sec. 3.3. Before presenting the sieving algorithm, we show how to solve the HNP, including 1-bit HNP, by the modulo-q lattice.

3.2 Resolving the HNP by the Modulo-q Lattice

To solve the HNP, we first use Albrecht and Heninger's method to construct the lattice. And next, instead of finding short vectors in a general lattice, our approach searches for short vectors in the modulo-q lattice.

Specifically, the basis $\mathbf{B}$ in Eq. (4) becomes:

$$\mathbf{B} = \begin{bmatrix} q & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & q & 0 & \cdots & 0 & 0 & 0 \\ & \vdots & & \ddots & & & \\ 0 & 0 & 0 & \cdots & q & 0 & 0 \\ \alpha'_1 & \alpha'_2 & \alpha'_3 & \cdots & \alpha'_{m-1} & 1 & 0 \\ \beta'_1 & \beta'_2 & \beta'_3 & \cdots & \beta'_{m-1} & 0 & w \end{bmatrix} \xrightarrow{[\cdot]_q} \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ & \vdots & & \ddots & & & \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ [\alpha'_1]_q & [\alpha'_2]_q & [\alpha'_3]_q & \cdots & [\alpha'_{m-1}]_q & [1]_q & 0 \\ [\beta'_1]_q & [\beta'_2]_q & [\beta'_3]_q & \cdots & [\beta'_{m-1}]_q & 0 & [w]_q \end{bmatrix},$$

where $w = 2^{l-1}$. The lattice $\mathcal{L}(\mathbf{B})$ generated by $\mathbf{B}$ becomes the modulo-q lattice $\mathcal{L}_q(\mathbf{B})$ generated by

$$\mathbf{a} = ([\alpha'_1]_q, \ldots, [\alpha'_{m-1}]_q, [1]_q, 0) \text{ and } \mathbf{b} = ([\beta'_1]_q, \ldots, [\beta'_{m-1}]_q, 0, [w]_q).$$

Let $\mathbf{s} = (-k'_1, -k'_2, \ldots, -k'_{m-1}, -k'_0, w)$ be the target vector in $\mathcal{L}(\mathbf{B})$ as discussed in Sec. 2.3, where $-w \le k'_i < w$ for $0 \le i < m$. It is easy to verify that the vector

$$\begin{aligned} [\mathbf{s}]_q &= ([-k'_1]_q, \ldots, [-k'_{m-1}]_q, [-k'_0]_q, [w]_q) \\ &= -k'_0 \cdot ([\alpha'_1]_q, \ldots, [\alpha'_{m-1}]_q, [1]_q, 0) + ([\beta'_1]_q, \ldots, [\beta'_{m-1}]_q, 0, [w]_q) \end{aligned}$$

is in the modulo-q lattice $\mathcal{L}_q(\mathbf{B})$. So $[\mathbf{s}]_q$ is the target vector we need to find in the modulo-q lattice $\mathcal{L}_q(\mathbf{B})$. Moreover, we also have

$$\|[\mathbf{s}]_q\| = \|\mathbf{s}\|.$$

Please note that the length of $[\mathbf{s}]_q$ is computed by Def. 5 because of $[\mathbf{s}]_q \in \mathcal{L}_q(\mathbf{B})$, while the length of $\mathbf{s}$ is the Euclidean norm.

Conversely, if we find a vector $\mathbf{v} = (v_0, \ldots, v_{m-1}, w) \in \mathcal{Z}_q^{m+1}$ in $\mathcal{L}_q(\mathbf{B})$ such that either $v_i \leq w$ or $q - v_i < w$ holds for $0 \leq i < m$. Then let $v'_i = v_i$ if $v_i \leq w$, and $v'_i = v_i - q$ if $q - v_i < w$, then the vector

$$\mathbf{v}' = (v'_0, \ldots, v'_{m-1}, w) \in Z^{m+1},$$

is a vector in $\mathcal{L}(\mathbf{B})$ such that $-w < v'_i \leq w$ for $0 \leq i < m$. As discussed in Sec. 2.3, the vector $\mathbf{v}'$ should just be the target vector $\mathbf{s}$ and hence, $\mathbf{v} = [\mathbf{s}]_q$ if $m > \log_2(q)$. Thus, the target vector $\mathbf{s}$ in $\mathcal{L}(\mathbf{B})$ can be obtained by searching for the above vector $\mathbf{v}$ in $\mathcal{L}_q(\mathbf{B})$.

When the target vector $[\mathbf{s}]_q$ is shortest in the modulo-q lattice $\mathcal{L}_q(\mathbf{B})$? We studied this question through experiments. In Sec. 4.1, we will see that the Gaussian heuristic does not work in the lattices constructed by Albrecht and Heninger's method when solving the 1-bit HNP. We think the reason is the lattices are very special in the 1-bit HNP because the q-vectors are very short.

According to our experiments, we have the conjecture that *If the dimension of $\mathcal{L}_q(\mathbf{B})$ is large enough, the target vector $[\mathbf{s}]_q$ is the shortest vector in $\mathcal{L}_q(\mathbf{B})$.* The difficulty in proving the above conjecture lies in that there are no theoretical methods to estimate the length of the shortest vector in the modulo-q lattice. But this conjecture is verified by experiments, and more details can be found in 4.1.

To reduce the time and space complexities in practical search for the target vector $[\mathbf{s}]_q$, similarly to what is done in [7], we also decrease the dimension of $\mathcal{L}_q(\mathbf{B})$, such that the target vector $[\mathbf{s}]_q$ may not be the shortest. And the tradeoff is that we need to check more vectors in the sieving list. The influences of the dimensions will be studied in Sec. 4.2.

