THE COPPERSMITH METHOD: A REVERSE PROOF AND APPLICATIONS

TIAGO OLIVEIRA MARQUES

Abstract. There is no known general efficient method for finding integer roots of polynomials $f(x) \in \mathbb{Z}[x]$ modulo some integer N. This survey presents the Coppersmith method, which is an algorithm capable of finding all small integer solutions, i.e., finding all roots $|x_0| < N^{1/d}$, where d is the degree of the polynomial f. We focus on providing a reverse proof by reconstructing some sufficient conditions required for success. We do so to provide a better motivation for the use of such techniques in the Coppersmith method. The Coppersmith method has important applications, such as breaking RSA with exponent $e = 3$ if part of the original message is known.

Contents

1. Introduction

In 1982, Lenstra, Lenstra, and Lovász presented the LLL algorithm, which finds a reduced basis for a lattice in polynomial time [LLL82]. Furthermore, they demonstrate that this approach can be used to find all integer roots of an integer polynomial $f(x) \in \mathbb{Z}[x]$ in polynomial time, with respect to the input size.

However, at that time, there was no algorithm that could find the roots of a polynomial modulo some number N, when N has an unknown factorization. Berlekamp developed an algorithm that can factor polynomials in one variable modulo primes p, and thus, in particular, also finds their roots [Ber70]. However, this was infeasible if N was hard to factor.

Coppersmith improved on this hard problem by designing a polynomial time algorithm that finds all small integer solutions of such a polynomial $f(x) \in \mathbb{Z}[x]$ modulo N [Cop96]. Here, small solutions x_0 are defined as $|x_0| < N^{1/d}$, where d is the degree of the polynomial f. The idea is to create auxiliary polynomials that

Date: May 5, 2026.

contain the same small roots, and then use the LLL algorithm to find a reduced integer linear combination of such polynomials, where such small roots modulo N must be roots of the new auxiliary polynomial. Then, the previously known methods from [LLL82] can be applied to find all such roots.

There are rigorous barriers that show that within the class of algorithms with Coppersmith-style auxiliary polynomials, this bound of $N^{1/d}$ is tight in the exponent [CHHS16]. Namely, there are no polynomial time algorithms in such a class that can find roots of size $N^{1/d+\varepsilon}$ for $\varepsilon > 0$.

The Coppersmith method was a big breakthrough, especially in cryptography. In the past, the RSA algorithm was widely used, and for efficiency reasons, the exponent $e = 3$ was often used [RSA77]. The hardness of RSA relies on the hardness of factoring, and Coppersmith showed that his method is enough to break RSA with $e = 3$, assuming we know part of the message, by not requiring to factor N to find small roots [Cop97].

The main focus of this paper is on the reverse proof of the Coppersmith method. Our goal is to provide a proof that will increase the reader's motivation for the techniques used for the Coppersmith method, by providing a new perspective.

We focus on the variant proposed by Howgrave–Graham [HG97], instead of the original one by Coppersmith [Cop96], due to simplicity. Howgrave–Graham showed how the algorithms are equivalent, and instead just work on dual lattices.

Howgrave–Graham proved the correctness of his algorithm by successively proving properties of the vectors in its algorithm, ending with the desired roots of the polynomial [HG97]. However, his proof does not fully motivate the choice of polynomials and constants when they were initially presented.

We start from the literature available in 1996, namely the results in [LLL82], to create some sufficient conditions to find the desired roots. Starting from such sufficient conditions, we show how previous results in literature would obtain them, and then we combine these ideas to motivate our choice of vectors and constants in the Coppersmith method.

This contrasts with the proof presented by Howgrave–Graham in [HG97] as, although it is longer, it better explains the motivation for each step taken. Our proof further helps show that his construction is not ad hoc, but essentially determined by some requirements. As a result, this paper provides a structural understanding of the Coppersmith method, which can be useful for understanding similar lattice-based constructions.

In this survey, we start in Section 2 by briefly presenting the necessary background knowledge, namely the LLL algorithm and the algorithm to find integer roots of integer polynomials, as presented in [LLL82]. After that, in Section 3, we present the Coppersmith method and how it can be used to obtain small zeros of monic polynomials modulo N. We prove the correctness of such an algorithm in a reverse engineering fashion in Section 4, by proceeding from the necessary condition we want, as explained above. Then, in Section 5, we present one application of the Coppersmith method, namely as an attack against small exponent RSA if part of the message is known, as presented in [Cop97]. We further present the current parameter choices used for RSA and how it is infeasible to break RSA currently with the Coppersmith method.

