mathematics

MDPI

Article

An Improved Coppersmith Algorithm Based on Block Preprocessing

Lu Zhang [1,2], Baodong Qin [1,2,*], Wen Gao [1] and Yiyuan Luo [3]

[1] School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an 710121, China; zhang_lu@stu.xupt.edu.cn (L.Z.); gaowen@xupt.edu.cn (W.G.)
[2] National Engineering Research Center for Secured Wireless, Xi'an University of Posts and Telecommunications, Xi'an 710121, China
[3] School of Computer Science and Engineering, Huizhou University, Huizhou 516007, China; luoyy@hzu.edu.cn
* Correspondence: qinbaodong@xupt.edu.cn

Abstract: Since Coppersmith proposed the use of the LLL algorithm to solve univariate modular polynomial equations at EUROCRYPT'96, it has sparked a fervent research interest in lattice analysis among cryptographers. Despite its polynomial-time nature, the LLL algorithm exhibits a high-order polynomial upper bound in terms of theoretical complexity, particularly with longer computation times when applied to high-dimensional lattices. In addressing this issue, we propose an improved algorithm based on block preprocessing, building on the original Coppersmith algorithm and thus providing proof of correctness for this algorithm. This approach effectively reduces the solution time of the algorithm, offering a maximum improvement of 8.1% compared to the original Coppersmith algorithm. Additionally, we demonstrate the compatibility of our algorithm with the rounding algorithm proposed at PKC 2014. The combined utilization of these approaches further enhances the efficiency of our algorithm. The experimental results show that the combined algorithm achieves a maximum improvement of 22.4% in solution time compared to the original Coppersmith algorithm. It also outperforms the standalone rounding algorithm with a maximum improvement of 12.1%. When compared to the improved Coppersmith algorithm based on row common factor extraction, our proposed algorithm demonstrates comparable or even superior performance in certain dimensions. The block preprocessing algorithm in our approach enables independent execution without data exchange, making it suitable for leveraging multi-processing advantages in scenarios involving higher degrees of modular polynomial equations. This offers a new perspective for achieving the parallel computation of the Coppersmith algorithm, facilitating parallel execution and providing valuable insights.

Keywords: LLL algorithm; Coppersmith algorithm; block preprocessing; RSA attack

MSC: 94A60

Citation: Zhang, L.; Qin, B.; Gao, W.; Luo, Y. An Improved Coppersmith Algorithm Based on Block Preprocessing. *Mathematics* **2024**, *12*, 173. https://doi.org/10.3390/math12020173

Academic Editor: Antanas Cenys

Received: 1 December 2023
Revised: 2 January 2024
Accepted: 3 January 2024
Published: 5 January 2024

1. Introduction

Currently, the research on lattice theory in cryptography is mainly divided into two directions [1]. The first direction focuses on cryptographic design, where lattice-based hard problems are utilized to construct post-quantum secure cryptographic algorithms that can resist quantum attacks. The second direction is cryptographic analysis, which uses lattice-based algorithms to analyze the security of traditional public-key cryptographic algorithms. Lattice-based analysis methods are crucial in modern cryptography analysis. The basic idea is to transform a specific cryptographic structure, under a particular configuration, into a class of mathematical problems based on lattices. By constructing a specific lattice, these mathematical problems are converted into difficult problems on lattices. Ultimately, lattice-based reduction algorithms [2] are used to effectively solve these problems and

obtain accurate results. In practical applications, experimental data often demonstrate the superior performance of lattice basis reduction algorithms compared to theoretical results [3].

Lattice analysis has broad applications in the field of cryptography, where it is widely used to analyze the security of many public-key cryptographic algorithms. In 1978, three cryptographers from the Massachusetts Institute of Technology—Rivest, Shamir, and Adleman—introduced the well-known RSA algorithm [4]. RSA was the first practical public-key cryptographic algorithm and continues to be extensively utilized. The analysis of its security has remained a prominent research topic in the field of cryptography. During the same period, Merkle and Hellman [5] proposed the MH public-key cryptographic algorithm based on the knapsack problem. It was initially considered a potential alternative to RSA. However, the MH algorithm was later broken by Shamir [6] using integer programming problems. In 1982, A.K. Lenstra, H.W. Lenstra, and L. Lovász [7] introduced the LLL algorithm—this algorithm is capable of reducing a random input set of lattice bases to a shorter and approximately orthogonal set of lattice bases. It is commonly used for solving the approximate shortest vector problem on lattices. In 1983, Adleman [8] made a significant advancement in cryptography by successfully breaking the MH public-key cipher using the LLL algorithm. This marked the first application of the LLL algorithm in cryptographic analysis and showcased its superiority over Shamir's attack algorithm in terms of simplicity and effectiveness. At EUROCRYPT'96, Coppersmith [9] proposed an algorithm based on the LLL algorithm for solving small root problems in univariate modular polynomial equations. In a later publication [10], he further extended this approach to solving small root problems in bivariate polynomial equations with integer coefficients. These developments played a crucial role in establishing the groundwork for analyzing the RSA algorithm. In 1997 [11], Coppersmith presented a significant breakthrough in the field of cryptography. He demonstrated how the LLL algorithm can be utilized to factorize the RSA modulus when partial information is available. This pioneering work set a precedent for using lattice-based methods in the analysis of RSA. Coppersmith's pioneering work has inspired extensive research by scholars. The same year, Howgrave-Graham [12] presented an algorithm, which is equivalent to Coppersmith's method, for solving small root problems in univariate modular equations. This algorithm offers a more straightforward understanding and implementation compared to Coppersmith's original approach [9]. Afterward, the analysis of RSA primarily revolved around specific variants of the RSA algorithm [13], with particular emphasis on the analysis of the CRT-RSA algorithm [14]. In 2010, May [15] conducted a systematic review of the applications of lattice basis reduction algorithms in RSA analysis. Additionally, lattice basis reduction algorithms have been used to analyze the security of other public-key cryptographic algorithms, including DSA [16] and ECDSA [17]. Micheli [18] provided relevant examples pertaining to key recovery in situations involving the partial leakage of information for these types of cryptographic problems. In recent years, the Coppersmith algorithm has remained one of the most important algorithms for analyzing RSA and its variants [19]. In addition, it has also been used to solve the sum of EC-HNP problems under certain conditions [20] and factorization problems when a sufficient number of bits of the prime factors are known [21].

The core of the Coppersmith algorithm is to use the LLL algorithm to convert modular equations into integer equations; thus, the overall running time of the Coppersmith algorithm primarily depends on the time of the LLL algorithm. Although the LLL algorithm is polynomial time, its time complexity is $O\left(n^6 \log^3 C\right)$, where n is the dimension of the matrix and C is the maximum length of the row vectors in the matrix [7]. In practical applications, for attacking RSA algorithms with limited information, constructing higher-degree modular polynomials and reducing higher-dimensional lattice basis matrices are often necessary. However, this leads to an exponential increase in the solution time of the algorithm. Therefore, finding efficient solutions and reducing the algorithm's solution time hold significant practical significance. Motivated by the aforementioned research background, in this study, the following contributions are proposed:

- Based on the analysis of the construction and characteristics of the Coppersmith matrix, we propose a preprocessing algorithm, which involves partitioning the lattice matrix into blocks according to the degree of a modular polynomial equation. The correctness of the algorithm is rigorously proved, demonstrating its validity and reliability.
- It is proven that the algorithm proposed in this study is compatible with the rounding algorithm proposed by Bi et al. [22] at PKC 2014. The combined algorithm can be improved by up to 22.4% compared with the original Coppersmith algorithm and by up to 12.1% compared with the rounding algorithm.
- The improved Coppersmith algorithm is obtained by combining the block preprocessing and rounding algorithms. Compared with the improved Coppersmith algorithm proposed by Wang et al. [23] in 2021, the improvement effect is almost the same, and notably, our algorithm performs better in some dimensions.
- Since block preprocessing does not need to exchange data when executing the LLL algorithm, it can run independently. Therefore, this study provides a way to support the parallel operation of the Coppersmith algorithm.