3.3 A Sieving Algorithm to Search for the Target Vector

We present a sieving algorithm to search for the target vector in the modulo-q lattice. Consider the modulo-q lattice generated by

$$\mathbf{a} = ([\alpha'_1]_q, \ldots, [\alpha'_{m-1}]_q, [1]_q, 0) \text{ and } \mathbf{b} = ([\beta'_1]_q, \ldots, [\beta'_{m-1}]_q, 0, [w]_q).$$

The goal of the algorithm is to search for a vector $\mathbf{v} = (v_0, \ldots, v_{m-1}, w) \in \mathcal{Z}_q^{m+1}$ in this modulo-q lattice such that either $v_i \leq w$ or $q - v_i < w$ holds for $0 \leq i < m$.

In [7], the introduction of w in the last column of the vector $(\beta'_1, \ldots, \beta'_{m-1}, 0, w)$ of Eq. (4) is to ensure the multiples of this vector are longer than itself, and hence, to guarantee the target vector to be the shortest in the lattice. This technique is necessary if general lattice algorithm tools, e.g., G6K [6, 19], are used to search for short vectors because the vector $(\beta'_1, \ldots, \beta'_{m-1}, 0, w)$ may be multiplied by integers during the computation. But in our sieving algorithm, we can control all fundamental operations of vectors, and we do not need the last column anymore.

In our sieving algorithm, we consider the following two vectors

$$\bar{\mathbf{a}} = ([\alpha'_1]_q, \ldots, [\alpha'_{m-1}]_q, [1]_q), \bar{\mathbf{b}} = ([\beta'_1]_q, \ldots, [\beta'_{m-1}]_q, 0) \in \mathcal{Z}_q^m,$$

which are obtained from $\mathbf{a}$ and $\mathbf{b}$ by removing the last column. Correspondingly, the target vector $[\mathbf{s}]_q$ becomes

$$\bar{\mathbf{s}} = ([-k'_1]_q, \ldots, [-k'_{m-1}]_q, [-k'_0]_q) \in \mathcal{Z}_q^m.$$

In the algorithm, we need to search for a vector $\mathbf{v} = (v_0, \ldots, v_{m-1}) \in \mathcal{Z}_q^{m+1}$ such that $\mathbf{v} = t \cdot \bar{\mathbf{a}} + \bar{\mathbf{b}}$, where $t \in \mathcal{Z}$ and either $v_i \leq w$ or $q - v_i < w$ holds for $0 \leq i < m$. Please note that the expected squared length of $\mathbf{s}$ is $\mathbf{E}\left[\|\mathbf{s}\|^2\right] = m \cdot w^2/3 + m/6 + w^2$, but after removing the last column, we have

$$\mathbf{E}\left[\|\bar{\mathbf{s}}\|^2\right] = m \cdot w^2/3 + m/6.$$

In our sieving algorithm, to not consider the multiples of the vector $\bar{\mathbf{b}}$, we require all appeared vectors to have the form

$$\mathbf{v} = g \cdot \bar{\mathbf{a}} + h \cdot \bar{\mathbf{b}}, \tag{9}$$

where $g \in \mathcal{Z}_q$ and $h \in \{0, 1\}$. We denote $\text{type}(\mathbf{v}) = h$ and $\text{multi}(\mathbf{v}) = g$. Clearly, we have $\text{type}(\bar{\mathbf{a}}) = 0$, $\text{multi}(\bar{\mathbf{a}}) = 1$, $\text{type}(\bar{\mathbf{b}}) = 1$, and $\text{multi}(\bar{\mathbf{b}}) = 0$.

The algorithm is given in Alg. 2.

Algorithm 2: Sieving with built-in modulo arithmetic

Input: Two vectors $\bar{\mathbf{a}}, \bar{\mathbf{b}} \in \mathcal{Z}_q^m$.
Output: A target vector $\mathbf{v} = t \cdot \bar{\mathbf{a}} + \bar{\mathbf{b}} = (v_0, \ldots, v_{m-1}) \in \mathcal{Z}_q^m$ such that either $v_i \leq w$ or $q - v_i < w$ holds for $0 \leq i < m$, where $w = 2^{l-1}$.

1 **begin**
2 $L \longleftarrow \{g \cdot \bar{\mathbf{a}} + h \cdot \bar{\mathbf{b}} \mid g \in \mathcal{Z}_q, h \in \{0, 1\}\}$
3 $len \longleftarrow$ the size of L
4 $bound \longleftarrow$ the length of the longest vector in L
5 **while** *there is no* $\mathbf{v} = (v_0, \ldots, v_{m-1}) \in L$ *s.t. either* $v_i \leq w$ *or* $q - v_i < w$ *holds for* $0 \leq i < m$ **do**
6 $bucket_0, \ldots, bucket_r \longleftarrow \text{Bucket}(L)$
7 **for** *each* $bucket_i$ **do**
8 $L \longleftarrow L \cup \text{Sieve}(bucket_i, bound)$
9 $L \longleftarrow$ the first len short vectors in L
10 $bound \longleftarrow$ the length of the longest vector in L
11 return $\mathbf{v}$

Please note that all operations of vectors in Alg. 2 should apply the modulo operator $[\cdot]_q$ automatically. That is, the implementation of this algorithm needs built-in modulo q operations. For convenience, we call the steps $6 \sim 10$ as a

round of sieving. So the sieving list L is updated every round. We usually decide whether to terminate the algorithm depending on the number of new vectors generated in each round. For example, we can stop the program if less than 100 new vectors are generated in some round.

To ensure the target $\mathbf{v}$ can be found within finite rounds, we require the size of the list L to be at least $2^{0.2075m}$.

There are two functions in Alg. 2, Bucket$(\cdot)$ and Sieve$(\cdot)$.