2. Background

We start by presenting some background knowledge needed to fully understand the Coppersmith method, without proof. This includes some other algorithms to solve similar problems that were already known in 1996, before the Coppersmith method. Some of these algorithms will help explain the motivation used for the different steps in the Coppersmith method, as explained in Section 4.

Initially, we give a quick notation definition for polynomials.

Definition 2.1. Given a polynomial $f(x) \in \mathbb{Z}[x]$, let $[x^k]f(x)$ denote the coefficient of the term x^k in $f(x)$.

Then, we explain some background on lattices and the LLL algorithm.

Definition 2.2 (Lattice). A lattice Λ is a subset of $\mathbb{R}^k$ such that there is a set of linearly independent vectors $B = \left\{\vec{b}_1, \ldots, \vec{b}_n\right\}$ such that

$$\Lambda \coloneqq \Lambda(B) = \left\{ \sum_{i=1}^{n} c_i \vec{b}_i \colon c_1, \ldots, c_n \in \mathbb{Z} \right\}.$$

We call B a basis of $\Lambda(B)$.

Throughout this paper, we identify a set of vectors B with the matrix with those vectors as columns, allowing us to compute $\det B$, for example, when B is square.

One important property of lattices is their determinant, defined as follows.

Definition 2.3 (Determinant). Given a lattice $\Lambda = \Lambda(B)$ for some basis B with the basis vectors as columns, let the determinant of Λ be

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

This definition of determinant matches $|\det B|$ when B is square, i.e., when B spans $\mathbb{R}^k$, but it is more general as it is also well-defined when the basis of the lattice does not span the whole space.

The LLL algorithm aims to reduce the basis of a lattice, with the goal of finding an exponential approximation to the shortest nonzero vector in the lattice [LLL82]. We explain now the results in more detail.

We start by defining the Gram–Schmidt process, for notation purposes, and the definition of an LLL-reduced basis.

Definition 2.4 (Gram–Schmidt). Given a basis $B = \left(\vec{b}_1, \ldots, \vec{b}_n\right)$, its Gram–Schmidt orthogonalization is $B^* = \left(\vec{b}_1^*, \ldots, \vec{b}_n^*\right)$ where

$$\vec{b}_i^* = \vec{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \vec{b}_j^* \quad \text{where} \quad \mu_{i,j} = \frac{\left\langle \vec{b}_i, \vec{b}_j^* \right\rangle}{\left\| \vec{b}_j^* \right\|_2^2}.$$

In particular, notice that the vector $B^* = \left(\vec{b}_1^*, \ldots, \vec{b}_n^*\right)$ is not necessarily in the lattice, as the coefficients $\{\mu_{i,j}\}$ are not necessarily integers.

Then, we can define an LLL-reduced basis as follows.

Definition 2.5 (LLL reduced). [LLL82, Section 1] Given a basis B and orthogonalized basis B^*, then B is LLL reduced if and only if

(1) $|\mu_{i,j}| \leq \frac{1}{2}$ whenever $i > j$.
(2) $\left|\left|\vec{b}_i^* + \mu_{i,i-1}\vec{b}_{i-1}^*\right|\right|_2^2 \geq \frac{3}{4}\left|\left|\vec{b}_{i-1}^*\right|\right|_2^2$.

LLL reduced bases are important due to the following theorem, which provides an upper bound on the length of the first vector of an LLL reduced basis. Therefore, they are useful to find short vectors in lattices.

THEOREM 2.6. [LLL82, Proposition 1.9] *If B is LLL reduced, then*

$$\left|\left|\vec{b}_1\right|\right|_2 \leq 2^{\frac{n-1}{4}} D(\Lambda(B))^{1/n}.$$

These inequalities work with the ℓ_2 norm, and it will be useful to use a bound between the ℓ_1 and the ℓ_2 norms. This bound is standard and derivable from Cauchy–Schwarz.

THEOREM 2.7. *For any vector $v \in \mathbb{Z}^n$, one has that*

$$||v||_1 \leq \sqrt{n}||v||_2.$$

Lenstra, Lenstra, and Lovász further present a polynomial time algorithm that computes an LLL-reduced basis of a lattice. Such an algorithm became known as the LLL Algorithm, and it plays a fundamental part in the Coppersmith method.