Section 2 of this paper provides an introduction to the necessary background knowledge. Section 3 presents and describes the improved Coppersmith algorithm. In Section 4, we present and analyze the experimental results. Finally, in Section 5, a summary and conclusion of the paper are provided.

2. Preliminary

2.1. Notation

Definition 1 (Lattice). *Given n linearly independent vectors $b_1, b_2, \ldots, b_n \in R^m$, a lattice L is the set of all integer linear combinations, defined as*

$$L(b_1, b_2 \ldots b_n) = \left\{ \sum_{i=1}^{n} x_i b_i | x_i \in Z \right\}.$$

We call $b_1, b_2, \ldots, b_n$ a basis of the lattice, n is referred to as the rank of the lattice, and m is referred to as the dimension of the lattice. In general, $n \leq m$; if $n = m$, the lattice is called a full-rank lattice. In most cases, we primarily consider examples of full-rank lattices.

When dealing with lattice bases, a more compact matrix form can be used. The matrix consists of column vectors $b_1, b_2, \ldots, b_n$ representing the basis of the lattice. In other words, we can express it as

$$B = \begin{pmatrix} | & & | \\ b_1 & \cdots & b_n \\ | & & | \end{pmatrix}.$$

Thus, the lattice can be represented as

$$L(B) = \{Bx | x \in Z^n\}.$$

Definition 2 (Unimodular Matrix). *A matrix $U \in Z^{n \times n}$ is called unimodular if* $\det(U) = \pm 1$. *For example, the matrix* $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ *is unimodular.*

Corollary 1. *Two bases $B_1, B_2 \in R^{m \times n}$ are equivalent if, and only if, $B_2 = B_1 U$ for some unimodular matrix U. This implies that for a given lattice, there can be multiple lattice bases. Any two lattice bases differ by a unimodular matrix.*

Definition 3 (Fundamental Parallelepiped). *Given n linearly independent vectors $b_1, b_2, \ldots, b_n$, their fundamental parallelepiped is defined as*

$$P(b_1, b_2, \ldots, b_n) = \left\{ \sum_{i=1}^{n} x_i b_i | x_i \in R,\ 0 \leq x_i < 1 \right\}.$$

Thus, a fundamental parallelepiped is the (half-open) region enclosed by the vectors $b_1, b_2, \ldots, b_n$. Evidently, different bases of the same lattice generate different fundamental parallelepipeds.

Definition 4 (Determinant). *Let L be a lattice of rank n. We define the determinant of L, denoted $\det(L)$, as the n-dimensional volume of $P(L)$. In symbols, this can be written as*

$$\det(L) = \sqrt{\det(B^T B)}.$$

Particularly, if the lattice L is a full-rank lattice and B is a square matrix, the formula is

$$\det(L) = \sqrt{\det(B^T B)} = \sqrt{\det(B)\det(B^T)} = |\det(B)|.$$

Definition 5 (Euclidean Norm). *The Euclidean norm (also known as the 2-norm) of a vector $V = (v_1, v_2, v_3, \ldots, v_n)$, referred to as the length of the vector, is defined as*

$$\|V\| = \sqrt{\sum_{i=1}^{n} v_i^2}.$$

Definition 6 (Polynomial Norm). *Let $f(x) = \sum_{i=1} a_i x^i$, where $a_i \neq 0$. We define the polynomial norm of $f(x)$ as*

$$\|f(x)\| = \left(\sum_i a_i^2\right)^{\frac{1}{2}}.$$

Definition 7 (Gram–Schmidt Orthogonalization). *For a sequence of n linearly independent vectors $b_1, b_2, \ldots, b_n$, we define their Gram–Schmidt orthogonalization as*

$$\tilde{b}_1 = b_1,$$

$$\tilde{b}_i = b_i - \sum_{j=1}^{i-1} \mu_{i,j} \tilde{b}_i,\ 1 < i \leq n.$$

Thus, $\tilde{b}_j$ is the component of b_i orthogonal to $\tilde{b}_1, \ldots, \tilde{b}_{i-1}$, and $\mu_{i,j}$ is called the Gram–Schmidt coefficients.

2.2. LLL Algorithm

In 1982, A.K. Lenstra, H.W. Lenstra, and L. Lovász [7] introduced the LLL lattice basis reduction algorithm. Although it does not directly yield the shortest vector exactly, it can produce a short vector with an approximate factor of $\left((1+\varepsilon)\sqrt{4/3}\right)^{(n-1)/2}$ in polynomial time. Here, ε is a positive constant, and n represents the dimension of the lattice [24]. The LLL algorithm ensures that the length of the output vectors does not exceed a multiple of the shortest vector's length. Notably, in practical applications, the algorithm exhibits a lower time complexity compared to its theoretical complexity [25]. In 2006, Nguyen and Stehlé [26] conducted extensive experiments on the LLL algorithm, revealing its superior ability to approximate the shortest vector in lower-dimensional lattices when compared to higher-dimensional cases.

Definition 8 (LLL Reduction Basis). *A basis $B = \{b_1, b_2, \ldots, b_n\} \in R^m$ is a $\delta - LLL$ reduction basis if the following holds*

(1) $\forall 1 \leq i \leq n,\ j < i,\ \left|\mu_{i,j}\right| \leq \frac{1}{2}$;

(2) $\forall 1 \leq i < n,\ \delta\left\|\tilde{b}_i\right\|^2 \leq \left\|\mu_{i+1,i}\tilde{b}_i + \tilde{b}_{i+1}\right\|^2$.

In other words, $\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_n$ are the results of the Gram–Schmidt orthogonalization on $b_1, b_2, \ldots, b_n$. Since $\tilde{b}_i$ and $\tilde{b}_{i+1}$ are orthogonal, the second condition in Definition 8 can be written as

$$\forall 1 \leq i < n, \ \left\|\tilde{b}_{i+1}\right\|^2 \geq \left(\delta - \mu_{i+1,i}^2\right)\left\|\tilde{b}_i\right\|^2.$$

Given any basis $b_1, b_2, \ldots, b_n$ of a lattice, the LLL algorithm can produce a reduced basis by running in polynomial time. The specific steps of the LLL algorithm are depicted in Algorithm 1.

Algorithm 1 LLL Algorithm

Input: A basis $b_1, b_2, \ldots, b_n$.
Output: A $\delta -$ LLL reduction basis.

```
Compute b̃_1, b̃_2, ..., b̃_n.
for i = 2 to n do
    for j = i − 1 to 1 do
        b_i ← b_i − μ_{i,j} b_j, where μ_{i,j} = ⌈⟨b_i, b̃_j⟩ / ⟨b̃_j, b̃_j⟩⌋.
    end for
end for
if ∃ i such that ‖b̃_{i+1}‖ < √(δ − μ²_{i+1,i}) ‖b̃_i‖ then
    swap b_i and b_{i+1}.
    return step 1.
end if
Return b_1, b_2, ..., b_n.
```

Remark 1. *We use $\lceil \cdot \rfloor$ to denote rounding to the nearest integer; for example, $\lceil 3.3 \rfloor = 3$, $\lceil 3.8 \rfloor = 4$.*

Here, we present a significant theorem regarding the length relationship among the output vectors of the LLL algorithm.