Bucketing Like other sophisticated sieving algorithms, we do not check all pairs of vectors in L and only deal with the pairs selected from a small subset of L. Thus, the function Bucket$(\cdot)$ takes a list of vectors as input and outputs r sets of vectors. We only need to consider the pairs in the output sets.

However, since the operations in the modulo-q lattice are not general vector addition and multiplication by integers, the angles between vectors cannot be used for bucketing. Thus, existing bucketing techniques, including [12, 11], do not work in the modulo-q lattice. We need new bucketing methods in the modulo-q lattice.

We think the key insight of bucketing techniques in [12], is that *if* $\mathbf{c}$ *is a center vector, then the vectors that are "close" to* $\mathbf{c}$ *are also "close" to each other.* In the general lattice, "close" implies the angle of two vectors is small. To develop a new bucketing method in the modulo-q lattice, we only need to define "close". After many failed experiments, we find the naive definition of "close" is good enough.

Let $\mathbf{v}$ and $\mathbf{c}$ be two vectors in the modulo-q lattice having the form of Eq. (9), we define the **difference vector** of $\mathbf{v}$ and $\mathbf{c}$ as

$$\mathrm{Diff}(\mathbf{v}, \mathbf{c}) = \begin{cases} \mathbf{v} - \mathbf{c}, & \mathrm{type}(\mathbf{v}) \geq \mathrm{type}(\mathbf{c}), \\ \mathbf{c} - \mathbf{v}, & \mathrm{type}(\mathbf{v}) < \mathrm{type}(\mathbf{c}). \end{cases} \tag{10}$$

And the **distance** between $\mathbf{v}$ and $\mathbf{c}$ is

$$\mathrm{Dist}(\mathbf{v}, \mathbf{c}) = \|\mathrm{Diff}(\mathbf{v}, \mathbf{c})\|.$$

The above definition guarantees the coefficient of $\bar{\mathbf{b}}$ in the difference vector $\mathrm{Diff}(\mathbf{v}, \mathbf{c})$ is still in $\{0, 1\}$, which is important to our implementation. Please remember that $\|\cdot\|$ is the length in the modulo-q lattice, defined in Def. 5. The computation of $\|\cdot\|$ can be done efficiently using GPU devices. Therefore, given two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, we say $\mathbf{v}_1$ is *closer* to $\mathbf{c}$ than $\mathbf{v}_2$ if $\mathrm{Dist}(\mathbf{v}_1, \mathbf{c})$ is smaller than $\mathrm{Dist}(\mathbf{v}_2, \mathbf{c})$.

The idea of bucketing is that given a center vector $\mathbf{c}$ and the size of a bucket, say $bucket\_size$, we compute the distances between $\mathbf{c}$ and the vectors in the list L and then choose the first $bucket\_size$ vectors that are closer to $\mathbf{c}$ than others. The procedure is shown in Alg. 3.

Clearly, we have $\mathbf{c}_i \in L$ and $\mathrm{Dist}(\mathbf{c}_i, \mathbf{c}_i) = 0$, so $\mathbf{c}_i \in bucket_i$ holds.

Note that, in Alg. 3, to find the first $bucket\_size$ vectors in L that are close to $\mathbf{c}_i$, it is not necessary to sort the set D, whose time complexity is $\log(|L|) \cdot |L|$.

Algorithm 3: Bucketing in the modulo-q lattice

Input: a list L of vectors in the modulo-q lattice.
Output: r sets of vectors.

```
begin
    c_0, ..., c_{r-1} ⟵ r center vectors from L
    for each c_i do
        compute the set D_i = {(Dist(v, c_i), v) | v ∈ L}
        bucket_i ⟵ find the first bucket_size vectors in L that are closer to c_i
          using D_i
    return bucket_0, ..., becket_{r-1}
```

We use a sampling method whose time complexity is only $\log(bucket\_size) \cdot bucket\_size$.

For the value of $bucket\_size$, we follow the strategy in G6K [19] and set $bucket\_size$ as a power of 2 that approximates $(2 \cdot |L|)^{1/2}$. The value of r is determined by the global memory size on GPU devices but is not bigger than $|L|/bucket\_size$.

Sieving We designed the sieving function following G6K [6], which will benefit the GPU implementation. The pseudo codes are given in Alg. 4.

Algorithm 4: Sieving in the modulo-q lattice

Input: A set of vectors, $bucket_i$; an integer $bound$.
Output: A set of new vectors.

```
begin
    new ⟵ ∅
    for v_i, v_j ∈ bucket_i do
        v = Diff(v_i, v_j)
        if ||v|| < bound then
            new ⟵ new ∪ {v}
return new
```

In Alg. 4., the coefficient of $\bar{\mathbf{b}}$ in $\mathbf{v}$ is still in $\{0, 1\}$ by the definition of $\mathrm{Diff}(\mathbf{v}_i, \mathbf{v}_j)$ in Eq. (10). Unlike G6K [6], we do not check the vector $\mathbf{v}_i + \mathbf{v}_j$ when $\mathrm{type}(\mathbf{v}_i) + \mathrm{type}(\mathbf{v}_j) \leq 1$, because the addition test costs equivalent computations compared to the subtraction test but generates much fewer new vectors in our experiments.

3.4 Decrease the Dimension of the Modulo-q Lattice

The technique of breaking the "lattice barrier", proposed in [7], shares the idea of the technique of "dimensions for free", proposed in [18]. Both techniques aim to reduce the dimension of the lattice by using more short vectors in the sieving list. Our method of solving 1-bit HNP by modulo-q lattice can also use this idea. Specifically, instead of searching for the shortest vector in the m-dimensional modulo-q lattice, we can reduce the dimension of this modulo-q lattice to $m' < m$. In this case, the target vector may not be the shortest anymore, but we could expect the target vector is still relatively short if m' is not far away from m. If so, we could have a high probability of finding the target vector in the sieving list.

