Cloud-Assisted LLL: A Secure and Efficient Outsourcing Algorithm for Approximate Shortest Vector Problem

Xiulan Li[1,2], Yanbin Pan[1,2], and Chengliang Tian[3](✉)

[1] Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
{lixiulan,panyanbin}@amss.ac.cn

[2] School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China

[3] College of Computer Science and Technology, Qingdao University, Qingdao 266071, China
tianchengliang@qdu.edu.cn

Abstract. Approximating the shortest vector of a given lattice is one of the most important computational problems in public-key cryptanalysis and lattice-based cryptography. However, existing LLL reduction algorithm and its variants for this problem are too time-consuming for resource-constrained clients. To handle this dilemma, in this paper, we propose an efficient and secure outsourcing algorithm under the cloud environment. Compared with the prior Liu et al.'s algorithm, besides realizing the privacy preservation of client's input/output information, satisfying verifiability and greatly reducing the local-client's computational overhead, our algorithm is superior in the following aspects. First, our algorithm is technically concise. The main technique ingredient involved in our algorithm is a skillful combination of the unimodular matrix transformation and the Gram matrix, which is concise and effective. Second, our algorithm does not reduce the quality of the reduced basis, that is, the vector finally obtained by the client is as short as that of the vector generated by the client directly performing the existing reduction algorithm. Last but not least, our algorithm not only works for the LLL reduction algorithm, but also for any other algorithms that solve (approximate-)SVP with Euclidean norm.

Keywords: Cloud computing · Outsourcing computation · Approximate SVP · Gram matrix · Unimodular transformation

This work is supported by National Key Research and Development Program of China (No. 2018YFA0704705, 2020YFA0712300), National Natural Science Foundation of China (No. 61702294, 62032009), National Development Foundation of Cryptography (MMJJ20170126).

1 Introduction

A lattice is a discrete additive subgroup of $\mathbb{R}^m$, which is a classic research object in the geometry of numbers. One of the most important computational problems in lattice theory is the Shortest Vector Problem (SVP) which aims to find the shortest nonzero vector of arbitrary given lattice and is shown to be NP-hard under random reduction [1].

Lattice has been widely used in mathematics and computer science, especially in cryptography. Many different problems can be solved via solving SVP in some lattices, such as factoring polynomials with rational coefficients [15], finding integer relations among real numbers [11], factoring integers and computing discrete logarithms [22], and attacking RSA [7]. In addition, lattice-based cryptographic constructions [10] have been widely considered as one of the most promising post-quantum cryptosystems and the hardness of SVP directly determines their theoretical security.

Due to the hardness of SVP, various lattice basis reduction algorithms have been designed to approximate shortest vector as far as possible. The first lattice basis reduction algorithm proposed by Lagrange [14] in 1773 is a groundbreaking work, though it finds a minimal basis in two dimensions. Hermite's proposed second lattice basis reduction algorithm generalizes Lagrange's algorithm to n dimensions. After that, the first polynomial-time lattice basis reduction algorithm, the LLL algorithm, was proposed by Lenstra, Lenstra and Lovász in 1982. It achieves an approximation factor of $2^{O(n)}$ for the approximation variant of SVP with worst-case time complexity $O(n^5 m \log^3 B)$, and can be used to attack some cryptosystems, such as knapsack-based cryptosystems and special cases of RSA [7]. To get somewhat better approximation factor $(6k^2)^{nk/2}$ where k is some integer, Schnorr [21] extended the LLL algorithm by making block size larger at the price of an increased running time in 1987.

In the last two decades, many improvements for lattice basis reduction algorithm have been investigated [2,6,18,19]. In 2005, Nguyen and Stehlé [18] introduced L^2 algorithm, a floating-point variant of L^3, to make LLL reduction algorithm practical with computational complexity $O(n^4 m(n+\log B)\log B)$. Saruchi et al. [20] proposed an effective reduction algorithm, which is the expansion of the algorithm proposed by Bi et al. [4]. Although these algorithms can find a relatively short vector in polynomial time, they are still time-consuming in practice. Especially, in practical applications, the dimension of the lattice is usually very large and the user or the terminal device could be with limited computing and storage capability. It is unrealistic for these clients to perform these algorithms to approximate the shortest vector.

The emergence of cloud computing provides a new paradigm for resource-constrained clients to handle heavy computational tasks, in which scenario, these resource-constrained clients can outsource their overloaded computational task to the resource-abundant cloud server on a pay-as-you-use manner. However, this promising computing paradigm also brings new security concerns [13,25]. The remote cloud server is out of control, and for the sake of business interests, it could deviate the prescribed execution rules and collect valuable information.

Simultaneously, the outsourcing computational task may contain client's privacy and sensitive information. The exposure of this information may cause critical loss of lift and property. Therefore, a well-designed outsourcing algorithm, apart from ensuring the client to achieve the correct computation result at a greatly reduced time cost, should protect the client's privacy information and discern the cloud server's misbehaviors. Therefore, in-depth studies on the outsourcing computation of many aspects have been conducted in recent years, such as large-scale linear algebra operations [3,8], solving quadratic congruences [26,27], modular exponentiations [5,12,29] and modular inversion [24] in cryptography, heavy computations in artificial intelligence (AI) and Internet of Things (IoT) [16,28].

In 2019, Liu et al. [17] proposed the first outsourcing computation mechanism of lattice-reduction algorithm based on the work of Saruchi et al. [20]. They utilized rounding technique and unimodular transformation matrix to encrypt the original computation task by generating a outsourcing task $\mathbf{B} + \varDelta\mathbf{B}$ for the lattice basis $\mathbf{B}$. The cloud server reduces it by LLL-reduction algorithm and returns the transformation matrix. After receiving the response of the cloud server, the client can obtain the LLL-reduced basis for the target lattice $\varLambda(\mathbf{B})$ finally, but with a bigger approximation factor. Their outsourcing algorithm has high-efficiency. However, the perturbation term $\varDelta\mathbf{B}$ in their outsourcing algorithm has to satisfy some special properties, which makes the algorithm complex. Furthermore, their outsource algorithm yields a LLL-reduced basis with a bigger approximation factor than applying LLL-reduction algorithm directly on the original basis, which weakens the requirement of the outsourcing computation task.

Our Contribution. In this paper, we study the algorithm for approximate SVP under the cloud environment, and propose a secure outsourcing algorithm for this problem. In our design, the resource-constrained client can efficiently find a relatively short lattice vector by leveraging the powerful computing capacity of the cloud server.