Theorem 1 ([7]). *For a lattice L, the LLL algorithm can find a set of reduced basis $b_1, \ldots, b_n$ in polynomial time, satisfying the following condition*

$$\|b_1\| \leq \|b_2\| \leq \ldots \leq \|b_n\| \leq 2^{\frac{n(n-1)}{4(n-i+1)}} \det(L)^{\frac{1}{n-i+1}}, \ 1 \leq i \leq n.$$

2.3. Coppersmith Algorithm

In their work [9], Coppersmith proposed a small root algorithm for solving univariate modular polynomial equations. In the following sections, we present an overview of this algorithm and provide a detailed description of the specific steps.

Definition 9 (Univariate Modular Equation). *Let N be an integer of unknown factorization, where b is one of its factors satisfying $b \geq N^\beta$, $0 < \beta \leq 1$. Let $f(x)$ be a univariate monic polynomial of degree δ. The objective is to find all small roots of $f(x)$, that is,*

$$f(x_0) \equiv 0 \bmod b, \ |x_0| \leq X,$$

where X represents the absolute upper bound for the small root solutions.

The goal of this problem is to maximize the upper bound X of the roots, allowing for a greater number of output roots satisfying the given condition while ensuring that the computational time complexity remains polynomial. In other words, the algorithm's running time is a polynomial function of $(\log N, \delta)$.

Theorem 2 (Coppersmith [15]). *Let N be an integer of unknown factorization, where b is one of its factors satisfying $b \geq N^{\beta}, 0 < \beta \leq 1$. Furthermore, let $f(x)$ be a univariate monic polynomial of degree δ, $0 < \varepsilon \leq \frac{1}{7}\beta$. Then, we can find all solutions for the following equation:*

$$f(x_0) \equiv 0 \bmod b,\ |x_0| \leq \frac{1}{2}N^{\frac{\beta^2}{\delta}-\varepsilon}.$$

The running time primarily depends on the time it takes the LLL algorithm to reduce a lattice basis of dimension $O(\varepsilon^{-1}\delta)$ with entries of bit-size $O(\varepsilon^{-1}\log N)$, which can be achieved in time $O\left(\varepsilon^{-7}\delta^5\log^2 N\right)$. Moreover, Coppersmith also provided a specific case of this theorem by using an exhaustive search, namely it is possible to eliminate the terms $\frac{1}{2}$ and ε described in Theorem 3.

Theorem 3 (Coppersmith [15]). *Let N be an integer of unknown factorization. Furthermore, let $f_N(x)$ be a univariate monic polynomial of degree δ. Then, we can find all solutions x_0 for the following equation*

$$f_N(x_0) \equiv 0 \mod N,\ |x_0| \leq N^{\frac{1}{\delta}},$$

in time $O\left(\delta^5\log^9 N\right)$.

In 1997, Howgrave-Graham summarized and simplified Coppersmith's original algorithm, presenting a more practical lattice algorithm [12]. Nowadays, the majority of applied Coppersmith algorithms are based on Howgrave-Graham's implementation. The subsequent references to the Coppersmith algorithm in this paper specifically refer to the improved algorithm by Howgrave-Graham. Here, we provide a detailed explanation of this algorithm.

The core idea of the Coppersmith algorithm is to transform the problem of finding small roots of a univariate modular equation into the problem of finding small roots of a univariate integer equation. In other words, the algorithm aims to solve the following problem

$$f(x_0) \equiv 0 \bmod N \ \Rightarrow\ g(x_0) = 0,\ |x_0| \leq X.$$

Thus, the goal is to find such values of $g(x)$. Howgrave-Graham presents the following theorem [12].

Theorem 4 (Howgrave-Granham [15]). *Let $g(x)$ be a univariate polynomial with n monomials. Further, let m be a positive integer. Suppose that*

(1) $g(x_0) \equiv 0 (\bmod N^m)$, *where $x_0 < X$;*

(2) $\|g(xX)\| < \frac{N^m}{\sqrt{n}}$.

Then, $g(x) = 0$ holds over the integers.

In the specific algorithm, the construction of $g(x)$ can be divided into the following two steps:

Step 1: Fix a positive integer and construct a set of polynomials where each polynomial shares a common root but has a modulus N^m. The commonly used approach for constructing is

$$g_{i,j}(x) = x^j f(x)^i N^{m-i},\ i = 0,\ldots,m-1,\ j = 0,\ldots,\delta-1.$$

Thus, it is easy to observe that

$$g_{i,j}(x_0) = x_0^j f(x_0)^i N^{m-i} = x_0^j\left(\frac{f(x_0)}{N}\right)^i N^m \equiv 0 \bmod N^m.$$

Step 2: Construct an integer coefficient linear combination of $g(x)$ in set C to satisfy $|g(x_0)| < N^m$. By combining this with $g(x_0) \equiv 0 \bmod N^m$ in step 1, we establish the validity of $g(x_0) = 0$ over integers.

The condition in step 2 can be achieved through the LLL algorithm, as demonstrated in Theorem 1. After applying the LLL algorithm for reduction, the length of the first output vector satisfies a certain upper bound requirement. We can utilize this vector to construct

$$\|b_1\| = \|g(xX)\| < \frac{N^m}{\sqrt{n}}.$$

As a result, polynomials with smaller norms can be obtained, which further allows for the deduction of $|g(x_0)| < N^m$; May provides proof of this process in [15]. Therefore, by following the aforementioned steps, a suitable $g(x)$ is found. It shares a common root with $f(x)$ and has an integer root. Finally, the solution can be obtained using a general method for solving integer equations. The general steps of the Howgrave-Graham-improved Coppersmith algorithm are presented in Algorithm 2.

Algorithm 2 Original Coppersmith Algorithm

Input: A univariate monic polynomial of degree d with modulus N, parameter β, ε, m, X.
Output: All $x_0 \in Z$, such that $|x_0| \leq N^{\frac{1}{\delta}}$ and $f(x_0) \equiv 0 \bmod N$.

1: Choose $\beta = 1$, $\varepsilon = \frac{\beta}{7}$, $m = \left\lceil \frac{\beta^2}{d\varepsilon} \right\rceil$.
2: Compute polynomials $g_{i,j}(x) = x^j \cdot N^{m-i} \cdot f(x)^i$, $i = 0, \ldots, m-1$, $j = 0, \ldots, d-1$.
3: Compute $X = \left\lceil \frac{1}{2} N^{\frac{\beta^2}{d} - \varepsilon} \right\rceil$, where X is the bound of x_0.
4: Build the $n \times n$ matrix B, the rows of which are the $g(xX)$ coefficients.
5: Apply the LLL algorithm to the lattice basis B. Let v be the shortest vector in the LLL-reduced basis. The vector v is the coefficient vector of some polynomial; then, construct $g(x)$ from v.
6: Find the set R of all roots of $g(x)$ over the integers using standard methods. For every root $x_0 \in R$, check whether $\gcd(N, f(x_0)) \geq N^{\beta}$. If this condition is not satisfied, remove x_0 from R.
7: Return R.