DEFINITION 2.8 (LLL Algorithm). A polynomial time algorithm proposed by Lenstra, Lenstra, and Lovász that, given a basis B, outputs an LLL-reduced basis B' such that $\Lambda(B) = \Lambda(B')$.

One main application of such an algorithm is to factor polynomials $f \in \mathbb{Q}[x]$. Moreover, if we can factor a polynomial, we can also recover all its integer roots.

THEOREM 2.9. [LLL82, Section 3] *There exists a polynomial time algorithm that finds all integer roots of integer polynomials in one variable.*

The intuition of this algorithm is to represent a polynomial $f(x) = \sum_{i=0}^{d} a_i x^i$ as a vector $a = (a_0, \ldots, a_d) \in \mathbb{R}^{d+1}$. By finding a root modulo some prime p, using Hensel's Lemma allows it to lift to a root modulo p^k, for some $k \in \mathbb{Z}$. Then, they can construct a lattice of polynomials sharing that root, and the LLL algorithm provides a short common factor, which must correspond to an integer factor of f.

Furthermore, it was also known how to find the roots of polynomials modulo primes p.

THEOREM 2.10. [Ber70] *There exists a polynomial time algorithm such that for any $f(x) \in \mathbb{Z}[x]$ and prime $p \in \mathbb{N}$, it finds all $x \in \mathbb{Z}/p\mathbb{Z}$ such that $f(x) \equiv 0 \pmod{p}$.*

Therefore, the question of finding roots of an integer polynomial modulo any number $N \in \mathbb{N}$ is a natural extension of this work. The Coppersmith method, presented in the following section, provides a polynomial time algorithm for finding "small" solutions, as defined in Section 3.

3. The Coppersmith Method

In 1996, Coppersmith published a polynomial time algorithm which, given a monic polynomial $f \in \mathbb{Z}[x]$ of degree d and some $\varepsilon > 0$, finds all the roots of $f(x) \equiv 0 \pmod{N}$ with size $|x| \leq N^{1/d-\varepsilon}$ in polynomial time in the size of the input and in ε^{-1} [Cop96, Section 2]. For simplicity, we assume that $\varepsilon \leq 1$ and

$d \geq 2$, as solving linear polynomials is easy. Such an algorithm became known as the Coppersmith method. This was a breakthrough in the problem, building on the previous results presented in Section 2.

In this section, we present a variant of the Coppersmith method proposed by Howgrave–Graham [HG97, Section 2]. He showed that his method has the same complexity and bounds as Coppersmith's, and they both perform the same technique of lattice reductions. However, the difference is that they do such lattice reductions in dual lattices, instead of on the same lattice. We chose to present Howgrave–Graham's variant as it is shorter and simpler. The algorithm is as follows.

Algorithm 1 Coppersmith Method

Input: A monic polynomial $f(x) \in \mathbb{Z}[x]$ given by its coefficients $f(x) = \sum_{i=0}^{d} a_i x^i$ and $N \in \mathbb{N}$.

1: Let $M \leftarrow \lfloor N^{1/d-\varepsilon} \rfloor$.
2: Let $h \leftarrow \lceil \varepsilon^{-1} \rceil$.
3: **for** $v = 0$ **to** $h - 1$ **do**
4: **for** $u = 0$ **to** $d - 1$ **do**
5: Let $q_{u,v}(x) \leftarrow N^{h-1-v} x^u f(x)^v$.
6: Let $b_{u,v} \in \mathbb{Z}^{hd}$ be the vector where $(b_{u,v})_i = [x^{i-1}] q_{u,v}(xM)$ for every $1 \leq i \leq hd$.
7: **end for**
8: **end for**
9: Let $\hat{B} \leftarrow \{b_{u,v} \colon 0 \leq u < d, 0 \leq v < h\}$.
10: Let B be an LLL-reduced basis of $\hat{B}$ using the LLL algorithm [LLL82].
11: Let $\vec{b}_1$ be the first vector in B.
12: Consider the polynomial $r(x) = \sum_{i=1}^{hd} (\vec{b}_1)_i (x/M)^{i-1}$.
13: Find all integer roots of $r(x)$ as in [LLL82], and store them in set $\mathcal{R}$.
14: Let $\mathcal{Z} \leftarrow \{x \in \mathcal{R} \colon f(x) \equiv 0 \pmod{N} \text{ and } |x| \leq M\}$.
15: Return $\mathcal{Z}$.