The idea is quite simple. Roughly speaking, for any given lattice basis $\mathbf{B}$, consider the corresponding Gram matrix $\mathbf{G} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$. Note that for any orthogonal matrix $\mathbf{O}$, the Gram matrix of $\mathbf{OB}$ is exactly $\mathbf{G}$, which means that $\mathbf{G}$ can protect $\mathbf{B}$ well. Moreover, the lattices generated by $\mathbf{OB}$ and $\mathbf{B}$ are different, but for any integer coefficient vector $\mathbf{z}$, the lattice vectors $\mathbf{OBz}$ and $\mathbf{Bz}$ has exactly the same length under the Euclidean norm. This inspires us to send the Gram matrix $\mathbf{G}$ to the cloud server, which can perform LLL algorithm on $\mathbf{G}$ (or some $\mathbf{C}$ such that $\mathbf{G} = \mathbf{C}^{\mathrm{T}}\mathbf{C}$, which can be obtained by Cholesky decomposition on $\mathbf{G}$ by the cloud server). The cloud server will send the transformation matrix to the client and the client can recover the LLL-reduced basis, since the orthogonal transformation does not affect that properties that an LLL-reduced basis should satisfy.

We can show that our design can protect the privacy of client's input/output information under CPA model, and make the client discern the cloud's fraud behavior with optimal probability 1. Hence, our design can be employed in the algorithms that involves with LLL algorithm, such as Coppersmith's attack [7].

Compared to Liu et al.'s work [17], our algorithm also has high-efficiency. Besides, our algorithm is technically concise, which just involves with a simple combination of unimodular matrix transformation and Gram matrix. Furthermore, our algorithm does not reduce the quality of the reduced basis, that is, the vector finally obtained by the client is as short as that of the vector generated by the client directly performing the existing reduction algorithm. Last but not least, it is obvious that our algorithm not only works for the LLL reduction algorithm, but also for any other algorithms that solve (approximate-)SVP with Euclidean norm, since we employ an isometry of the lattice essentially.

Roadmap. The paper is organized as follows: Sect. 2 introduces the system model and security definitions of the outsourcing computation. Section 3 reviews the main computational problems in lattices and presents some necessary preliminaries. In Sect. 4, we propose Gram matrix-based outsourcing algorithm for approximate SVP. In Sect. 5, we analyze the correctness, security and efficiency of our algorithm, followed by extensive experimental analysis to evaluate the practical performance of our design in Sect. 6. In Sect. 7, we give a simple application. Finally, we conclude this article in Sect. 8.

2 System Model and Security Definitions

2.1 System Model

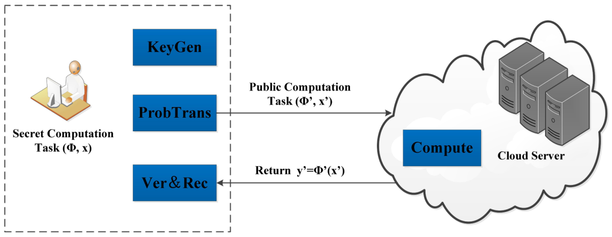

KeyGen
ProbTrans
Ver&Rec
Secret Computation Task (Φ, x)
Public Computation Task (Φ', x')
Return y'=Φ'(x')
Compute
Cloud Server

Fig. 1. The system model

Outsourcing computation is an interaction system between two entities with asymmetric computing capacities: the client and the cloud server, out of which, the client is with limited computational power and storage space, and the cloud server providing computing and storage service is resource-abundant yet maybe untrusted. Formally, as illustrated in Fig. 1, the light-weight client C (or customer, data user, etc.) wants to take advantage of the capacity of the cloud server to accomplish his overloaded calculation task $\Phi(\cdot)$ with an input x.

To protect the input information form being stolen, C transforms the original computation task $\Phi(\cdot)$ into another computation task $\Phi'(\cdot)$, which may be the same as Φ, with corresponding encrypted input x' by the previously generated secret key sk. Then, C sends $(\Phi(\cdot), x')$ to the cloud server S. Next, the cloud server S applies its resources to compute the specified task $y' = \Phi'(x')$ and returns y' to C. After that, C verifies the correctness of y' and recovers y' to the actual result $y = \Phi(x)$ if it passes the verification. Otherwise, C rejects. Precisely, the general framework of secure outsourcing computation can be formalized as a four-tuple OCAlg_{Φ} = (**KeyGen**, **ProbTrans**, **Compute**, **Ver**&**Rec**) with the following four probabilistic polynomial-time (PPT) sub-algorithms [9]:

1. **KeyGen**$(\Phi, x, 1^{\kappa}) \rightarrow \{sk\}$: Input a security parameter κ, the client C conducts the randomized algorithm **KeyGen** to generate a secret key sk for any input computation task (Φ, x).
2. **ProbTrans**$(\Phi, x, sk) \rightarrow \{\Phi', x'\}$: The algorithm **ProbTrans** utilizes the secret key sk to transform (Φ, x) into another computation task (Φ', x'). This algorithm is performed by the client C.
3. **Compute**$(\Phi', x') \rightarrow \{y'\}$: According to the computation task Φ' with the encoded input x', the cloud server S invokes the algorithm **Compute** to compute $y' = \Phi'(x')$ and returns the encoded output y' to the client C.
4. **Ver**&**Rec**$(\Phi, y', sk) \rightarrow \{y\}$: With the secret key sk, the algorithm **Ver**&**Rec** performs as follows. It firstly verifies the correctness of y'. Then, if y' passes the verification, the algorithm recovers y' to $y = \Phi(x)$. Else, the algorithm outputs $y = \perp$.

2.2 Threat Models

In an outsourcing system, the threats are mainly from the untrusted cloud server. The potential misbehaviors of the cloud can be included into two types:

– **Honest-and-curious (*HC*) Server.** In this case, the cloud server will honestly perform the assigned computation task and return the correct result to the client. However, for financial incentive, it may collect or even sell client's valuable information.
– **Fully-malicious (*FM*) Server.** In this case, the cloud server not only tries to capture client's valuable information, but also may deliberately forge a false result to fool the client.

Further, according to the attack abilities owned by the untrusted cloud server, there mainly exist three kinds of attack models:

– **Ciphertext-only attack (*COA*) Model.** In *COA* model, the cloud server is assumed to have only access to the computation tasks Φ, Φ' and the blinded input x', and tries to recover the actual input x and the actual output $y = \Phi(x)$.
– **Known-plaintext attack (*KPA*) Model.** In *KPA* model, the cloud server knows the computation tasks (Φ, Φ') and has the ability of collecting a polynomial number of plaintexts x_i $(1 \leq i \leq \ell)$ and the corresponding ciphertexts

x'_i, y'_i and δ_i. Given challenge ciphertexts x' and $y' = \Phi'(x')$, the cloud server tries to recover the actual input x and the actual output $y = \Phi(x)$.

– **Chosen-plaintext attack (*CPA*) Model.** In *CPA* model, the cloud server knows the computation tasks (Φ, Φ') and is allowed to adaptively choose a polynomial number of inputs x_i $(1 \leq i \leq N)$ and obtain the corresponding ciphertext x'_i, y'_i and δ_i. Given challenge ciphertexts x' and $y' = \Phi'(x')$, the cloud server tries to recover the actual input x and the actual output $y = \Phi(x)$.