3. Improved Coppersmith Algorithm

3.1. Block Preprocessing Algorithm

According to the number of modular polynomial equations, it is necessary to partition the lattice basis matrix before applying the LLL reduction algorithm. To illustrate the concept of partitioning, let us consider an example using a monic polynomial $f(x) = x^3 + a_{12}x^2 + a_{11}x + a_{10}$ of degree 3, where the square of $f(x)$ is denoted as $f(x)^2 = x^6 + a_{25}x^5 + a_{24}x^4 + a_{23}x^3 + a_{22}x^2 + a_{21}x + a_{20}$. Computing the small roots of a polynomial equation $f(x)$ with modulus N. The Coppersmith matrix constructed by selecting parameter $m = 3$ is represented as follows

$$\begin{array}{c} \\ g_{0,0}(xX) \\ g_{0,1}(xX) \\ g_{0,2}(xX) \\ g_{1,0}(xX) \\ g_{1,1}(xX) \\ g_{1,2}(xX) \\ g_{2,0}(xX) \\ g_{2,1}(xX) \\ g_{2,2}(xX) \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 & x^8 \end{array} \\ \begin{bmatrix} N^3 & & & & & & & & \\ & XN^3 & & & & & & & \\ & & X^2N^3 & & & & & & \\ a_{10}N & a_{11}XN^2 & a_{12}X^2N^2 & X^3N^2 & & & & & \\ & a_{10}XN^2 & a_{11}X^2N^2 & a_{12}X^3N^2 & X^4N^2 & & & & \\ & & a_{10}X^2N^2 & a_{11}X^3N^2 & a_{12}X^4N^2 & X^5N^2 & & & \\ a_{20}N & a_{21}XN & a_{22}X^2N & a_{23}X^3N & a_{24}X^4N & a_{25}X^5N & X^6N & & \\ & a_{20}XN & a_{21}X^2N & a_{22}X^3N & a_{23}X^4N & a_{24}X^5N & a_{25}X^6N & X^7N & \\ & & a_{20}X^2N & a_{21}X^3N & a_{22}X^4N & a_{23}X^5N & a_{24}X^6N & a_{25}X^7N & X^8N \end{bmatrix} \end{array}$$

In the above case of the polynomial $f(x)$ with a degree of 3, the constructed lattice basis matrix B is evenly partitioned into three blocks. Each block is separately preprocessed via the LLL algorithm. Finally, the blocks are reassembled in order, as shown in Algorithm 3. Step 6 means choosing a vector from row $d \cdot (i-1)$ to row $d \cdot i$ of matrix B, which is denoted as B_i.

Algorithm 3 Improved Coppersmith Algorithm Based on Block Preprocessing

Input: A univariate degree d monic polynomial $f(x)$ with modulus N, parameter β, ε, m, X.
Output: All $x_0 \in Z$ such that $|x_0| \leq N^{\frac{1}{\delta}}$ and $f(x_0) \equiv 0 \bmod N$.

1: Choose $\beta = 1$, $\varepsilon = \frac{\beta}{7}$, $m = \left\lceil \frac{\beta^2}{d\varepsilon} \right\rceil$.
2: Compute polynomials $g_{i,j}(x) = x^j \cdot N^{m-i} \cdot f(x)^i$, $i = 0, \ldots, m-1$, $j = 0, \ldots, d-1$.
3: Compute $X = \left\lceil \frac{1}{2} N^{\frac{\beta^2}{d} - \varepsilon} \right\rceil$, where X is the bound of x_0.
4: Build the $n \times n$ matrix B, the rows of which are the $g(xX)$ coefficients.
5: **for** $i = 1$ to m **do**
6: $\quad B_i = B[d \cdot (i-1) : d \cdot i]$.
7: $\quad$ Applying the LLL algorithm on each block matrix B_i produces ${B_i}'$.
8: **end for**
9: Combine $B' = \left({B_1}' \parallel {B_2}' \parallel \ldots \parallel {B_m}'\right)$ in order.
10: Apply the LLL algorithm to the lattice basis B. Let v be the shortest vector in the LLL-reduced basis. The vector v is the coefficient vector of some polynomial; then, construct $g(x)$ from v.
11: Find the set R of all roots of $g(x)$ over the integers using standard methods. For every root $x_0 \in R$, check whether $\gcd(N, f(x_0)) \geq N^{\beta}$. If this condition is not satisfied, remove x_0 from R.
12: Return R.

Applying the LLL algorithm to both the preprocessed matrix B' and the original matrix B yields completely consistent results. This implies that the preprocessing algorithm does not affect the correctness of the subsequent execution of the LLL algorithm, as depicted in Theorem 5.

Theorem 5. *Let B be a lattice basis matrix of rank n, and divide matrix B into m $(m \leq n)$ blocks along the rows as $B = (B_1 \parallel B_2 \parallel B_3 \parallel \ldots \parallel B_m)$; let B' be the LLL-reduced result of matrix B, then $B' = \left({B_1}' \parallel {B_2}' \parallel \ldots \parallel {B_m}'\right)'$. The parameter m is chosen for constructing the Coppersmith matrix.*

Based on the construction characteristics of Algorithm 2, the dimension of the Coppersmith matrix is equal to the product of degree δ and parameter m. Therefore, when dividing matrix B into m blocks along the rows, there is no case in which it cannot be evenly divided.

To prove Theorem 5 without the loss of generality, we randomly select a lattice basis matrix B of rank 4. This choice allows us to avoid the inadequacy of a low-rank lattice basis in illustrating the problem, as well as the excessive complexity of proof steps associated with a high-rank lattice basis. We can divide the lattice basis matrix B into two rank-2 matrices $B_1 = (b_1, b_2)$ and $B_2 = (b_3, b_4)$. After applying the LLL algorithm for reduction, matrix B is transformed into matrix $B' = \left(\widetilde{b_1}, \widetilde{b_2}, \widetilde{b_3}, \widetilde{b_4}\right)$, where the vector form is denoted as

$$
\begin{aligned}
\widetilde{b_1} &= b_1 \\
\widetilde{b_2} &= b_2 - \lceil \langle b_2, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \rfloor \cdot \widetilde{b_1} \\
\widetilde{b_3} &= b_3 - \lceil \langle b_3, \widetilde{b_2} \rangle / ||\widetilde{b_2}||^2 \rfloor \cdot \widetilde{b_2} - \lceil \langle b_3, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \rfloor \cdot \widetilde{b_1} \\
\widetilde{b_4} &= b_4 - \lceil \langle b_4, \widetilde{b_3} \rangle / ||\widetilde{b_3}||^2 \rfloor \cdot \widetilde{b_3} - \lceil \langle b_4, \widetilde{b_2} \rangle / ||\widetilde{b_2}||^2 \rfloor \cdot \widetilde{b_2} - \lceil \langle b_4, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \rfloor \cdot \widetilde{b_1}.
\end{aligned}
$$

Similarly, after the application of the LLL algorithm for reduction, the matrix B_1 is transformed into matrix $B_1{}' = (b_1{}', b_2{}')$, while matrix B_2 is transformed into matrix $B_2{}' = (b_3{}', b_4{}')$. The vector forms are represented as

$$b_1{}' = b_1 \qquad b_3{}' = b_3$$
$$b_2{}' = b_2 - \lceil \langle b_2, b_1{}' \rangle / ||b_1{}'||^2 \rfloor \cdot b_1{}' \qquad b_4{}' = b_4 - \lceil \langle b_4, b_3{}' \rangle / ||b_3{}'||^2 \rfloor \cdot b_3{}'.$$

After combining them, we can obtain the matrix $B_{12}{}' = (B_1{}' \parallel B_2{}') = (b_1{}', b_2{}', b_3{}', b_4{}')$. The reduction result obtained after executing the LLL algorithm on matrix $B_{12}{}'$ is $B_{12}{}'' = (b_1{}'', b_2{}'', b_3{}'', b_4{}'')$, where the vector form is

$$b_1{}'' = b_1{}'$$
$$b_2{}'' = b_2{}' - \lceil \langle b_2{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}''$$
$$b_3{}'' = b_3{}' - \lceil \langle b_3{}', b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' - \lceil \langle b_3{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}''$$
$$b_4{}'' = b_4{}' - \lceil \langle b_4{}', b_3{}'' \rangle / ||b_3{}''||^2 \rfloor \cdot b_3{}'' - \lceil \langle b_4{}', b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' - \lceil \langle b_4{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}''.$$