Succinctly, Algorithm 1 behaves as follows. Initially, in lines 3 through 8, it constructs a basis of polynomials given by $\{q_{u,v}\}_{u,v}$, which can be seen as the basis of a lattice. Then, it uses the LLL algorithm in line 10 to obtain a short lattice vector in line 11. Finally, in lines 12 through 14, it extracts the desired roots. More details and motivation for these steps are presented in the reverse proof in Section 4.

4. A Reverse Proof of the Coppersmith Method

In this section, we show that Algorithm 1 is correct and runs in polynomial time in the input size and ε^{-1}. Our proof is inspired by the one provided by Howgrave–Graham in [HG97, Section 2], but in a different order.

We start in Section 4.1 with a proof of the runtime. Then, in Section 4.2, we construct some sufficient conditions for the correctness of Algorithm 1. Finally, in Section 4.3, we show how to obtain such sufficient conditions.

4.1. Runtime analysis.

We now prove that Algorithm 1 runs in polynomial time.

Lemma 4.1. *Algorithm 1 runs in polynomial time in the input size and ε^{-1}.*

Proof. Let $A = \max_{i=0}^{d} |a_i|$ be the largest coefficient of f. Then, we wish to show that Algorithm 1 runs in $\text{poly}(\log A, \log N, d, \varepsilon^{-1})$.

Notice that $h = \lceil \varepsilon^{-1} \rceil \in \text{poly}(\varepsilon^{-1})$. Hence, the loops run for $hd \in \text{poly}(d, \varepsilon^{-1})$ many times, and we construct a basis with hd many vectors.

We claim that the maximum entry of any basis vector is upper bounded by $(2dAN)^h$. Notice that when we perform the exponentiation $f(x)^v$, as $f(x)$ has at most $d+1$ nonzero coefficients, then each coefficient in $f(x)^v$ is the sum of at most $(d+1)^v \leq 2^v d^v$ many coefficients. However, each coefficient is upper bounded by A, and so the product of v of them is upper bounded by A^v. Hence, any coefficient of $f(x)^v$ is upper bounded by $2^v d^v A^v \leq (2dA)^h$. Therefore, the maximum entry of $N^{h-1-v} f(x)^v x^u$ is at most $(2dAN)^h$.

Thus, the maximum entry can be written with $\log(2dAN)^h = h \log N + h \log A + h \log d + h \in \text{poly}(\log N, \log A, d, \varepsilon^{-1})$ many bits.

Moreover, the LLL algorithm runs in polynomial time in the size of the basis, and therefore $\text{poly}(hd, \log N, \log A, \varepsilon^{-1})$ time.

Finally, finding all integer roots of $r(x)$ runs in polynomial time in its size, which is the same as before.

Hence, Algorithm 1 runs in $\text{poly}(\log A, \log N, d, \varepsilon^{-1})$ time. □

4.2. **Sufficient conditions for correctness.** Now, we start our reverse proof of the correctness of Algorithm 1. This will involve a reverse proof, where we use the intuition of the other existing algorithms for similar problems to derive some sufficient conditions we would like to have, and then we show how to obtain them. In this subsection, we work towards Lemma 4.2, which are going to be our sufficient conditions for Algorithm 1 being correct.

Let $f(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{Z}[x]$ be a monic polynomial and $N \in \mathbb{N}$. Let

$$\mathcal{Z}' = \{x \in \mathbb{Z} \colon f(x) \equiv 0 \pmod N\}$$

be its set of zeros modulo N. We would like to find $\mathcal{Z}'$ in polynomial time. However, as stated in Section 3, the Coppersmith Method cannot find all the roots, but only the small roots. Nevertheless, we write it as our main initial goal, and then we show why we only consider small solutions.

Lenstra, Lenstra, and Lovász have an algorithm that can find the integer roots of any integer polynomial in polynomial time. The existence of such an algorithm motivates constructing some polynomial $\hat{r}(x)$ such that for any $x_0 \in \mathcal{Z}'$, one has that $\hat{r}(x_0) = 0$. However, we are working with polynomials in modulo N, so a possible sufficient condition is for $\hat{r}(x_0) \equiv 0 \pmod N$ and $|\hat{r}(x_0)| < N$.