Overall, there exist 6 possible threat models: *HC+COA*, *HC+KPA*, *HC+CPA*, *FM+COA*, *FM+KPA*, *FM+CPA*. Clearly, in the above-mentioned threat models, the ability of the *FM+CPA* model is the most powerful. Therefore, an outsourcing algorithm that is secure under the *FM+CPA* model is certainly secure under the other models, and thus, it is more meaningful to design a secure outsourcing algorithm under the *FM+CPA* model.

2.3 Correctness and Security Definitions

Based on the above-mentioned system architecture and threat models, a secure outsourcing algorithm should at least meet four requirements: correctness, input/output privacy, verifiability and High efficiency. We now present their strictly formalized definitions.

Correctness is a basic requirements for an outsourcing algorithm. It means that, if the cloud server performed the specified computations in the algorithm honestly, the client can correctly achieve the result of the original computation task. Precisely,

Definition 1 (Correctness). *A secure outsourcing algorithm* $\mathrm{OCAlg}_{\Phi}(\cdot)$ *of some computation task* Φ *is correct if the key generation algorithm generates key* $\{sk\} \leftarrow \mathbf{KeyGen}(\Phi, x, 1^\kappa)$ *such that, for any valid input* x *of* Φ*, if* $\{\Phi', x'\} \leftarrow \mathbf{ProbTrans}(\Phi, x, sk)$ *and* $y' \leftarrow \mathbf{Compute}(\Phi', x')$*, where* $y' = \Phi'(x')$*, the algorithm* $\mathbf{Ver\&Rec}(\Phi, y', sk)$ *outputs* $y = \Phi(x)$*.*

Input/output privacy is a security requirement for an outsourcing algorithm. It asks the outsourcing algorithm to protect the input and output information of client's computation task from being disclosed to the cloud server. Here, we mainly discuss the input (resp. output) privacy with one-way notion under the *FM+CPA* model. To give the strict privacy definition, we first formalize the description of *CPA* model with the following experiments $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[\Phi, 1^\kappa]$ and $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[\Phi, 1^\kappa]$.

```
Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[\Phi, 1^\kappa]$
 *Query and response*:
 $x_0 = \sigma_{x_0} = \perp$.
 For $i = 1, \cdots, \ell = \mathrm{poly}(\kappa)$
   $x_i \leftarrow \mathcal{A}(\Phi, (x_j, \sigma_{x_j})_{0 \leq j \leq i-1})$.
   $sk_i \leftarrow \mathbf{KeyGen}(\Phi, x_i, 1^\kappa)$.
   $\sigma_{x_i} = (\Phi', x'_i) \leftarrow \mathbf{ProbTrans}(\Phi, sk_i, x_i)$.
```

```
  Challenge:
    $\tilde{x} \leftarrow \mathrm{Domain}(\Phi)$.
    $\tilde{sk} \leftarrow \mathbf{KeyGen}(\Phi, \tilde{x}, 1^{\kappa})$.
    $\sigma_{\tilde{x}} = (\Phi', \tilde{x}') \leftarrow \mathbf{ProbTrans}(\Phi, \tilde{sk}, \tilde{x})$.
    $\bar{x} \leftarrow \mathcal{A}(\Phi, (x_j, \sigma_{x_j})_{0 \le j \le \ell}, \sigma_{\tilde{x}})$.
    if $\bar{x} = \tilde{x}$ , output $'1'$;
    else output $'0'$.
Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[\Phi, 1^{\kappa}]$
  Query and response:
  $x_0 = \sigma_{x_0} = y_0 = \perp$.
  For $i = 1, \cdots, \ell = \mathrm{poly}(\kappa)$
    $x_i \leftarrow \mathcal{A}(\Phi, (x_j, \sigma_{x_j}, y_j)_{0 \le j \le i-1})$.
    $sk_i \leftarrow \mathbf{KeyGen}(\Phi, x_i, 1^{\kappa})$.
    $\sigma_{x_i} = (\Phi', x') \leftarrow \mathbf{ProbTrans}(\Phi, sk_i, x_i)$.
    $y_i' \leftarrow \mathcal{A}(\Phi, (x_j, \sigma_{x_j}, y_j)_{0 \le j \le i-1}, \sigma_{x_i})$.
    $y_i \leftarrow \mathbf{Ver\&Rec}(\Phi, sk_i, y_i')$.
  Challenge:
    $\tilde{x} \leftarrow \mathrm{Domain}(\Phi)$.
    $\tilde{sk} \leftarrow \mathbf{KeyGen}(\Phi, \tilde{x}, 1^{\kappa})$.
    $\sigma_{\tilde{x}} = (\Phi', \tilde{x}') \leftarrow \mathbf{ProbTrans}(\Phi, \tilde{sk}, \tilde{x})$.
    $\tilde{y}' \leftarrow \mathbf{Compute}(\sigma_{\tilde{x}})$.
    $\tilde{y} \leftarrow \mathcal{A}(\Phi, (x_j, \sigma_{x_j}, y_j)_{0 \le j \le \ell}, \sigma_{\tilde{x}}, \tilde{y}')$.
    if $\tilde{y} = \Phi(\tilde{x})$, output $'1'$;
    else output $'0'$.
```

Now, the input/output privacy can be exactly defined.

Definition 2 (Input/output privacy). *A secure outsourcing algorithm* $\mathrm{OCAlg}_{\Phi}(\cdot)$ *of some computation task* Φ *is input-private (resp. output-private) if, for any PPT adversary* $\mathcal{A}$*, the probability of the experiment* $\boldsymbol{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[\Phi, 1^{\kappa}]$ *(resp.* $\boldsymbol{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[\Phi, 1^{\kappa}]$*) outputting* 1 *is negligible,* i.e.

$$\Pr[\boldsymbol{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[\Phi, 1^{\kappa}] = 1] \le \mathrm{negli}(\kappa) \;\; (resp. \Pr[\boldsymbol{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[\Phi, 1^{\kappa}] = 1] \le \mathrm{negli}(\kappa)),$$

where $\mathrm{negli}(\kappa)$ *is a negligible function of the security parameter* κ.

Verifiability is another security requirement for an outsourcing algorithm. That is, the algorithm should guarantee that the client can not be cheated by an untrusted cloud server. Conversely, the client can verify the correctness of the results returned from the cloud with a non-negligible probability.

Definition 3 ((1 − β)-Verifiable). *A secure outsourcing algorithm* $\mathrm{OCAlg}_{\Phi}(\cdot)$ *of some computation task* Φ *is* $(1-\beta)$*-verifiable if, for any valid input* x*, the algorithm* **KeyGen** *outputs a secret key* sk *such that, if* $(\Phi', x') \leftarrow$ $\mathbf{ProbTrans}(\Phi, x, sk)$ *and* $y' \leftarrow \mathbf{Compute}(\Phi', x')$*, the probability of* **Ver&Rec** (Φ, y', sk) *outputting* y *satisfies*

$$\Pr[y = \Phi(x) \leftarrow \mathbf{Ver\&Rec}(\Phi, y', sk) \mid y' = \Phi'(x')] = 1,$$
$$\Pr[y = \Phi(x) \leftarrow \mathbf{Ver\&Rec}(\Phi, y', sk) \mid y' \ne \Phi'(x')] \le \beta.$$

High efficiency is a necessary requirement for an outsourcing algorithm which refers to that the client's calculation amount in the outsourcing algorithm must be lower than the original computation task performed by the client itself.