4 Experiments on Solving the 1-bit HNP

As there are no existing sieving methods for solving the 1-bit HNP, we conduct several experiments to study the characteristics of the 1-bit HNP and the performance of the sieving algorithm. Through these experiments, we want to verify the following two observations.

1. If the dimension of the modulo-q lattice is large enough, the target vector is the shortest.
2. Increasing the dimension of the modulo-q lattice could lower the number of vectors that are shorter than the target vector.
3. If the number of vectors that are shorter than the target vector is small, then the target vector will be found with a high probability.

The first observation will be verified in Sec. 4.1. The second and third observations will be studied in Sec. 4.2 and 4.3. The performances of our implementation on solving large 1-bit HNP are shown in Sec. 4.4.

The experiments in this section were conducted on two platforms.

- One computer with 1 AMD EPYC 7763 CPU (64 cores, 128 threads), 1 TB RAM, and 2 NVIDIA RTX 3080Ti cards, running Ubuntu 20.04.
- Two computers, each has 1 AMD Threadripper 3990x (64 cores, 128 threads), 256 GB RAM, and 2 NVIDIA RTX 4090 cards, running Ubuntu 20.04. The two computers communicate data with each other through the intranet.

The experiments in Sec. 4.1 and 4.2 were performed on the first platform. The times in Sec. 4.4 were obtained from the second platform.

4.1 Minimum Dimension such that the Target Vector Is the Shortest

In this section, we study by experiments how many dimensions are necessary to ensure the target vector is the shortest in the modulo-q lattice. For this goal, our experiments were conducted in the following way, assuming the value of q is given first.

1. Choose a dimension $m \geq \log_2(q)$.
2. Generate a solution x and the data (α_i, β_i) for $0 \leq i < m$ by choosing a seed such that the squared length of the target vector $\bar{\mathbf{s}}$ equals approximately the expected squared length of the target vectors.
3. Generate at least $2^{0.2075m}$ data randomly, and start sieving by Alg. 2.
4. If a vector $\mathbf{v}$ is found such that $\|\mathbf{v}\| < \|\bar{\mathbf{s}}\|$, then increase m to $m+1$ and go to step 2.
5. If the number of new vectors generated in some round of Alg. 2 is smaller than 100, then stop the program[2], and m should be the minimum dimension such that the target vector is the shortest.

In Tab. 1, m is the minimum dimension obtained from experiments. Although $\bar{\mathbf{s}}$ is used in our sieving algorithm, to compare with the values obtained by the Gaussian heuristic of $\mathcal{L}(\mathbf{B})$, we consider in this table the target vector $\mathbf{s} = (-k'_1, -k'_2, \ldots, -k'_{m-1}, -k'_0, w)$ which is in the lattice generated by $\mathbf{B}$ in Eq. (4). The values of $\mathbf{E}[\|\mathbf{s}\|^2]$ and $\mathrm{gh}(\mathcal{L}(\mathbf{B}))$ are calculated by Eq. (6) and (5). " $\#q$ vectors" shows the number of q-vectors that are shorter than $\|\mathbf{s}\|$. Fig. 2 shows the variations of m with respect to different q's.

$\log_2(q)$	m	$\log_2(\mathbf{E}[\|\mathbf{s}\|^2])$	$\log_2(\|\mathbf{s}\|^2)$	$\log_2^2(\mathrm{gh}(\mathcal{L}(\mathbf{B})))$	$\#q$ vectors
64	82	128.772590	128.772572	128.690487	162
68	86	136.841302	136.841309	136.739557	170
72	92	144.938599	144.938597	144.853570	182
76	99	153.044394	153.044391	152.989665	19208
80	102	161.087463	161.087440	161.000076	20402
84	107	169.156504	169.156505	169.068104	22472
88	115	177.260528	177.260524	176.538868	25992
92	120	185.321928	185.321926	184.382241	28322
96	126	193.392317	193.392316	192.223167	31250

Table 1. The minimum dimensions such that the target vectors are the shortest.

From Tab. 1, we can see that for any q, we can always find a dimension m in our experiment. Although we cannot prove this dimension is the minimum dimension in theory, we believe such a minimum dimension does exist, which verifies the first observation in Sec. 4. Fig. 2 shows that the values of m vary almost linearly with respect to $\log_2(q)$, which is a bit surprising to us. From Tab. 1, we also note that the values obtained by the Gaussian heuristic are always smaller than the squared lengths of the shortest vectors in the modulo-q lattice. So we think the Gaussian heuristic probably fails to estimate the shortest vector in $\mathcal{L}(\mathbf{B})$ because there are many short q-vectors in $\mathcal{L}(\mathbf{B})$, which makes the lattice $\mathcal{L}(\mathbf{B})$ special.

[2] We do not stop the program immediately after finding the target vector.