Idea 1. *We want to find a polynomial $\hat{r}(x) \in \mathbb{Z}[x]$ such that if $x_0 \in \mathcal{Z}'$, then $\hat{r}(x_0) \equiv 0 \pmod N$ and $|\hat{r}(x_0)| < N$.*

Suppose that $\hat{r}(x) = \sum_{i=0}^{k} \hat{b}_i x^i$. Then, one bound on $|\hat{r}(x)|$ is given by

$$|\hat{r}(x)| = \left| \sum_{i=0}^{k} \hat{b}_i x^i \right| \leq \sum_{i=0}^{k} \left| \hat{b}_i x^i \right|.$$

Therefore, this inequality creates the idea of finding all the zeros of $f(x) \equiv 0 \pmod N$ within a certain bound M, to be determined later, as such a bound implies a bound in the previous inequality. Hence, we adapt our goal, and instead of trying to compute $\mathcal{Z}'$, we compute the set

$$\mathcal{Z} = \{x \in \mathbb{Z} \colon |x| \leq M \wedge f(x) \equiv 0 \pmod N\}.$$

IDEA 2. *We want to find a polynomial* $\hat{r}(x) \in \mathbb{Z}[x]$ *such that if* $x_0 \in \mathcal{Z}$, *then* $\hat{r}(x_0) \equiv 0 \pmod{N}$ *and* $\sum_{i=0}^{k} |\hat{b}_i M^i| < N$.

However, Idea 2 is still not enough. We use one further idea from Lenstra, Lenstra, and Lovász, and instead of just analyzing modulo N, we analyze the polynomial modulo larger powers of N. In particular, for some $h \in \mathbb{N}$ determined later, we analyze it modulo N^{h-1}. This provides us with more freedom later.

IDEA 3. *We want to find a polynomial* $\hat{r}(x) \in \mathbb{Z}[x]$ *such that if* $x_0 \in \mathcal{Z}$, *then* $\hat{r}(x_0) \equiv 0 \pmod{N^{h-1}}$ *and* $\sum_{i=0}^{k} |\hat{b}_i M^i| < N^{h-1}$.

We formalize now that Idea 3 is enough to find the small roots of f, and that this is exactly what we do in Algorithm 1.

LEMMA 4.2. *Consider the polynomial* $r(x) = \sum_{i=1}^{hd} \frac{(\vec{b}_1)_i}{M^{i-1}} x^{i-1} \in \mathbb{Q}[x]$ *determined in step* 12 *of Algorithm 1, from the first vector* $\vec{b}_1 \in B$ *of an LLL-reduced basis of* $\hat{B}$. *If*

(1) $r(x) \in \mathbb{Z}[x]$,
(2) for any $x_0 \in \mathcal{Z}$, *one has that* $r(x_0) \equiv 0 \pmod{N^{h-1}}$,
(3) $\left\|\vec{b}_1\right\|_1 = \sum_{i=0}^{hd-1} |(\vec{b}_1)_{i+1}| < N^{h-1}$,

then Algorithm 1 is correct.

Proof. Assume that $r(x)$ satisfies all 3 conditions.

Then, for any $x_0 \in \mathcal{Z}$, we have that $r(x_0) \equiv 0 \pmod{N^{h-1}}$ and

$$|r(x_0)| = \left| \sum_{i=0}^{hd-1} \frac{(\vec{b}_1)_{i+1}}{M^i} x_0^i \right| \leq \sum_{i=0}^{hd-1} \left| \frac{(\vec{b}_1)_{i+1}}{M^i} M^i \right| = \sum_{i=0}^{hd-1} \left| (\vec{b}_1)_{i+1} \right| < N^{h-1}.$$

Since $r(x_0)$ is a multiple of N^{h-1} whose absolute value is strictly smaller than N^{h-1}, then $r(x_0) = 0$. Thus, the integer roots of r include all elements of $\mathcal{Z}$.

However, if we assume condition 1, then r is an integer polynomial and so step 13 of Algorithm 1 efficiently finds all roots of r, and then we can check whether they are in $\mathcal{Z}$ efficiently. Thus, Algorithm 1 is correct. □

4.3. **How to achieve the sufficient conditions.** Finally, in this subsection, we show how Algorithm 1 satisfies the three conditions imposed by Lemma 4.2.