Definition 4 (α-Efficient). *For some computation task Φ, a secure outsourcing algorithm $\mathrm{OCAlg}_{\Phi}(\cdot)$ is α-efficient if $\frac{t_c}{t_o} \le \alpha$, out of which, t_o is the client's time cost of achieving the task without outsourcing, and t_c represents the local-client's time cost of achieving the task by employing the outsourcing algorithm $\mathrm{OCAlg}_{\Phi}(\cdot)$.*

3 Notations and Preliminaries

In the rest of our paper, we use bold upper-case letter to denote matrix and use bold lower-case letter to denote vector. The frequently used notations are described in Table 1. Next, we introduce some basic knowledge about lattices, and then review the famous LLL reduction algorithm for solving approximate SVP.

Table 1. Notations

Symbols	Descriptions
m, n	Positive integers
$\mathbb{R}^m$	m-dimensional Euclidean space
$\mathbf{v}$	A column vector in $\mathbb{Z}^m$
Λ	A lattice in $\mathbb{R}^m$ with rank n
$\mathbf{B}$	A basis matrix of the lattice
$\mathbf{U}$	An unimodular matrix in $\mathbb{Z}^{n\times n}$
$\lambda_i(\Lambda)$	The ith successive minimum of the lattice Λ
κ	Security parameter

3.1 Lattice

A lattice Λ is a discrete subgroup of $\mathbb{R}^m$, or equivalently,

Definition 5 (Lattice). *Given $n(\le m)$ linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \in \mathbb{R}^m$, the lattice Λ generated by them is the set of all integral linear combinations of $\mathbf{b}_i$, i.e.,*

$$\Lambda(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i | x_i \in \mathbb{Z} \right\},$$

where the matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is called a basis of the lattice Λ and n is the rank of the lattice. When $n = m$, the lattice is full-rank.

The lattice generated by the basis $\mathbf{B}$ is also denoted as

$$\Lambda(\mathbf{B}) = \Lambda(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^n\}.$$

Since there are many bases for a lattice, a natural question is how to determine whether two bases $\mathbf{B}_1, \mathbf{B}_2$ generate the same lattice (*i.e.*, $\Lambda(\mathbf{B}_1) = \Lambda(\mathbf{B}_2)$. To illustrate this, we need to introduce the definition of unimodular matrix.

Definition 6 (Unimodular Matrix). *A matirx $\mathbf{U} \in \mathbb{Z}^{n\times n}$ is called unimodular if* $\det \mathbf{U} = \pm 1$.

Then, we have

Lemma 1. *Two bases $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{R}^{m\times n}$ are equivalent, i.e., $\Lambda(\mathbf{B}_1) = \Lambda(\mathbf{B}_2)$ if and only if $\mathbf{B}_2 = \mathbf{B}_1\mathbf{U}$, for some unimodular matrix $\mathbf{U}$.*

Definition 7 (Determinant). *Let $\Lambda = \Lambda(\mathbf{B})$ be a lattice of rank n. The determinant of Λ is defined as $vol(\Lambda) = \sqrt{\det(\mathbf{B}^{\mathrm{T}}\mathbf{B})}$, where $\mathbf{B}^{\mathrm{T}}$ denotes the transpose of the basis $\mathbf{B}$.*

Besides, the length of the shortest nonzero vector in the lattice Λ is denoted as $\lambda_1(\Lambda)$, where the length refers to the Euclidean norm. That is, for any vector $\mathbf{x}$, $\|\mathbf{x}\|_2 = \sqrt{\sum x_i^2}$. The most important computational problem SVP is defined as follows:

Definition 8 (SVP). *Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m\times n}$, find a nonzero vector $\mathbf{x} \in \Lambda(\mathbf{B})$ such that $\|\mathbf{x}\| = \lambda_1(\Lambda(\mathbf{B}))$.*

So far, there is no known efficient algorithm to solve this problem. However, we are more interested in the approximation variant, which aims to find a relatively shorter lattice vector with length no bigger than $\gamma(n)\lambda_1(\Lambda)$. Here, n is the dimension of the lattice and the approximation factor $\gamma = \gamma(n) \geq 1$. Formally,

Definition 9 (SVP$_\gamma$). *Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m\times n}$, find a nonzero vector $\mathbf{x} \in \Lambda(\mathbf{B})$ such that $\|\mathbf{x}\| \leq \gamma \cdot \lambda_1(\Lambda(B))$.*

3.2 LLL Reduction Algorithm and Its Properties

This section describes the famous LLL reduction algorithm of solving SVP_γ. First, we recall what is an LLL-reduced basis.

Definition 10 [15]**.** *A basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is a δ-LLL reduced basis if the following two inequalities hold:*

1. $\forall 1 \leq i \leq n, j < i, |\mu_{i,j}| \leq \frac{1}{2}$.
2. $\forall 1 \leq i < n, \delta\|\tilde{\mathbf{b}}_i\|^2 \leq \|\mu_{i+1,i}\tilde{\mathbf{b}}_i + \tilde{\mathbf{b}}_{i+1}\|^2$.

Some useful properties are given as follows:

Proposition 1 [15]. *Let* $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ *be an LLL reduced basis of a lattice* Λ*, then*

1. $vol(\Lambda) \leq \prod_{i=1}^{n} \|\mathbf{b}_i\| \leq 2^{\frac{n(n-1)}{4}} vol(\Lambda)$.
2. $\|\mathbf{b}_1\| \leq 2^{\frac{n-1}{4}} (vol(\Lambda))^{\frac{1}{n}}$.
3. $\forall 1 \leq i \leq n, \|\mathbf{b}_i\| \leq 2^{\frac{n-1}{2}} \lambda_i(\Lambda)$.

Lenstra et al. [15] presented an efficient algorithm (*i.e.*, the celebrated LLL algorithm) to output a LLL reduced basis. Precisely,

Proposition 2 [15]. *Let* $\Lambda = \Lambda(\mathbf{B})$ *be a rank-n lattice with* $\mathbf{B} \in \mathbb{Q}^{m \times n}$. *Then LLL reduction algorithm can find an LLL-reduce basis within time* $O(n^5 m B^3)$ *without fast multiplication techniques, where* $B = \max_{1 \leq i \leq n} \log \|\mathbf{b}_i\|$.