Proof of Theorem 5. If we can obtain $b_i{}'' = \widetilde{b_i}$ $(i = 1, 2, 3, 4)$, it can be proven that the two are equivalent. The specific proof process is given below

$$\begin{aligned}
b_1{}'' &= b_1{}' = b_1 = \widetilde{b_1} \\
b_2{}'' &= b_2{}' - \lceil \langle b_2{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}'' \\
&= b_2 - \lceil \langle b_2, b_1{}' \rangle / ||b_1{}'||^2 \rfloor \cdot b_1{}' - \lceil \langle b_2 - \lceil \langle b_2, b_1{}' \rangle / ||b_1{}'||^2 \rfloor \cdot b_1{}', b_1{}' \rangle / ||b_1{}'||^2 \rfloor \cdot b_1{}' \\
&= b_2 - \lceil \langle b_2, b_1{}' \rangle / ||b_1{}'||^2 \rfloor \cdot b_1{}' - \lceil \langle b_2, b_1{}' \rangle / ||b_1{}'||^2 - \lceil \langle b_2, b_1{}' \rangle / ||b_1{}'||^2 \rfloor \rfloor \cdot b_1{}' \\
&= b_2 - \lceil \langle b_2, b_1{}' \rangle / ||b_1{}'||^2 \rfloor \cdot b_1{}' \\
&= b_2 - \lceil \langle b_2, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \rfloor \cdot \widetilde{b_1} \\
&= \widetilde{b_2} \\
b_3{}'' &= b_3{}' - \lceil \langle b_3{}', b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' - \lceil \langle b_3{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}'' \\
&= b_3 - \lceil \langle b_3, b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' - \lceil \langle b_3, b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}'' \\
&= b_3 - \lceil \langle b_3, \widetilde{b_2} \rangle / ||\widetilde{b_2}||^2 \rfloor \cdot \widetilde{b_2} - \lceil \langle b_3, \widetilde{b_1} \rangle / ||\widetilde{b_1}|^2 \rfloor \cdot \widetilde{b_1} \\
&= \widetilde{b_3} \\
b_4{}'' &= b_4{}' - \lceil \langle b_4{}', b_3{}'' \rangle / ||b_3{}''||^2 \rfloor \cdot b_3{}'' - \lceil \langle b_4{}', b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' - \lceil \langle b_4{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}'' \\
&= b_4 - \lceil \langle b_4, b_3{}' \rangle / ||b_3{}'||^2 \rfloor \cdot b_3{}' \\
&\quad - \lceil \langle b_4, b_3{}'' \rangle / ||b_3{}''||^2 - \lceil \langle b_4, b_3{}' \rangle / ||b_3{}'||^2 \rfloor \cdot \langle b_3{}', b_3{}'' \rangle / ||b_3{}''||^2 \rfloor \cdot b_3{}'' \\
&\quad - \lceil \langle b_4, b_2{}'' \rangle / ||b_2{}''||^2 - \lceil \langle b_4, b_3{}' \rangle / ||b_3{}'||^2 \rfloor \cdot \langle b_3{}', b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' \\
&\quad - \lceil \langle b_4, b_1{}'' \rangle / ||b_1{}''||^2 - \lceil \langle b_4, b_3{}' \rangle / ||b_3{}'||^2 \rfloor \cdot \langle b_3{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}'' \\
&= b_4 - \lceil \langle b_4, b_3{}' \rangle / ||b_3{}'||^2 \rfloor \cdot b_3{}' - \lceil \langle b_4, b_3{}'' \rangle / ||b_3{}''||^2 \rfloor \cdot b_3{}'' - \lceil \langle b_4, b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' - \lceil \langle b_4, b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}'' \\
&\quad + \lceil \langle b_4, b_3{}' \rangle / ||b_3{}'||^2 \rfloor \cdot \left(\lceil \langle b_3{}', b_3{}'' \rangle / ||b_3{}''||^2 \rfloor \cdot b_3{}'' + \lceil \langle b_3{}', b_2{}'' \rangle / ||b_2{}''||^2 \rfloor \cdot b_2{}'' + \lceil \langle b_3{}', b_1{}'' \rangle / ||b_1{}''||^2 \rfloor \cdot b_1{}'' \right) \\
&= b_4 - \lceil \langle b_4, b_3 \rangle / ||b_3||^2 \rfloor \cdot b_3 - \lceil \langle b_4, \widetilde{b_3} \rangle / ||\widetilde{b_3}||^2 \rfloor \cdot \widetilde{b_3} - \lceil \langle b_4, \widetilde{b_2} \rangle / ||\widetilde{b_2}||^2 \rfloor \cdot \widetilde{b_2} - \lceil \langle b_4, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \rfloor \cdot \widetilde{b_1} \\
&\quad + \lceil \langle b_4, b_3 \rangle / ||b_3||^2 \rfloor \cdot \left(\lceil \langle b_3, \widetilde{b_3} \rangle / ||\widetilde{b_3}||^2 \rfloor \cdot \widetilde{b_3} + \lceil \langle b_3, \widetilde{b_2} \rangle / ||\widetilde{b_2}||^2 \rfloor \cdot \widetilde{b_2} + \lceil \langle b_3, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \rfloor \cdot \widetilde{b_1} \right) \\
&= b_4 - \lceil \langle b_4, b_3 \rangle / ||b_3||^2 \rfloor \cdot b_3 - \lceil \langle b_4, \widetilde{b_3} \rangle / ||\widetilde{b_3}||^2 \rfloor \cdot \widetilde{b_3} - \lceil \langle b_4, \widetilde{b_2} \rangle / ||\widetilde{b_2}||^2 \rfloor \cdot \widetilde{b_2} - \lceil \langle b_4, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \rfloor \cdot \widetilde{b_1} \\
&\quad + \lceil \langle b_4, b_3 \rangle / ||b_3||^2 \rfloor \cdot b_3 \\
&= b_4 - \langle b_4, \widetilde{b_3} \rangle / ||\widetilde{b_3}||^2 \cdot \widetilde{b_3} - \langle b_4, \widetilde{b_2} \rangle / ||\widetilde{b_2}||^2 \cdot \widetilde{b_2} - \langle b_4, \widetilde{b_1} \rangle / ||\widetilde{b_1}||^2 \cdot \widetilde{b_1} \\
&= \widetilde{b_4}.
\end{aligned}$$

From the above deduction, we can conclude that $b_i{}'' = \widetilde{b_i} (i = 1, 2, 3, 4)$, indicating that applying the LLL algorithm to the lattice basis matrix with block partitioning and sequential combination yields results consistent with directly applying the LLL algorithm to the entire matrix. When the rank of the lattice basis matrix increases, the number of lattice bases that need to be reduced also increases. However, fundamentally, the same block partitioning concept and computational steps are used. Therefore, this approach can be extended to the case of rank n. □

3.2. Rounding Algorithm

At PKC 2014, Bi et al. [22] proposed a rounding algorithm. This algorithm simplifies the row vectors through elementary row operations, ensuring that each non-diagonal element in the matrix is smaller than the corresponding diagonal element in its column. It is guaranteed to reduce the size of elements when the matrix dimension is unchanged, aiming to shorten the running time in the subsequent execution of the LLL algorithm. See Algorithm 4 for more details.