Two main ways of obtaining a new polynomial $f^*(x)$ from $f(x)$ such that for any $x_0 \in \mathcal{Z}$, we also have that $f^*(x_0) \equiv 0 \pmod{N^{h-1}}$ are to multiply the original polynomial by large powers of N, or take large powers of f itself. This gives the idea of defining the polynomials $N^{h-1-v} f(x)^v$ for every $0 \leq v < h$.

Then, if we have a set $\{f_\alpha(x)\}_{\alpha \in \mathcal{A}}$ of polynomials such that $x_0 \in \mathcal{Z}$ satisfies $f_\alpha(x_0) \equiv 0 \pmod{N^{h-1}}$ for every $\alpha \in \mathcal{A}$, then x_0 is also a root modulo N^{h-1} of any integer linear combination of the polynomials. Furthermore, the condition 3 of Lemma 4.2 requires such an integer linear combination to be small in ℓ_1 norm, and therefore we use the LLL algorithm on the lattice generated by the vectors corresponding to each polynomial.

Our initial intuition would be to construct the set

$$\{N^{h-1-v} f(x)^v : 0 \leq v < h\},$$

as these polynomials have roots $\mathcal{Z}$ modulo N^{h-1}, as seen above. However, such a set is still not enough. Intuitively, these polynomials have degree $\deg(N^{h-1-v} f(x)^v) =$

$dv \leq d(h-1)$, and so they are represented in $\mathbb{Z}^{dh-d+1}$. However, there are only h many polynomials. Hence, we obtain a lattice that does not span all the dimensions of $\mathbb{Z}^{dh-d+1}$. In that case, we check that adding more polynomials to our basis, to ensure it indeed spans the whole space, should (in general) decrease the smaller element of the lattice.

Furthermore, Theorem 2.6 provides an inequality relating the norm of the first vector of an LLL basis to the determinant of the basis. As our goal from Lemma 4.2 is to reduce such a norm, we can upper bound it using the determinant. Hence, it would be important if our basis has an easily computable determinant.

There is a possible construction that allows us to achieve both goals described in the last two paragraphs, as presented now. We extend the previous basis to

$$\mathcal{P} \coloneqq \{N^{h-1-v}x^u f(x)^v \colon 0 \leq v < h, 0 \leq u < d\}.$$

Then, the degrees of the polynomials are $\deg(N^{h-1-v}x^u f(x)^v) = vd + u \leq hd - 1$. Thus, they are represented in $\mathbb{Z}^{hd}$, and there are hd many polynomials in $\mathcal{P}$. So, as desired, our basis spans the whole space. In particular, the determinant of this basis will also be easily computed, as shown later in Lemma 4.6, where we show that there is an ordering of polynomials where the basis is upper triangular.

According to such basis $\mathcal{P}$ and Algorithm 1, let $q_{u,v}(x) = N^{h-1-v}x^u f(x)^v$ for $0 \leq v < h, 0 \leq u < d$. Furthermore, we also need to account for the fact that condition (3) of Lemma 4.2 requires that the vector $r(xM)$ has small ℓ_1 norm, and not $r(x)$, and so we use as basis

$$\hat{B} \coloneqq \{q_{u,v}(xM) \colon 0 \leq v < h, 0 \leq u < d\}.$$

Fact 4.3. *The $\hat{B}$ defined above is the same one as in Algorithm 1.*

To illustrate such a construction and the fact that $\hat{B}$ is upper triangular, we provide now a concrete example.

Example 4.4. Consider the polynomial $f(x) = x^2 + a_1 x + a_2 \in \mathbb{Z}[x]$. Here, $d = \deg f = 2$. Suppose that $h = 2$ and fix some $M \in \mathbb{N}$. This provides the polynomials:

- $q_{0,0}(xM) = N^{h-1-0}(xM)^0 f(xM)^0 = N$.
- $q_{1,0}(xM) = N^{h-1-0}(xM)^1 f(xM)^0 = NMx$.
- $q_{0,1}(xM) = N^{h-1-1}(xM)^0 f(xM)^1 = f(xM) = M^2x^2 + a_1 Mx + a_2$.
- $q_{1,1}(xM) = N^{h-1-1}(xM)^1 f(xM)^1 = xMf(xM) = M^3x^3 + a_1 M^2 x^2 + a_2 Mx$.