For a lattice basis $\mathbf{B}$, the corresponding Gram matrix is defined as $\mathbf{G} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$. Gram matrix has been used in LLL algorithm [2,18,19]. In practice, we can directly perform LLL reduction on it to get the transformation matrix $\mathbf{T}$ such that $\mathbf{BT}$ is LLL-reduced. Here, we summarize the result as the following Lemma 2.

Lemma 2. *Let* $\Lambda = \Lambda(\mathbf{B})$ *be a lattice of rank n. There exists a variant of LLL algorithm, with the input of the Gram matrix* $\mathbf{G} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$*, outputting a transformation matrix* $\mathbf{T}$ *such that* $\mathbf{BT}$ *is LLL-reduced.*

This algorithm has already been implemented in SageMath [23].

4 Our Outsourcing Algorithm for SVP$_\gamma$

In this section, we first overview the design rationale, and then present our algorithm in detail.

4.1 Design Rationale

Given some lattice basis $\mathbf{B} \in \mathbb{R}^{m \times n}$, a resource-limited client aims to leverage the computing resource of the cloud to find a non-zero shorter vector $\mathbf{x} \in \Lambda(\mathbf{B})$ such that $\|\mathbf{x}\| \leq 2^{\frac{n-1}{4}} \cdot (vol(\Lambda))^{\frac{1}{n}}$, where n is the dimension of the lattice. If the cloud server is honest, the client can directly rent the cloud server to perform the famous LLL algorithm with input $\mathbf{B}$. However, under the *FM+CPA* model, the client must figure out an effective method to protect the privacy of the input lattice basis $\mathbf{B}$ and the output lattice vector $\mathbf{x}$. By Lemma 1, a natural idea is to blind $\mathbf{B}$ with a random unimodular matrix $\mathbf{U}$. Namely, compute $\mathbf{B}' = \mathbf{BU}$ and let the cloud perform LLL algorithm on $\mathbf{B}'$. However, since $\Lambda(\mathbf{B}) = \Lambda(\mathbf{B}')$, this simple method can not ensure the privacy of the output vector $\mathbf{x}$. Enlightened by the property of the variant of LLL in Lemma 2, to protect the output information, we can send the Gram matrix $\mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}'$ to the cloud and rent the cloud to perform the variant algorithm on $\mathbf{G}$ to get the transformation matrix $\mathbf{T}$ such that $\mathbf{B}'\mathbf{T}$ is LLL-reduced. After that, the cloud returns the first column vector $\mathbf{z}$ of $\mathbf{T}$. Finally, the client can obtain a shorter lattice vector by computing $\mathbf{x} = \mathbf{B}'\mathbf{z}$. Since $\mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}' = (\mathbf{OB}')^{\mathrm{T}}(\mathbf{OB}')$ for any orthogonal matrix $\mathbf{O}$, even if the cloud performs Cholesky decomposition on $\mathbf{G}$ and obtain $\mathbf{G} = \mathbf{C}^{\mathrm{T}}\mathbf{C}$, it can not distinguish $\mathbf{C} = \mathbf{B}'$ from $\mathbf{C} = \mathbf{OB}'$ for any orthogonal matrix $\mathbf{O}$. This ensures the input privacy. Meanwhile, the cloud only obtains vector $\mathbf{z}$, without knowing the lattice basis $\mathbf{B}'$, it can not recover the shorter lattice vector $\mathbf{x}$. This ensures the output privacy. Besides, the verifiability can be realized by checking $\|\mathbf{x}\| \leq 2^{\frac{n-1}{4}} \cdot (\sqrt{\det(\mathbf{G})})^{\frac{1}{n}}$.

4.2 Detailed Algorithm

Note that we almost focus on the short vector in practice, such as in the lattice-based cryptanalysis, instead of the whole LLL-reduced basis, so below we don't take consideration into getting the whole LLL-reduced basis, but just aims to find a short lattice vector, and we would like to point out that it is very easy to extend the algorithm below to compute the whole LLL-reduced basis as discussed in Sect. 4.3.

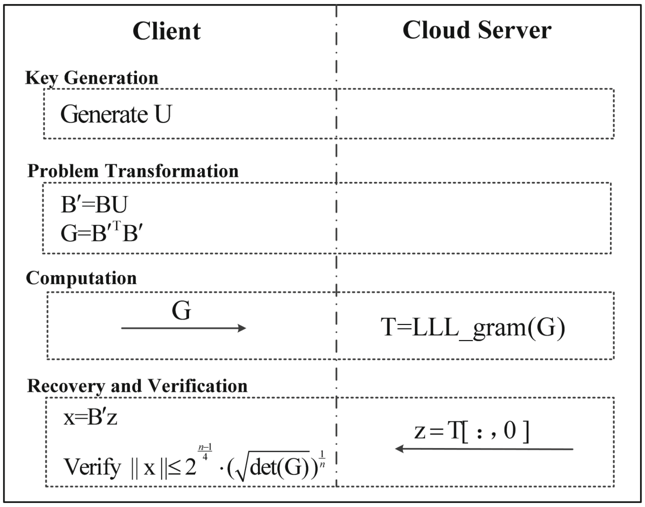


Fig. 2. The workflow of algorithm OCAlg_{SVP}.

Figure 2 shows the workflow of our outsourcing algorithm OCAlg_{SVP}, which is designed as follows:

1. **Key Generation**: With an inputted basis matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$ of lattice Λ, the client picks a secret unimodular matrix $\mathbf{U} \in \mathbb{Z}^{n\times n}$ in a secret finite set consisting of unimodular matrices produced in advance.

2. **Problem Transformation**: With the secret key $\mathbf{U}$, the client first computes $\mathbf{B}' = \mathbf{B}\mathbf{U}$ and further calculates the Gram-matrix $\mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}'$. Finally, $\mathbf{G}$ is sent to the cloud server.
3. **Computation**: On receiving the matrix $\mathbf{G}$, the cloud server applies the variant algorithm of LLL to $\mathbf{G}$ and obtains a transformation matrix $\mathbf{T}$. Then, the cloud returns the first column vector $\mathbf{z} \in \mathbb{Z}^n$ of the matrix $\mathbf{T}$ to the client.
4. **Recovery and Verification**: After receiving the vector $\mathbf{z} \in \mathbb{Z}^n$, the client computes $\mathbf{x} = \mathbf{B}'\mathbf{z}$ and checks whether the inequality $\|\mathbf{x}\| \leq 2^{\frac{n-1}{4}} \cdot (\sqrt{\det(\mathbf{G})})^{\frac{1}{n}}$ holds. If it holds, the vector $\mathbf{x}$ is the desired result. Otherwise, the client rejects the result.