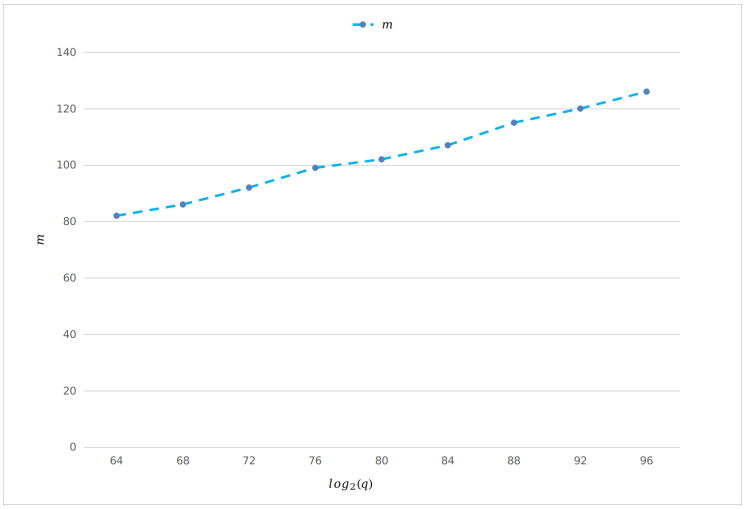

Fig. 2. The variations of minimum dimensions with respect to different q's.

4.2 Influences of the Dimensions of the Modulo-q Lattice

As discussed in Sec. 3.4, we can use Albrecht and Heninger's idea in [7] to decrease the dimensions of the lattices for speed-up. But the tradeoff is that there may appear some vectors that are shorter than the target vector, which makes the target vector hard to find. In this section, we study by experiments the influences of dimensions on finding the target vectors. For this goal, we set $q = 2^{100}$ and solve the 1-bit HNP in the modulo-q lattices of different dimensions.

The experiments were conducted as follows.

1. Choose a dimension m in $\{100, 104, 108, 112, 116, 120\}$.
2. Generate a solution x and the data (α_i, β_i) for $0 \leq i < m$ by choosing a seed such that the squared length of the target vector $\bar{\mathbf{s}}$ equals approximately the expected squared length of the target vectors.
3. Generate at least $2^{0.2075m}$ data randomly, and start sieving by Alg. 2.
4. Stop the program immediately after finding the target vector.
5. Repeat step $2 \sim 4$ for 5 times with respect to the same dimension to obtain an average time.

In Tab. 2, m is the dimension of the modulo-q lattice, and $\log_2(|L|)$ shows the size of the sieving list L. Here the target vector is $\bar{\mathbf{s}} = ([-k'_1]_q, \ldots, [-k'_{m-1}]_q, [-k'_0]_q)$ in the modulo-q lattice, but is not $\mathbf{s}$ in $\mathcal{L}(\mathbf{B})$. We also show the number of vectors that are shorter than $\bar{\mathbf{s}}$ in the column $\#(< \|\bar{\mathbf{s}}\|)$. The vector $\mathbf{v}_{min}$ is the shortest vector in the list when the target vector is found.

From Tab. 2, we can see that if the dimension increases, the number of vectors that are shorter than the target vector drops quickly. But the cost is that the size of the sieving list becomes larger, so the time for solving the HNP rises. In

m	$\log_2(\lvert L\rvert)$	$\log_2(\mathbf{E}[\lVert\bar{\mathbf{s}}\rVert^2])$	$\log_2(\lVert\bar{\mathbf{s}}\rVert^2)$	$\log_2(\lVert\mathbf{v}_{min}\rVert^2)$	$\#(< \lVert\bar{\mathbf{s}}\rVert)$	time
100	23	201.058894	201.058894	200.635365	4771494	fail
104	23	201.115477	201.115477	200.664383	603260	2977s
108	24	201.169925	201.169925	200.855302	75458	4218s
112	24	201.222392	201.222392	200.971528	9414	7280s
116	25	201.273018	201.273018	201.054592	1156	9118s
120	25	201.321928	201.321929	201.219587	143	27817s

Table 2. Influences of dimensions. The times are given in seconds.

the above experiment, we cannot find the target vector when the dimension is 100. We think this is because many vectors are shorter than the target vector. Tab. 2 also verifies the second observation in Sec. 4.

4.3 Influences of the Lengths of the Target Vectors

For a random 1-bit HNP, the squared length of the target vector $\bar{\mathbf{s}}$ approximates $\mathbf{E}(\lVert\bar{\mathbf{s}}\rVert^2)$. But we observed that the lengths of different target vectors do affect the solving procedure of the HNP, particularly for large problems. In this section, we study the influences of the lengths of the target vectors. In these experiments, we set $q = 2^{100}$ and $m = 108$.

Firstly, we study the distribution of the lengths of the target vectors, by the following steps.

1. Generate a solution x and the data (α_i, β_i) for $0 \le i < m$ randomly.
2. Calculate the length of the target vector $\bar{\mathbf{s}} = ([-k'_1]_q, \ldots, [-k'_{m-1}]_q, [-k'_0]_q)$.
3. Repeat step $1 \sim 2$ for 10^6 times.

The distribution of the squared lengths of the target vectors is shown in Fig. 3.

From Fig. 3, we can see that the distribution of the lengths is approximately a Gaussian distribution. Interestingly, the expected squared length of the target vector does not lie at the strict center. We think this is because the value $\mathbf{E}[\lVert\mathbf{s}\rVert]^2$ is slightly smaller than $\mathbf{E}[\lVert\mathbf{s}\rVert^2]$. Note that $\mathbf{E}[\lVert\mathbf{s}\rVert]$ is a bit hard to estimate, which is why the value $\mathbf{E}[\lVert\mathbf{s}\rVert^2]$ is used in [7].

Secondly, we study the differences when the target vectors have different lengths, and the experiments were performed in the following way.

1. Let the 10^6 target vectors generated above be sorted with respect to their lengths, and let the shortest one lie in the first position. Choose a target vector at the position $p{\cdot}10^6$ where $1-(1-p)^t = 0.5$ and $t \in \{5, 4, 3, 2, 1, 1/2, 1/3\}$.
2. Generate at least $2^{0.2075m}$ data randomly, and start sieving by Alg. 2.
3. Stop the program only when the number of new vectors generated in some round of Alg. 2 is smaller than 100, and store the final sieving list.
4. Repeat step $2 \sim 4$ for 5 times for each t to obtain an average time of finding the target vector.