If we order the polynomials as $q_{0,0}, q_{1,0}, q_{0,1}, q_{1,1}$, as presented above, they are represented in $\mathbb{Z}^{2\times 2} = \mathbb{Z}^4$. Then, the basis consists of such polynomials in each column, i.e.,

$$\hat{B} = \begin{pmatrix} N & 0 & a_2 & 0 \\ 0 & NM & a_1 M & a_2 M \\ 0 & 0 & M^2 & a_1 M^2 \\ 0 & 0 & 0 & M^3 \end{pmatrix}.$$

In particular, we check that $\hat{B}$ is upper triangular, and its determinant is easily computed as the product of the diagonal entries.

We can furthermore define B to be the LLL-reduced basis from $\hat{B}$ returned by the LLL algorithm, and $\vec{b}_1$ to be its first vector, as in Algorithm 1.

Using this construction of $\hat{B}$ and B, we have enough tools to show conditions (1) and (2) of Lemma 4.2.

LEMMA 4.5. *Consider the polynomial* $r(x) = \sum_{i=1}^{hd} \frac{(\vec{b}_1)_i}{M^{i-1}} x^{i-1} \in \mathbb{Q}[x]$ *determined in step* 12. *Then,*

(1) $r(x) \in \mathbb{Z}[x]$,
(2) for any $x_0 \in \mathcal{Z}$, *one has that* $r(x_0) \equiv 0 \pmod{N^{h-1}}$.

Proof. Consider the polynomial $t(x) = r(xM) = \sum_{i=0}^{hd-1} (\vec{b}_1)_{i+1} x^i$. Then, $t(x)$ corresponds to the first vector $\vec{b}_1$ in B, which is an LLL-reduced basis of $\hat{B}$. Furthermore, $r(x) = t(x/M)$.

In particular, $\vec{b}_1$ is in the lattice generated by $\hat{B}$, and therefore there are integer coefficients $c_{u,v}$ for $0 \le v < h, 0 \le u < d$ such that

$$t(x) = \sum_{v=0}^{h-1} \sum_{u=0}^{d-1} c_{u,v} q_{u,v}(xM).$$

Hence,

$$r(x) = t(x/M) = \sum_{v=0}^{h-1} \sum_{u=0}^{d-1} c_{u,v} q_{u,v}(x) \in \mathbb{Z}[x],$$

proving the first condition.

Then, for any $x \in \mathcal{Z}$, we have that $f(x) \equiv 0 \pmod{N}$. Hence, $q_{u,v}(x_0) = N^{h-1-v} x_0^u f(x_0)^v \equiv 0 \pmod{N^{h-1}}$ for any $0 \le v < h, 0 \le u < d$. Thus,

$$r(x_0) = \sum_{v=0}^{h-1} \sum_{u=0}^{d-1} c_{u,v} q_{u,v}(x_0) \equiv 0 \pmod{N^{h-1}}.$$

□

Therefore, we only need to show condition 3 of Lemma 4.2 to prove the correctness of Algorithm 1. To do so, we invoke Theorem 2.6 to give an upper bound on the length of the first vector returned by the LLL algorithm, by first bounding the determinant of $\hat{B}$. This is an easy computation because $\hat{B}$ was made upper triangular.

LEMMA 4.6. *The determinant of* $\Lambda(\hat{B})$ *satisfies*

$$D(\Lambda(\hat{B})) = |\det \hat{B}| = M^{\frac{hd(hd-1)}{2}} N^{\frac{hd(h-1)}{2}}.$$

Proof. Notice that $\hat{B}$ will be a square matrix with the polynomials as columns, as we have hd many vectors in $\mathbb{Z}^{hd}$. So, $D(\Lambda(\hat{B})) = |\det \hat{B}|$. Furthermore, swapping the order of the vectors either maintains the determinant or changes the sign, so we can pick any ordering.

Suppose we order the columns in $\hat{B}$ as $(u_1, v_1) \prec (u_2, v_2)$ if and only if $v_1 < v_2$, or $v_1 = v_2$ and $u_1 < u_2$. We claim that $\hat{B}$ is upper triangular.

Let $1 \le j < i \le hd$, and we analyze $\hat{B}_{i,j}$. This corresponds to $v = \lfloor \frac{j-1}{d} \rfloor$ and $u = j - 1 - dv$. Then,

$$\deg q_{u,v} = \deg N^{h-1-v}(xM)^u f(xM)^v = u + v \deg f = j - 1 - dv + dv = j - 1.$$

Thus, the entries $\hat{B}_{i,j}$ are zero whenever $i > j$, and so $\hat{B}$ is upper triangular.