4.3 Some Remarks

Note that the secret unimodular transformation $\mathbf{U}$ is not necessary if no additional information about the private input $\mathbf{B}$ leaks to the adversary. However, we may know some additional information about $\mathbf{B}$ in some cases, such as for Coppersmith's algorithm [7], $\mathbf{B}$ must be an upper (or lower) triangular matrix, which means that the domain of the outsourcing task is not the whole $\mathbb{R}^{m\times n}$ any more and some secret information about $\mathbf{B}$ can be implied from the Gram matrix. Hence, we employ a unimodular transformation to keep the privacy of $\mathbf{B}$ further in our algorithm and would like to point out that it can be removed in some cases which depends on the domain of the task.

It is obvious that if the cloud server sends the transformation $\mathbf{T}$ to the client, the client can recover the LLL-reduced basis $\mathbf{B}'\mathbf{T}$ for the original lattice due to Lemma 2, which shows that the outsoucing algorithm can be extended to compute the LLL-reduced basis easily.

Compared to Liu et al.'s work [17], since we employ an isometry of the lattice essentially, the orthogonal transformation, our algorithm does not reduce the quality of the reduced basis, and our algorithm can work for any other algorithms that solve (approximate-)SVP with Euclidean norm (sometimes, a matrix $\mathbf{C}$ such that $\mathbf{G} = \mathbf{C}^{\mathrm{T}}\mathbf{C}$ should be computed by the cloud server with Cholesky decomposition on $\mathbf{G}$).

Another way to construct the outsourcing algorithm for the client is to generate the orthogonal transformation $\mathbf{O}$ directly and to send $\mathbf{O}\mathbf{B}$ to the cloud server. There are many efficient ways to construct $\mathbf{O}$, such as the Householder transformation. In this paper, we adopt the Gram matrix to avoid generating the orthogonal matrix.

5 Correctness, Security, Verifiability and Efficiency

In this section, we will present strict analysis on the correctness, input/output privacy, verifiability and efficiency of our proposed algorithm.

5.1 Correctness

Clearly, if the cloud server performs the assigned computation task honestly, the client definitely obtains a shorter vector. In fact, by Lemma 1, $\mathbf{B}$ and $\mathbf{B}' = \mathbf{BU}$ are equivalent, *i.e.*, $\Lambda = \Lambda(\mathbf{B}) = \Lambda(\mathbf{B}')$. By Lemma 2, if the cloud is honest, we have $\mathbf{B}'\mathbf{T}$ is LLL-reduced. Then $\mathbf{x} = \mathbf{B}'\mathbf{z} \in \Lambda$ is the first vector of the LLL-reduced basis $\mathbf{B}'\mathbf{T}$. According to the properties (Proposition 1) of LLL-reduced basis, $\|\mathbf{x}\| \leq 2^{\frac{n-1}{4}} \cdot (\sqrt{\det(\mathbf{G})})^{\frac{1}{n}}$.

5.2 Input/Output Privacy

Here, we mainly argue the privacy of our algorithm with the one-way notation under *CPA* model.

Theorem 1. *For any input lattice basis* $\mathbf{B}$*, our proposed outsourcing algorithm* OCAlg_{SVP} *satisfies the input/output privacy according to Definition 2.*

Proof. We first argue the input privacy. Corresponding to our design, in the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[\Phi, 1^\kappa]$, the computation task Φ refers to finding a non-zero shorter vector $\mathbf{x}$ of some lattice Λ such that $\|\mathbf{x}\| \leq 2^{\frac{n-1}{4}} \cdot (vol(\Lambda))^{\frac{1}{n}}$, Φ' represents performing the variant algorithm of LLL on the Gram-matrix $\mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}'$, and $\kappa = mn \log\|\mathbf{B}\| = mn \log \max_{1\leq i\leq m, 1\leq j\leq n} |b_{ij}|$ denotes the bit-size of the input lattice basis $\mathbf{B}$. The adversary $\mathcal{A}$ can adaptively choose $x_i = \mathbf{B}_i$ and obtain corresponding Gram matrices $\mathbf{G}_i = (\mathbf{B}_i')^{\mathrm{T}}\mathbf{B}_i'$ for $1 \leq i \leq \ell$ in the *Query and response* phase. In the *Challenge* phase, given a challenge input lattice basis $\tilde{x} = \mathbf{B}$, the adversary tries to recover $\mathbf{B}$ based on the collected information $\mathbf{G}_i = (\mathbf{B}_i')^{\mathrm{T}}\mathbf{B}_i'(1 \leq i \leq \ell)$ and $\mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}'$. Since $\mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}' = (\mathbf{OB}')^{\mathrm{T}}(\mathbf{OB}')$ for any orthogonal matrix $\mathbf{O}$ and there exist at least exponentially many orthogonal matrices, the probability of the adversary recovering the correct $\mathbf{B}'$ is negligible. Due to the fact that $\mathbf{B}' = \mathbf{BU}$ for some unimodular matrix $\mathbf{U}$ produced in advance, the probability of the adversary recovering the correct $\mathbf{B}$ is clearly negligible, *i.e.*,

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[\Phi, 1^\kappa] = 1] \leq \mathrm{negli}(\kappa).$$

Now, we argue the output privacy. Similar to the above analysis, the adversary can adaptively choose and obtain $(x_i, \sigma_{x_i}, y_i) = (\mathbf{B}_i, \mathbf{G}_i, \mathbf{x}_i)$ (or $(\mathbf{B}_i, \mathbf{G}_i, \perp)$) for $1 \leq i \leq \ell$ in the *Query and response* phase. In the *Challenge* phase, given a challenge input lattice basis $\tilde{x} = \mathbf{B}$ of the lattice Λ, the adversary tries to recover a shorter vector $\tilde{y} = \mathbf{x} \in \Lambda(\mathbf{B})$ such that $\|\mathbf{x}\| \leq 2^{\frac{n-1}{4}} vol(\Lambda)^{\frac{1}{n}}$. Besides the collected information in the *Query and response* phase, the adversary also captures the Gram matrix $\tilde{x}' = \mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}' = (\mathbf{BU})^{\mathrm{T}}(\mathbf{BU})$ for some secret and random matrix $\mathbf{U}$, and an integer vector $\tilde{y}' = \mathbf{z}$, which is the first column of the transformation matrix $\mathbf{T}$. Since $\mathbf{G} = (\mathbf{B}')^{\mathrm{T}}\mathbf{B}' = (\mathbf{OB}')^{\mathrm{T}}(\mathbf{OB}')$ for any orthogonal matrix $\mathbf{O}$ and there exist infinitely many orthogonal matrices, the probability of the adversary recovering the correct $\mathbf{B}'$ is negligible. Due to

$\mathbf{x}' = \mathbf{B}'\mathbf{z}$, we have that, without knowing the correct $\mathbf{B}'$, the probability of the adversary recovering the correct $\mathbf{x}$ is clearly negligible, *i.e.*,

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{Opriv}[\Phi, 1^\kappa] = 1] \leq \text{negli}(\kappa).$$

5.3 Verifiability

Theorem 2. *Our proposed outsourcing algorithms* OCAlg_{SVP} *is 100%-verifiable.*

Proof. Since $\Lambda(B) = \Lambda(\mathbf{B}')$, we have $\mathbf{x} = \mathbf{B}'\mathbf{z} \in \Lambda(\mathbf{B})$. According to Lemma 2 and the Proposition 1, $\mathbf{B}'\mathbf{T}$ is LLL-reduced and thus $\|\mathbf{x}\| = \|\mathbf{B}'\mathbf{z}\| \leq 2^{\frac{n-1}{4}} \cdot (\sqrt{\det(G)})^{\frac{1}{n}}$, Hence, $\Pr[y = \Phi(x) \leftarrow \mathbf{Ver\&Rec}(\Phi, y', sk) \mid y' = \Phi'(x')] = 1$. If the cloud server returns a forged integer vector $\mathbf{z}$, then the inequality $\|\mathbf{x}\| = \|\mathbf{B}'\mathbf{z}\| \leq 2^{\frac{n-1}{4}} \cdot (\sqrt{\det(G)})^{\frac{1}{n}}$ doesn't hold.