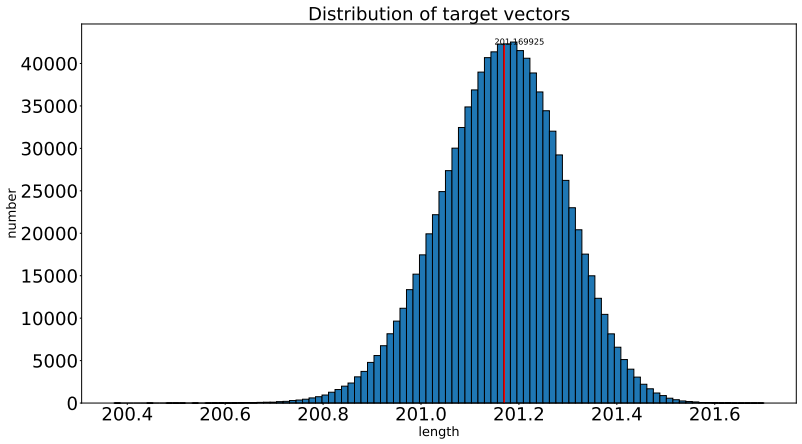

Fig. 3. The distribution of the lengths of the target vectors.

Note that if a target vector $\bar{\mathbf{s}}$ is at the position $p \cdot 10^6$ of the sorted list of target vectors, it means that the target vector of a random HNP has the probability p to be no longer than $\bar{\mathbf{s}}$. For simplification, we say the target vector $\bar{\mathbf{s}}$ is *at the position* p or *at* p for short. The selection of p will be explained in the last part of this subsection.

In Tab. 3, the values $\log_2(\mathbf{E}[\|\bar{\mathbf{s}}\|^2])$, $\log_2(\|\bar{\mathbf{s}}\|^2)$, and $\log_2(\|\mathbf{v}_{min}\|^2)$ are the expected squared lengths of the target vectors, and the squared lengths of true target vectors, and the squared lengths of the shortest vectors in the list. Here, $\#(< \mathbf{E}[\|\bar{\mathbf{s}}\|^2])$ and $\#(< \|\bar{\mathbf{s}}\|^2)$ show the numbers of vectors in the final list whose squared lengths are shorter than $\mathbf{E}[\|\bar{\mathbf{s}}\|^2]$ and $\|\bar{\mathbf{s}}\|^2$, respectively. Please note that the times in Tab. 3 are the running times for obtaining the target vectors but not the time the programs stop.

t	p	$\log_2(\mathbf{E}[\|\bar{\mathbf{s}}\|^2])$	$\#(< \mathbf{E}[\|\bar{\mathbf{s}}\|^2])$	$\log_2(\|\bar{\mathbf{s}}\|^2)$	$\#(< \|\bar{\mathbf{s}}\|^2)$	$\log_2(\|\mathbf{v}_{min}\|^2)$	time
5	0.129449	201.169925	77406	201.022898	305	200.872789	3020s
4	0.159103	201.169925	77401	201.040397	623	200.838928	3257s
3	0.206299	201.169925	77135	201.063997	1471	200.873948	3682s
2	0.292893	201.169925	76798	201.099610	5497	200.742548	3674s
1	0.500000	201.169925	77294	201.168665	72933	200.878865	3988s
1/2	0.750000	201.169925	76729	201.251008	1381122	200.858196	4986s
1/3	0.875000	201.169925	77517	201.306621	8694745	200.888406	fail

Table 3. Influences of the lengths of target vectors. The times are given in seconds.

Firstly, from Tab. 3, we can see that, *the shorter the target vector, the faster it can be found.* In other words, if we could find a target vector at the position p in time T, then we can find a target vector at the position $p' < p$ using no more than the time T. We think there are two reasons for this phenomenon. The first one is that the shorter the target vector, the fewer vectors are shorter than the target vector, which is also shown in Tab. 3. The second reason is that our sieving algorithm tends to generate shorter vectors in each round, as shown in Fig. 4. The vectors generared in each round are obatined using 32 threads in our implementation. We classify the new vectors generated in each round into 32 groups according to the lower and upper bounds. Some numbers of vectors in the groups are shown in Fig. 4. We can see that the numbers of shorter vectors are more than those of longer ones. Although the figures are only drawn based on the data from two specific rounds, this phenomenon happens almost in every round of our sieving algorithm. So in all, because fewer vectors are shorter than the target vector at the position p' than at p and our sieving algorithm tends to generate more short vectors, the probability of finding the target vector at p' is relatively higher than finding that at p.

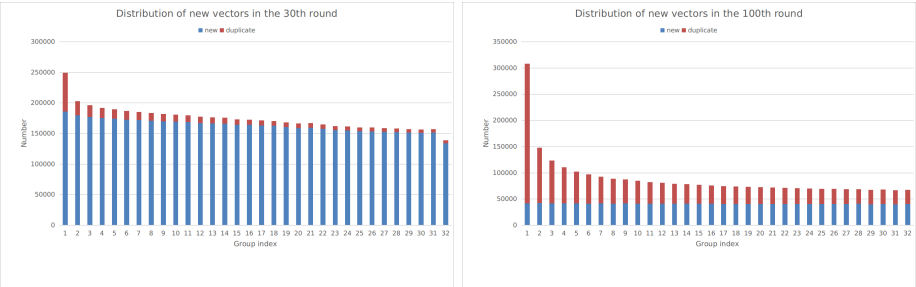

Fig. 4. The distributions of new generated vectors in the 30th and 100th rounds when we solve the HNP with $q = 2^{100}$ and $m = 108$. "*new*" refers to the number of vectors not existing in the list, while "*duplicate*" is the number of vectors that are already in the list.