Algorithm 4 Improved Coppersmith Algorithm Based on Rounding

Input: A univariate degree d monic polynomial $f(x)$ with modulus N, parameter β, ε, m, X.
Output: All $x_0 \in Z$ such that $|x_0| \leq N^{\frac{1}{\delta}}$ and $f(x_0) \equiv 0 \bmod N$.

1: Choose $\beta = 1$, $\varepsilon = \frac{\beta}{7}$, $m = \left\lceil \frac{\beta^2}{d\varepsilon} \right\rceil$.
2: Compute polynomials $g_{i,j}(x) = x^j \cdot N^{m-i} \cdot f(x)^i$, $i = 0, \ldots, m-1$, $j = 0, \ldots, d-1$.
3: Compute $X = \left\lceil \frac{1}{2} N^{\frac{\beta^2}{d} - \varepsilon} \right\rceil$, where X is the bound of x_0.
4: Build the $n \times n$ matrix B, the rows of which are the $g(xX)$ coefficients.
5: **for** $i = n-2$ to 0 **do**
6: **for** $j = i+1$ to $n-1$ **do**
7: $CO = \lfloor (B[j,i]/B[i,i]) \rfloor$.
8: $B[j] = B[j] - CO \cdot B[i]$.
9: **end for**
10: **end for**
11: Apply the LLL algorithm to the lattice basis B. Let v be the shortest vector in the LLL-reduced basis. The vector v is the coefficient vector of some polynomial; then, construct $g(x)$ from v.
12: Find the set R of all roots of $g(x)$ over the integers using standard methods. For every root $x_0 \in R$, check whether $\gcd(N, f(x_0)) \geq N^{\beta}$. If this condition is not satisfied, remove x_0 from R.
13: Return R.

The lattice basis matrix constructed using the Coppersmith algorithm is a lower triangular matrix, with all elements in the upper right corner being zero. Consequently, the algorithm is designed to operate from the lower right corner to the upper left corner of the matrix. This transformation process preserves the main diagonal elements and does not alter the determinant of the matrix, ensuring the correctness of subsequent computations.

3.3. Improved Coppersmith Algorithm

Wang et al. [23] introduced an improved algorithm in 2021 that combines the structural characteristics and element properties of the lattice basis matrix in the Coppersmith algorithm. By iteratively extracting common factors from the row vectors of different blocks in the lattice basis matrix, the solving time of the Coppersmith algorithm is effectively reduced. Furthermore, they demonstrated the compatibility between the row common factor extraction algorithm and the rounding algorithm. The experimental results showed that the combined algorithm further improved the efficiency of the Coppersmith algorithm. The key distinction of this work is the proposal of an improved algorithm based on block preprocessing and its compatibility with the rounding algorithm. As described in Section 3.1, the constructed lattice basis matrix is first divided into blocks for preprocessing using the LLL algorithm; then, the remaining steps are carried out. Through testing on the same software platform and with the same parameter settings, the experimental results demonstrated that both the proposed improved algorithm with the rounding algorithm and the improved algorithm, as proposed by Wang et al., with the rounding algorithm correctly solve the problem. In terms of time efficiency, these two improved algorithms show consistent improvements over the original Coppersmith algorithm. In certain di-

mensions, the proposed algorithm in this work performs even better. Additionally, the proposed improved algorithm in this work does not require data exchanging during block preprocessing, as each block can be processed independently. This provides a method to support the Coppersmith algorithm in parallel. The following theorem provides a detailed introduction to the proposed improved algorithm and demonstrates its compatibility with the rounding algorithm, as outlined in Theorem 6.

Theorem 6. *After executing the block preprocessing algorithm on the Coppersmith matrix, the output matrix remains a lower triangular matrix, which does not affect the subsequent execution of the rounding algorithm.*

Proof of Theorem 6. According to Algorithm 2, the initial Coppersmith matrix is a lower triangular matrix. Taking the construction of a Coppersmith matrix with a dimension of 9×9 as an example from Section 3.1, utilizing the idea of block preprocessing, the constructed lattice basis matrix B can be divided into three matrices of dimension 3×9. The vector form of the second block matrix is as follows.

$$\begin{array}{lccccccccc} b_4 = (& a_{10}N^2 & a_{11}XN^2 & a_{12}X^2N^2 & X^3N^2 & 0 & 0 & 0 & 0 & 0 \ ) \\ b_5 = (& 0 & a_{10}XN^2 & a_{11}X^2N^2 & a_{12}X^3N^2 & X^4N^2 & 0 & 0 & 0 & 0 \ ) \\ b_6 = (& 0 & 0 & a_{10}X^2N^2 & a_{11}X^3N^2 & a_{12}X^4N^2 & X^5N^2 & 0 & 0 & 0 \ ) \end{array}$$

It can be seen that when applying the LLL algorithm to the above block during the reduction step $b_i \leftarrow b_i - \mu_{i,j}b_j$, vectors b_4, b_5, b_6 have zero elements in the later components. Since the reduction process proceeds from lower to upper, the computation results will not affect the form of the later components. However, it may change the portion of the earlier components of vectors b_5 and b_6, where the elements are zero. Due to the construction method of the Coppersmith algorithm, the non-zero elements in the vector of each row of the block matrices are N times the non-zero elements of the vector of the previous row, arranged in increasing order of row vector lengths. Therefore, the swapping step is not affected. After executing the LLL algorithm, the matrix can still maintain the aforementioned stepped structure. Similarly, for each subsequent block, as the dimension of the matrix increases, the number of non-zero elements in the vectors increases accordingly. However, during the reduction process, the calculation formulae between the elements remain the same, and the execution order remains unchanged. Therefore, each block retains its stepped structure after executing the LLL algorithm. When combining all the blocks in order, the resulting matrix remains a lower triangular matrix, ensuring that the subsequent execution of the rounding algorithm is not affected. □

The rounding algorithm is essentially a row-equivalent transformation, representing the original lattice matrix. Thus, the transformation does not impact the correctness of the final result. Combining it with the block preprocessing algorithm enhances the solving efficiency of the Coppersmith algorithm. This integration leads to the development of the improved Coppersmith algorithm, as presented in Algorithm 5.

Algorithm 5 Improved Coppersmith Algorithm Based on Block Preprocessing and Rounding

Input: A univariate degree d monic polynomial $f(x)$ with modulus N, parameter β, ε, m, X.

Output: All $x_0 \in Z$ such that $|x_0| \leq N^{\frac{1}{\delta}}$ and $f(x_0) \equiv 0 \bmod N$.

1: Choose $\beta = 1$, $\varepsilon = \frac{\beta}{7}$, $m = \left\lceil \frac{\beta^2}{d\varepsilon} \right\rceil$.
2: Compute polynomials $g_{i,j}(x) = x^j \cdot N^{m-i} \cdot f(x)^i$, $i = 0, \ldots, m-1$, $j = 0, \ldots, d-1$.
3: Compute $X = \left\lceil \frac{1}{2} N^{\frac{\beta^2}{d} - \varepsilon} \right\rceil$, where X is the bound of x_0.
4: Build the $n \times n$ matrix B, the rows of which are the $g(xX)$ coefficients.
5: **for** $i = 1$ to m **do**
6: $\quad B_i = B[d \cdot (i-1) : d \cdot i]$.
7: $\quad$ Applying the LLL algorithm on each block matrix B_i produces $B_i{}'$.
8: **end for**
9: Combine $B' = \left(B_1{}' \parallel B_2{}' \parallel \ldots \parallel B_m{}'\right)$ in order.
10: **for** $i = n-2$ to 0 **do**
11: $\quad$ **for** $j = i+1$ to $n-1$ **do**
12: $\quad\quad CO = \lfloor (B[j,i]/B[i,i]) \rfloor$.
13: $\quad\quad B[j] = B[j] - CO \cdot B[i]$.
14: $\quad$ **end for**
15: **end for**
16: Apply the LLL algorithm to the lattice basis B. Let v be the shortest vector in the LLL-reduced basis. The vector v is the coefficient vector of some polynomial; then, construct $g(x)$ from v.
17: Find the set R of all roots of $g(x)$ over the integers using standard methods. For every root $x_0 \in R$, check whether $\gcd(N, f(x_0)) \geq N^{\beta}$. If this condition is not satisfied, remove x_0 from R.
18: Return R.