5.4 Efficiency

Theorem 3. *Our outsourcing algorithm* OCAlg_{SVP} *is at least* $O(\frac{1}{n^2 \log B})$*-efficient.*

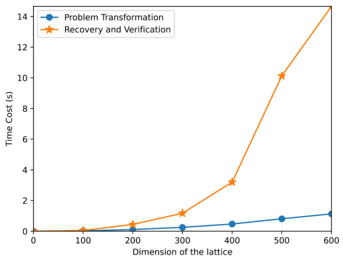
Proof. The original algorithm used to approximate the shortest nonzero lattice vector is the famous LLL-reduction algorithm with an asymptotic complexity of $O(n^5 m \log^3 B)$ [15], where $B = \max\limits_{1 \leq i \leq n} \log \|\mathbf{b}_i\|$. While the principal term of the computational complexity on the client side in our proposed algorithm is the time cost of matrix multiplication and determinant computation which takes time at most $O(n^3 m \log^2 B)$. Therefore, according to Definition 4, our algorithm is $O(\frac{n^3 m \log^2 B}{n^5 m \log^3 B}) = O(\frac{1}{n^2 \log B})$-efficient.

6 Practical Performance Evaluation

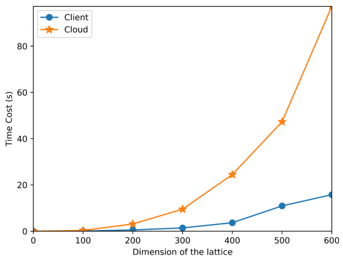
6.1 Evaluation Methodology

After analyzing the correctness, verifiability, efficiency and security of our proposed algorithm, we conclude that our proposed algorithm are beneficial to the client. We next evaluate the practical performance of our proposed algorithm by simulating both client and cloud on a Windows 10 machine with Intel(R) Core(TM) i7-7500U 2.70 GHz CPU and 12 GB RAM. We implemented our proposed algorithm in a free software SageMath [23] in which the function LLL_gram() adapted from Nguyen and Stehlé's algorithm [18] returns the LLL transformation matrix for this Gram matrix. Because this LLL reduction algorithm has default parameters $(0.51, 0.99)$ meaning that the Gram-Schmidt coefficients for the reduced basis satisfy $|\mu_{i,j}| \geq 0.51$, and the Lovász's constant is 0.99, thus we modified parameters of the function LLL() as $(0.51, 0.99)$ for better evaluation.

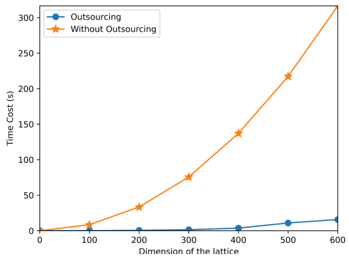
In each of the experimental instances, we first constructed basis matrix inputted by the client, whose entries were randomly chosen from interval $[-2^{20}, 2^{20}]$. For the sake of convenience, basis matrices were chosen in $\mathbb{Z}^{n\times n}$ and their dimensions varied from 100 to 600. Then we produced a sufficiently large and finite set consisting of unimodular matrices whose entries ranging from -100 to 100. Besides, we simulated all stages of our proposed algorithm in each of experiments to evaluate the performance more objectively. The client-side time "Client", which is also the time "Outsourcing", denotes the sum of time "Problem Transformation" and "Recovery and Verification". The time "Without Outsourcing" denotes the time for the client to solve the approximation of SVP without outsourcing. Theoretically, the value (Outsourcing/Without Outsourcing) is a positive number less than 1. The time "Cloud" refers to the time for the cloud server to compute the outsourced task.



(a) Time comparison of different phases



(b) Time comparison between cloud server and client



(c) Time comparison of outsourcing and without outsourcing

Fig. 3. Evaluation results for algorithm

6.2 Evaluation Results

Figure 3 shows the evaluation results of our proposed algorithm. In Fig. 3(a), we make time comparison among phases. From it, we can see that the time of "Problem Transformation" and "Recovery and Verification" increases with the dimension of the lattice and it is sound. Moreover, it is obvious that "Recovery and Verification" takes more time than "Problem Transformation". By rigorous analysis, we can come to the same conclusion. In the Problem Transformation phase, the main time-consuming operation is matrix multiplication, while there are operations for determinant and exponentiation besides matrix multiplication in the Recovery and Verification phase. Then the second result is also rational. After outsourcing process, the time taken by the client decreases dramatically in Fig. 3(b), where we make time comparison between cloud server and client. It means that our proposed algorithm works well. In Fig. 3(c), we show a visual efficiency comparison and compare it with theoretical analysis above. Clearly, the client takes less time in outsourcing process than implementing original algorithm itself, i.e., our proposed outsourcing algorithm is very efficient.

7 Applications

Next, we take the famous Copersmith's algorithm [7] as an example to show how to use our outsourcing algorithm.

Copersmith's algorithm, described in Algorithm 1, finds all small roots x_0 of a univariate modular equation $f(x) = 0 \bmod N$ with $|x_0| \leq N^{\frac{1}{\delta}}$, where the polynomial $f(x)$ is a monic polynomial of degree δ. It has many applications in the cryptanalysis. It can be seen that the most time-consuming step is Step 5, running the LLL algorithm to get a short lattice vector. Hence, our outsourcing algorithm can be used directly to obtain a short lattice vector while keeping the privacy of input and output.