Secondly, we can see from Tab. 3 that, *the distributions of vectors in the final lists are almost the same if q and m are fixed, no matter what the values of nonce data are.* We draw the distributions of the vectors in the seven final lists in Fig. 5. The bottom right figure is obtained by putting all 7 figures together. We can see that the distributions of the lists match very well.

The last thing we can read from Tab. 3 is that the target vector is very difficult to find if there are enormous vectors shorter than the target vector.

4.4 On Solving Large 1-bit HNPs

In this section, we show the performance of our implementation on solving large 1-bit HNPs. Our second platform is used for this computation, and the two

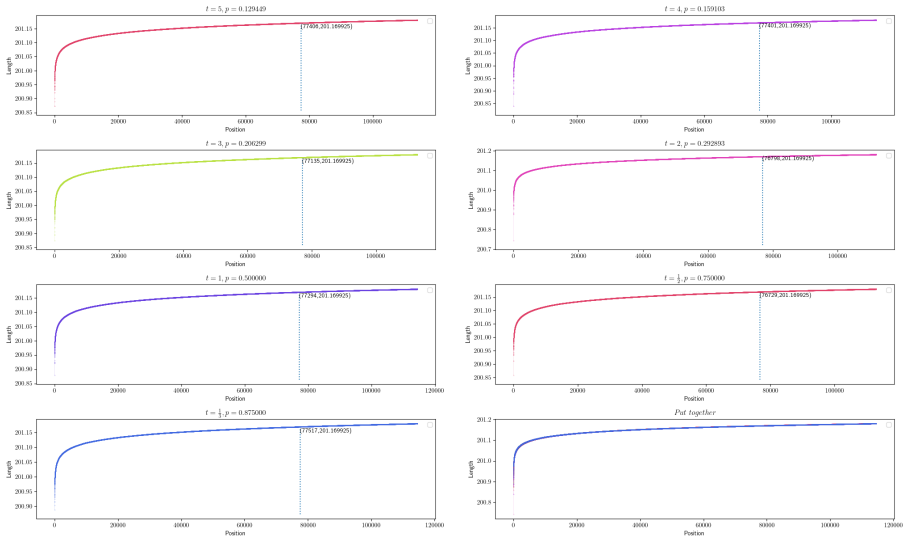

Fig. 5. The distributions of vectors in the final lists. The x-axis gives the positions of vectors in the lists, while the y-axis shows the squared lengths of the vectors.

computers communicate new short vectors with each other through the intranet. Particularly, it costs about 100 GB memory on the host of each computer to solve the 1-bit HNP ($q = 2^{128}$).

We solved 1-bit HNP ($q = 2^{120}$) and 1-bit HNP ($q = 2^{128}$) where the lengths of solutions equal to the expected lengths. The size of the list is set as a power of 2 and is larger than $2^{0.2075m}$.

$\log_2(q)$	m	$\log_2(\mathbf{E}[\|\|\mathbf{s}\|\|^2])$	$\#(< \mathbf{E}[\|\|\mathbf{s}\|\|^2])$	$\log_2(\|\|\mathbf{s}\|\|^2)$	$\#(< \|\|\mathbf{s}\|\|^2)$	time
120	138	241.523562	11950	241.523562	11950	118.32h
128	140	257.544321	257804	257.544321	257804	388.21h

Table 4. On solving large 1-bit HNPs. The times are given in hours.

To verify the data in Tab. 4, we could provide the nonce data, the lattices, as well as the vectors that are shorter than $\log_2(\mathbf{E}[\|\|\mathbf{s}\|\|^2])$.

5 Conclusion and Further Improvements

In this paper, we present a new sieving approach for solving the 1-bit HNP using built-in modulo arithmetic. Compared with the previous state-of-the-art lattice method of solving HNP, the proposed approach has two improvements. Firstly, by using the residue class ring of the lattice, the number of vectors that are shorter than the target vector is significantly reduced, such that the target vector becomes easier to find. Secondly, the residue classes of vectors are used instead of vectors during the computation; many duplicated computations are avoided. Thus, the proposed algorithm is efficient for solving large 1-bit HNPs.

References

1. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing. pp. 601–610. STOC '01, Association for Computing Machinery, New York, NY, USA (2001)
2. Akavia, A.: Solving hidden number problem with one bit oracle and advice. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 337–354. Springer, Heidelberg (Aug 2009)
3. Albrecht, M.R., Bai, S., Fouque, P.A., Kirchner, P., Stehlé, D., Wen, W.: Faster enumeration-based lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 186–212. Springer, Heidelberg (Aug 2020)
4. Albrecht, M.R., Cid, C., Faugère, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the bkw algorithm on lwe. Designs, Codes and Cryptography (2015)
5. Albrecht, M.R., Deo, A., Paterson, K.G.: Cold boot attacks on ring and module LWE keys under the NTT. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 173–213 (Aug 2018)
6. 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, Part II. LNCS, vol. 11477, pp. 717–746. Springer, Heidelberg (May 2019)
7. Albrecht, M.R., Heninger, N.: On bounded distance decoding with predicate: Breaking the "lattice barrier" for the hidden number problem. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 528–558. Springer, Heidelberg (Oct 2021)
8. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
9. Aranha, D.F., Fouque, P.A., Gérard, B., Kammerer, J.G., Tibouchi, M., Zapalowicz, J.C.: GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 262–281. Springer, Heidelberg (Dec 2014)
10. Aranha, D.F., Novaes, F.R., Takahashi, A., Tibouchi, M., Yarom, Y.: LadderLeak: Breaking ECDSA with less than one bit of nonce leakage. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 225–242. ACM Press (Nov 2020)
11. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th SODA. pp. 10–24. ACM-SIAM (Jan 2016)