4. Experiments and Analysis

To verify the solving efficiency of the improved Coppersmith algorithm, simulation experiments were conducted on the proposed and comparative algorithms. The simulations were performed using a computer with the following configuration:

- RAM: 32 GB;
- CPU: 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz;
- Operating System: Ubuntu 20.04.5 (64-bit).

The simulation experiments were conducted using SageMath 9.0 software, each experiment was conducted during the same time period and network environment. The following five algorithms were included:

- **Algorithm 2(Alg. 2):** the original Coppersmith algorithm [12];
- **Algorithm 3(Alg. 3):** the improved Coppersmith algorithm based on the block preprocessing algorithm;
- **Algorithm 4(Alg. 4):** the improved Coppersmith algorithm based on the rounding algorithm [22];
- **Algorithm 5(Alg. 5):** the improved Coppersmith algorithm based on the block preprocessing and rounding algorithm;
- **Algorithm 6(Alg. 6):** the improved Coppersmith algorithm based on the row common factor extraction algorithm and rounding algorithm [23].

Referring to the problem described in reference [27] and the commonly used RSA algorithm parameters in practical applications, we selected a polynomial equation with a degree of 3 and a modulus of 2048 bits for the experiments. We then constructed the corresponding Coppersmith matrix using parameters $m = 8, 9, \ldots, 19, 20$ to solve the small roots of the equation. All experiments were able to correctly solve the problem. We performed 10 experiments for each parameter and calculated the average solving time.

The results are shown in Table 1, with the time measured in seconds. We compared the performance of the algorithm with the following five aspects:

- **Property 1(Prop. 1):** the performance improvement in Algorithm 3 compared to Algorithm 2;
- **Property 2(Prop. 2):** the performance improvement in Algorithm 4 compared to Algorithm 2;
- **Property 3(Prop. 3):** the performance improvement in Algorithm 5 compared to Algorithm 2;
- **Property 4(Prop. 4):** the performance improvement in Algorithm 5 compared to Algorithm 4;
- **Property 5(Prop. 5):** the performance improvement in Algorithm 6 compared to Algorithm 2.

The experimental results in Table 1 demonstrate that the proposed improved Coppersmith algorithm, based on block preprocessing, enhances the solving efficiency of the algorithm to a certain extent. Compared to the original Coppersmith algorithm, it achieves a maximum improvement of 8.1% (when $m = 20$) and an average improvement of 5.6%. When combined with the rounding algorithm, the improved Coppersmith algorithm exhibits a maximum improvement of 22.4% (when $m = 10$) and an average improvement of 18.0% over the original algorithm. Furthermore, compared to using the rounding algorithm alone, it achieves a maximum improvement of 12.1% and an average improvement of 7.2%. In comparison to the improved Coppersmith algorithm proposed by Wang et al. in 2021, the average improvement of this proposed algorithm is consistent at 17.1%, even surpassing it in cases with matrix dimensions of 30, 36, and 45.

Table 1. Comparison of experimental results from different algorithms. Average execution time (seconds) for 10 runs of each algorithm.

m	**Alg. 2**	**Alg. 3**	**Alg. 4**	**Alg. 5**	**Alg. 6**	**Prop. 1**	**Prop. 2**	**Prop. 3**	**Prop. 4**	**Prop. 5**
8	8.79	8.16	7.85	7.30	7.17	7.2%	10.7%	17.0%	7.0%	18.4%
9	18.01	17.07	16.06	14.94	14.85	5.2%	10.8%	17.0%	7.0%	17.5%
10	35.67	33.19	30.44	27.68	28.31	7.0%	14.7%	22.4%	9.1%	20.6%
11	65.52	62.35	56.37	50.94	51.16	4.8%	14.0%	22.3%	9.6%	21.9%
12	108.33	102.50	96.05	84.42	87.15	5.4%	11.3%	22.1%	12.1%	19.6%
13	174.74	163.68	156.28	137.74	139.44	6.3%	10.6%	21.2%	11.9%	20.2%
14	279.86	264.58	251.47	227.26	228.10	5.5%	10.1%	18.8%	9.6%	18.5%
15	457.45	420.68	393.85	384.15	399.78	8.0%	13.9%	16.0%	2.5%	12.6%
16	655.06	633.84	572.97	543.31	547.18	3.2%	12.5%	17.1%	5.2%	16.5%
17	976.39	938.30	847.44	821.70	826.47	3.9%	13.2%	15.8%	3.0%	15.4%
18	1452.16	1396.65	1311.77	1173.08	1198.52	3.8%	9.7%	19.2%	10.6%	17.5%
19	1973.83	1897.81	1757.55	1702.86	1709.87	3.9%	11.0%	13.7%	3.1%	13.4%
20	2724.54	2504.61	2457.98	2399.13	2438.76	8.1%	9.8%	11.9%	2.4%	10.5%

The block preprocessing algorithm in our method supports independent execution without data exchange, making it suitable for taking advantage of multiprocessing in scenarios involving a high degree of modular polynomial equations and a large number of blocks. For example, when analyzing the RSA algorithm with a large encryption index, it can be divided into several large dimensional blocks, and each block can be calculated in parallel without affecting each other, which may not only save the solving time of the LLL algorithm but also achieve the purpose of reducing the size of matrix elements, thus reducing the solving time of the Coppersmith algorithm. This provides a new perspective for implementing the parallel computation of the Coppersmith algorithm, facilitates parallel execution, and provides valuable insights.

In addition, during the experimental process, this study also attempted to present two heuristic algorithms:

- **Heuristic Algorithm 1:** In an effort to enhance the Coppersmith algorithm, based on row common factor extraction proposed by Wang et al. in 2021, this study integrates

it with the block preprocessing algorithm. The lattice basis matrix is first subjected to the block preprocessing algorithm, followed by the execution of the rounding algorithm and subsequent row common factor extraction. However, during testing, it was observed that although the combined algorithm successfully obtains accurate results, it does not noticeably reduce the overall solving time of the Coppersmith algorithm. The solving time is nearly equivalent to that of combining the rounding algorithm and the row common factor extraction algorithm. This can be attributed to the fact that the block preprocessing algorithm already reduces the matrix element scale significantly, taking additional steps to further reduce it less impactful.

- **Heuristic Algorithm 2:** We attempted to place the rounding algorithm before the block preprocessing algorithm. By applying an equivalence transformation prior to block preprocessing, accurate results are obtained. However, it does not lead to a reduction in the solving time of the algorithm. The difference lies in Algorithm 5 proposed in this paper, which firstly pretreats each matrix using the block LLL algorithm and reduces each matrix to its simplest form. At this point, the rounding algorithm is used to transform the whole matrix from the lower-right corner to the upper-left corner, potentially achieving a reduction in the size of matrix elements to the greatest extent. On the contrary, if the constructed Coppersmith matrix is rounded, then LLL partitioning is performed. Although this process reduces the element size to a certain extent, it does not reflect the advantage of the block LLL algorithm in reducing the matrix to a simpler one; that is, the overall solving time of the algorithm is increased.