Algorithm 1. Coppersmith's Algorithm

```
Require: A monic polynomial $f(x) \in \mathbb{Z}_N[x]$ of degree $\delta$, where $N$ is a modulus with
    unknown factorization.
Ensure: Set $R = \{x_0 \in \mathbb{Z} | f(x_0) = 0 \bmod N$ and $|x_0| \leq X\}$.
1: for $i \leftarrow 0$ to $h-1$ do
2:     for $j \leftarrow 0$ to $\delta - 1$ do
3:         $g_{i,j}(x) = N^{h-i-1} f(x)^i x^j$.
4: Construct the lattice basis $B$, where the basis vectors are the coefficient vectors of
    $g_{i,j}(xX)$.
5: $v = LLL(B).get\_column(0)$.
6: Construct $g(x)$ from $v$.
7: Find the set $R$ of all roots of $g(x)$ over the integers using standart techniques. For
    every root $x_0 \in R$ check wether or not $\gcd(N, f(x_0)) \geq N$. If it doesn't hold then
    remove $x_0$ from $R$.
```

8 Conclusion

In this paper, we present an efficient and secure outsourcing algorithm solving (approximate-)SVP for the client with limited computing and storage capability, which has many applications in computational number theory, cryptanalysis and some other related areas.

Acknowledgements. We thank the anonymous referees for their valuable suggestions on how to improve this paper.

References

1. Ajtai, M.: The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In: Vitter, J.S. (ed.) Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, 23–26 May 1998, pp. 10–19. ACM (1998). https://doi.org/10.1145/276698.276705
2. Backes, W., Wetzel, S.: An efficient LLL gram using buffered transformations. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2007. LNCS, vol. 4770, pp. 31–44. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75187-8_4
3. Benjamin, D., Atallah, M.J.: Private and cheating-free outsourcing of algebraic computations. In: Korba, L., Marsh, S., Safavi-Naini, R. (eds.) Sixth Annual Conference on Privacy, Security and Trust, PST 2008, Fredericton, New Brunswick, Canada, 1–3 October 2008, pp. 240–245. IEEE Computer Society (2008). https://doi.org/10.1109/PST.2008.12
4. Bi, J., Coron, J., Faugère, J., Nguyen, P.Q., Renault, G., Zeitoun, R.: Rounding and chaining LLL: finding faster small roots of univariate polynomial congruences. IACR Cryptol. ePrint Arch. **2014**, 437 (2014). http://eprint.iacr.org/2014/437
5. Chen, X., Li, J., Ma, J., Tang, Q., Lou, W.: New algorithms for secure outsourcing of modular exponentiations. IEEE Trans. Parallel Distributed Syst. **25**(9), 2386–2396 (2014). https://doi.org/10.1109/TPDS.2013.180
6. Cohen, H.: A Course in Computational Algebraic Number Theory, Graduate Texts in Mathematics, vol. 138. Springer, Heidelberg (1993). https://www.worldcat.org/oclc/27810276
7. Coppersmith, D.: Finding a small root of a univariate modular equation. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 155–165. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_14
8. Fiore, D., Gennaro, R.: Publicly verifiable delegation of large polynomials and matrix computations, with applications. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) The ACM Conference on Computer and Communications Security, CCS 2012, Raleigh, NC, USA, 16–18 October 2012, pp. 501–512. ACM (2012). https://doi.org/10.1145/2382196.2382250
9. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_25
10. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, 17–20 May 2008, pp. 197–206. ACM (2008). https://doi.org/10.1145/1374376.1374407
11. Håstad, J., Just, B., Lagarias, J.C., Schnorr, C.: Polynomial time algorithms for finding integer relations among real numbers. SIAM J. Comput. **18**(5), 859–881 (1989). https://doi.org/10.1137/0218059

12. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_15
13. Hu, C., Alhothaily, A., Alrawais, A., Cheng, X., Sturtivant, C., Liu, H.: A secure and verifiable outsourcing scheme for matrix inverse computation. In: 2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, 1–4 May 2017, pp. 1–9. IEEE (2017). https://doi.org/10.1109/INFOCOM.2017.8057199
14. Lagrange, J.L.: Recherches d'arithmétique. Proc. Nouv. Mém. Acad. (1773)
15. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**(4), 515–534 (1982)
16. Liu, D., Bertino, E., Yi, X.: Privacy of outsourced k-means clustering. In: Moriai, S., Jaeger, T., Sakurai, K. (eds.) 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS 2014, Kyoto, Japan, 03–06 June 2014, pp. 123–134. ACM (2014). https://doi.org/10.1145/2590296.2590332
17. Liu, J., Bi, J.: Secure outsourcing of lattice basis reduction. In: Gedeon, T., Wong, K.W., Lee, M. (eds.) ICONIP 2019, Part II. LNCS, vol. 11954, pp. 603–615. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36711-4_51
18. Nguên, P.Q., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_13
19. Nguyen, P.Q., Stehlé, D.: LLL on the average. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 238–256. Springer, Heidelberg (2006). https://doi.org/10.1007/11792086_18
20. Saruchi, Morel, I., Stehlé, D., Villard, G.: LLL reducing with the most significant bits. In: Nabeshima, K., Nagasaka, K., Winkler, F., Szántó, Á. (eds.) International Symposium on Symbolic and Algebraic Computation, ISSAC 20, Kobe, Japan, 23–25 July 2014, pp. 367–374. ACM (2014). https://doi.org/10.1145/2608628.2608645
21. Schnorr, C.: A hierarchy of polynomial time lattice basis reduction algorithms. Theor. Comput. Sci. **53**, 201–224 (1987). https://doi.org/10.1016/0304-3975(87)90064-8
22. Schnorr, C.P.: Factoring integers and computing discrete logarithms via diophantine approximation. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 281–293. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_24
23. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 9.2) (2020). https://www.sagemath.org
24. Tian, C., Yu, J., Zhang, H., Xue, H., Wang, C., Ren, K.: Novel secure outsourcing of modular inversion for arbitrary and variable modulus. IEEE Trans. Serv. Comput., 1 (2019). https://doi.org/10.1109/TSC.2019.2937486
25. Yang, Y., et al.: A comprehensive survey on secure outsourced computation and its applications. IEEE Access **7**, 159426–159465 (2019). https://doi.org/10.1109/ACCESS.2019.2949782
26. Zhang, F., Ma, X., Liu, S.: Efficient computation outsourcing for inverting a class of homomorphic functions. Inf. Sci. **286**, 19–28 (2014). https://doi.org/10.1016/j.ins.2014.07.017
27. Zhang, H., Yu, J., Tian, C., Xu, G., Gao, P., Lin, J.: Practical and secure outsourcing algorithms for solving quadratic congruences in internet of things. IEEE Internet Things J. **7**(4), 2968–2981 (2020). https://doi.org/10.1109/JIOT.2020.2964015

28. Zhang, L., Zhang, H., Yu, J., Xian, H.: Blockchain-based two-party fair contract signing scheme. Inf. Sci. **535**, 142–155 (2020). https://doi.org/10.1016/j.ins.2020.05.054
29. Zheng, Y., Tian, C., Zhang, H., Yu, J., Li, F.: Lattice-based weak-key analysis on single-server outsourcing protocols of modular exponentiations and basic countermeasures. J. Comput. Syst. Sci. **121**, 18–33 (2021). https://doi.org/10.1016/j.jcss.2021.04.006. https://www.sciencedirect.com/science/article/pii/S0022000021000441