12. Becker, A., Gama, N., Joux, A.: Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522 (2015), https://eprint.iacr.org/2015/522
13. Bleichenbacher, D.: On the generation of one-time keys in DL signature schemes. Presentation at IEEE P1363 working group meeting (2000)
14. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM **50**(4), 506–519 (July 2003)
15. 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'96. LNCS, vol. 1109, pp. 129–142. Springer, Heidelberg (Aug 1996)
16. Coppersmith, D.: Finding a small root of a bivariate integer equation; Factoring with high bits known. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 178–189. Springer, Heidelberg (May 1996)
17. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 329–358. Springer, Heidelberg (Aug 2020)
18. Ducas, L.: Shortest vector from lattice sieving: A few dimensions for free. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 125–145. Springer, Heidelberg (Apr / May 2018)
19. Ducas, L., Stevens, M., van Woerden, W.P.J.: Advanced lattice sieving on GPUs, with tensor cores. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 249–279. Springer, Heidelberg (Oct 2021)
20. Fincke, U., Pohst, M.E.: Improved methods for calculating vectors of short length in a lattice. Mathematics of Computation (1985)
21. Gama, N., Nguyen, P.Q.: Finding short lattice vectors within Mordell's inequality. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing. pp. 207–216. STOC '08, Association for Computing Machinery, New York, NY, USA (2008)
22. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257–278. Springer, Heidelberg (May / Jun 2010)
23. Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: Solving LWE using lattice codes. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 23–42. Springer, Heidelberg (Aug 2015)
24. Hanrot, G., Stehlé, D.: Improved analysis of kannan's shortest lattice vector algorithm. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 170–186. Springer, Heidelberg (Aug 2007)
25. Herold, G., Kirshanova, E.: Improved algorithms for the approximate k-list problem in euclidean norm. In: Fehr, S. (ed.) Public-Key Cryptography – PKC 2017. pp. 16–40. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)
26. Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving LWE. Designs, Codes and Cryptography (2018)
27. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. Designs, Codes and Cryptography **23**(3), 283–290 (July 2001)
28. Jancar, J., Sedlacek, V., Svenda, P., Sys, M.: Minerva: The curse of ECDSA nonces : Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4), 281–308 (August 2020)
29. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: 15th ACM STOC. pp. 193–206. ACM Press (Apr 1983)

30. Kirchner, P., Fouque, P.A.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 43–62. Springer, Heidelberg (Aug 2015)
31. Klein, P.N.: Finding the closest lattice vector when it's unusually close. In: Shmoys, D.B. (ed.) 11th SODA. pp. 937–941. ACM-SIAM (Jan 2000)
32. Laarhoven, T.: Search problems in cryptography: From fingerprinting to lattice sieving. Ph.D. thesis, Eindhoven University of Technology (2015)
33. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen (1982)
34. Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: An update. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer, Heidelberg (Feb / Mar 2013)
35. Merget, R., Brinkmann, M., Aviram, N., Somorovsky, J., Mittmann, J., Schwenk, J.: Raccoon attack: Finding and exploiting most-significant-bit-oracles in TLS-DH(E). In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 213–230. USENIX Association (Aug 2021)
36. Micciancio, D., Regev, O.: Lattice-based Cryptography, pp. 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
37. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: Charika, M. (ed.) 21st SODA. pp. 1468–1480. ACM-SIAM (Jan 2010)
38. Micciancio, D., Walter, M.: Fast lattice point enumeration with minimal overhead. In: Indyk, P. (ed.) 26th SODA. pp. 276–294. ACM-SIAM (Jan 2015)
39. Micciancio, D., Walter, M.: Practical, predictable lattice basis reduction. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 820–849. Springer, Heidelberg (May 2016)
40. Moghimi, D., Sunar, B., Eisenbarth, T., Heninger, N.: TPM-fail: TPM meets timing and lattice attacks. In: Proceedings of the 29th USENIX Conference on Security Symposium. SEC'20, USENIX Association, USA (2020)
41. Nemec, M., Sýs, M., Svenda, P., Klinec, D., Matyas, V.: The return of Coppersmith's attack: Practical factorization of widely used RSA moduli. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1631–1648. ACM Press (Oct / Nov 2017)
42. Nguyen, P.Q., Shparlinski, I.: The insecurity of the digital signature algorithm with partially known nonces. Journal of Cryptology **15**(3), 151–176 (Jun 2002)
43. Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. Journal of Mathematical Cryptology **2**(2), 181–207 (2008)
44. Pohst, M.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. SIGSAM Bull. **15**(1), 37–44 (February 1981)
45. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005)
46. Ryan, K.: Return of the hidden number problem: A widespread and novel key extraction attack on ECDSA and DSA. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 146–168 (November 2018)
47. Ryan, K.: Hardware-backed Heist: Extracting ECDSA keys from Qualcomm's trustzone. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 181–194. CCS '19, Association for Computing Machinery, New York, NY, USA (2019)

48. Schnorr, C.P.: A hierarchy of polynomial time lattice basis reduction algorithms. Theor. Comput. Sci. **53**(2), 201–224 (June 1987)
49. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming (1994)
50. Takahashi, A., Tibouchi, M., Abe, M.: New bleichenbacher records: Fault attacks on qDSA signatures. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 331–371 (August 2018)
51. Weiser, S., Schrammel, D., Bodner, L., Spreitzer, R.: Big numbers - Big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations. In: Proceedings of the 29th USENIX Conference on Security Symposium. SEC'20, USENIX Association, USA (2020)