5. Conclusions

The proposed improved Coppersmith algorithm, based on the block preprocessing algorithm, effectively reduces the solving time of the algorithm. It also demonstrates compatibility with the rounding algorithm proposed by Bi et al. in 2014 [22]. This combination further enhances the efficiency of the Coppersmith algorithm. The experimental results show that the combined algorithm almost performs as equally well as the improved algorithm based on row common factor extraction proposed by Wang et al. in 2021 [23]. In fact, in certain dimensions, the algorithm proposed in this study exhibits even superior performance.

In practical applications, as the RSA encryption exponent increases, the degree of the corresponding modular polynomial equations also increases. Consequently, the dimensions of the resulting block matrices also increase. To ensure accurate solutions, it is necessary to choose larger parameters for constructing the lattice basis matrix, which, in turn, leads to a larger number of blocks. In such cases, the computational advantage of utilizing multiple processes in parallel can be harnessed. By parallelizing the proposed improved Coppersmith algorithm, which combines block preprocessing with the rounding algorithm, the solving time of the algorithm can be effectively reduced. Currently, the optimization of the Coppersmith algorithm can be explored in two directions. First, improvements can be made in constructing the lattice basis matrix by enhancing the polynomials used. Second, more-efficient LLL algorithms can be considered as replacements in the reduction process. Many aspects of the algorithm still require further research and investigation.

Author Contributions: Conceptualization, L.Z., B.Q. and W.G.; methodology, L.Z., B.Q. and W.G.; software, L.Z.; validation, B.Q., W.G. and Y.L.; writing—original draft preparation, L.Z.; writing—review and editing, B.Q., W.G. and Y.L.; project administration, B.Q.; funding acquisition, W.G., Y.L. and L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundations of China (grant nos.: 62002288 and 62072207), Xi'an University of Posts and Telecommunications Postgraduate Innovation Fund (grant no.: CXJJYL2022088), and the Guangdong Basic and Applied Basic Research Foundation (grant no.: 2022A1515140090).

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy restrictions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zhou, Y.B.; Jiang, Z.M.; Wang, T.Y.; Yuan, S.M.; Xu, J.; Wang, K.P.; Liu, Y.J. Progress of Lattice-based Cryptanalysis of RSA and Its Variant Algorithms. *J. Softw.* **2022**, *34*, 4310–4335.
2. Nguyen, P.Q. Lattice Reduction Algorithms: Theory and Practice. In Proceedings of the Advances in Cryptology—EUROCRYPT 2011, Tallinn, Estonia, 15–19 May 2011; pp. 2–6.
3. Gama, N.; Nguyen, P.Q. Predicting Lattice Reduction. In Proceedings of the Advances in Cryptology—EUROCRYPT 2008, Istanbul, Turkey, 13–17 April 2008; pp. 31–51.
4. Rivest, R.L.; Shamir, A.; Adleman L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
5. Merkle, R.; Hellman, M. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inf. Theory* **1978**, *24*, 525–530. [CrossRef]
6. Shamir, A. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Trans. Inf. Theory* **1984**, *30*, 699–704. [CrossRef]
7. Lenstra, A.K.; Lenstra, H.W.; Lovász, L. Factoring polynomials with rational coefficients. *Math. Ann.* **1982**, *261*, 515–534. [CrossRef]
8. Adleman, L.M. On breaking generalized knapsack public key cryptosystems. In Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC'83), Boston, MA, USA, 25–27 April 1983; pp. 402–412.
9. Coppersmith, D. Finding a Small Root of a Univariate Modular Equation. In Proceedings of the Advances in Cryptology—EUROCRYPT'96, Zaragoza, Spain, 12–16 May 1996; pp. 155–165.
10. Coppersmith, D. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In Proceedings of the Advances in Cryptology—EUROCRYPT'96, Zaragoza, Spain, 12–16 May 1996; pp. 178–189.
11. Coppersmith D. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. Cryptol.* **1997**, *10*, 233–260. [CrossRef]
12. Howgrave-Graham, N. Finding small roots of univariate modular equations revisited. In Proceedings of the Cryptography and Coding 1997, Cirencester, UK, 17–19 December 1997; pp. 131–142.
13. Bunder, M.; Nitaj, A.; Susilo, W.; Tonien, J. A New Attack on Three Variants of the RSA Cryptosystem. In Proceedings of the Information Security and Privacy—ACISP 2016, Melbourne, Australia, 4–6 July 2016; pp. 258–268.
14. Quisquater, J.J.; Couvreur, C. Fast decipherment algorithm for RSA public-key cryptosystem. *Electron. Lett.* **1982**, *18*, 905–907. [CrossRef]
15. May, A. Using LLL-Reduction for Solving RSA and Factorization Problems. In *The LLL Algorithm: Survey and Applications*; Nguyen, P.Q., Vallée, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 315–348. ISBN 978-3-642-02295-1.
16. Nguyen, P.Q.; Shparlinski, I.E. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *J. Cryptol.* **2002**, *15*, 151–176. [CrossRef]
17. Nguyen, P.Q.; Shparlinski, I.E. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Des. Codes Cryptogr.* **2003**, *30*, 201–217. [CrossRef]
18. Micheli, G.D.; Heninger, N. Recovering cryptographic keys from partial information, by example. *IACR Cryptol. ePrint Arch.* **2020**, 1506.
19. Nitaj, A.; Susilo, W.; Tonien, J. A new attack on some RSA variants. *Theor. Comput. Sci.* **2023**, *960*, 113898. [CrossRef]
20. Meers, J.; Nowakowski, J. Solving the Hidden Number Problem for CSIDH and CSURF via Automated Coppersmith. *IACR Cryptol. ePrint Arch.* **2023**, 1409.
21. Ajani, Y.; Bright, C. A Hybrid SAT and Lattice Reduction Approach for Integer Factorization. In Proceedings of the 8th International Workshop on Satisfiability Checking and Symbolic Computation, Tromsø, Norway, 24–28 July 2023; pp. 39–43.
22. Bi, J.G.; Coron, J.S.; Faugère, J.C.; Nguyen, P.Q.; Renault, G.; Zeitoun, R. Rounding and Chaining LLL: Finding Faster Small Roots of Univariate Polynomial Congruences. In Proceedings of the Public-Key Cryptography—PKC 2014, Buenos Aires, Argentina, 26–28 March 2014; pp. 185–202.
23. Wang, Y.F.; Li, G.S. Improved Coppersmith Algorithm Based on Extraction of Row Common Factor. *J. Inf. Eng. Univ.* **2021**, *22*, 81–86.
24. Wang, X.Y.; Liu, M.J. Survey of Lattice-based Cryptography. *J. Cryptologic Res.* **2014**, *1*, 13–27. [CrossRef]
25. Yu, W.C. *Lattice Reduction Theory and Its Applications to Cipher Design*; Southwest Jiaotong University: Chengdu, China, 2005.
26. Nguyen, P.Q.; Stehlé, D. LLL on the Average. In Proceedings of the Algorithmic Number Theory—ANTS 2006, Berlin, Germany, 23–28 July 2006; pp. 238–256.
27. Coupé, C.; Nguyenhttp, P.; Stern, J. The Effectiveness of Lattice Attacks Against Low-Exponent RSA. In Proceedings of the Public-Key Cryptography—PKC 1999, Kamakura, Japan, 1–3 March 1999; pp. 204–218.