In particular, its determinant is the product of diagonal terms, which are the leading coefficients in each polynomial. Thus, as the leading term of $q_{u,v}(xM) = N^{h-1-v}(xM)^u f(xM)^v$ is $N^{h-1-v}M^u M^{vd}$ as f is monic, then

$$\begin{aligned}\left|\det \hat{B}\right| &= \prod_{u=0}^{d-1}\prod_{v=0}^{h-1} N^{h-1-v}M^u M^{vd} \\ &= \left(\prod_{v=0}^{h-1} N^{h-1-v}\right)^d \left(\prod_{u=0}^{d-1} M^u\right)^h \left(\prod_{v=0}^{h-1} M^{vd}\right)^d \\ &= \left(N^{\sum_{v=0}^{h-1} v}\right)^d \left(M^{\sum_{u=0}^{d-1} u}\right)^h \left(M^{d\sum_{v=0}^{h-1} v}\right)^d \\ &= N^{\frac{dh(h-1)}{2}} M^{\frac{hd(d-1)}{2}} M^{\frac{hd^2(h-1)}{2}} \\ &= N^{\frac{hd(h-1)}{2}} M^{\frac{hd(hd-1)}{2}}.\end{aligned}$$

□

Then, we obtain an upper bound on the ℓ_1 norm of the returned polynomial.

COROLLARY 4.7. *We have that*

$$\left|\left|\vec{b}_1\right|\right|_1 \leq \sqrt{hd}\cdot 2^{\frac{hd-1}{4}} M^{\frac{hd-1}{2}} N^{\frac{h-1}{2}}.$$

Proof. We have that

$$\left|\left|\vec{b}_1\right|\right|_1 \leq \sqrt{hd}\left|\left|\vec{b}_1\right|\right|_2 \leq \sqrt{hd}\cdot 2^{\frac{hd-1}{4}} D(\Lambda(\hat{B}))^{\frac{1}{hd}} = \sqrt{hd}\cdot 2^{\frac{hd-1}{4}} M^{\frac{hd-1}{2}} N^{\frac{h-1}{2}},$$

where the first inequality follows from Theorem 2.7, the second inequality follows from Theorem 2.6 and the last equality follows from Lemma 4.6. □

Therefore, we now need to pick some value of M and h such that

$$\sqrt{hd}\cdot 2^{\frac{hd-1}{4}} M^{\frac{hd-1}{2}} N^{\frac{h-1}{2}} < N^{h-1}.$$

As M is an integer, the largest M that satisfies the previous inequality is

$$M = \left\lfloor 2^{-\frac{1}{2}}(hd)^{-\frac{1}{hd-1}} N^{\frac{h-1}{hd-1}} \right\rfloor.$$

Notice that as $h \to \infty$, then $\lim_{h\to\infty}(hd)^{-\frac{1}{hd-1}} = 1$, and so as $h \to \infty$, we have that

$$\lim_{h\to\infty} M = 2^{-\frac{1}{2}} N^{\frac{1}{d}}.$$

In particular, if we want $M \geq N^{1/d-\varepsilon}$, it is enough that

$$\frac{h-1}{hd-1} > \frac{1}{d} - \varepsilon \Leftrightarrow h \geq \frac{1}{\varepsilon d} + \frac{1}{d} - \frac{1}{\varepsilon d^2}.$$

So, the choice $h \geq \varepsilon^{-1}$ works, while assuming $d \geq 2$ and $\varepsilon \leq 1$.

Therefore, the following lemma holds.

LEMMA 4.8. *The choice* $M = \left\lfloor N^{1/d-\varepsilon} \right\rfloor$ *and* $h = \left\lceil \varepsilon^{-1} \right\rceil$ *is enough to satisfy condition (3) of Lemma 4.2.*

Thus, we have that all 3 conditions of Lemma 4.2 hold, and therefore Algorithm 1 is correct.

5. Applications to RSA

In this section, we present how the Coppersmith method can be used to break RSA with a small exponent e if we know part of the original message.

Suppose we have a secret message $m + x$, where m is some known part and x is unknown. Notice that this includes the case where nothing is known, and then $m = 0$.

Suppose the message is encrypted using RSA with primes p and q, and let $N = pq$. The encryption corresponds to the ciphertext

$$c = (m + x)^e \bmod N.$$

The details of RSA encryption can be found in [RSA77].

There is no known polynomial algorithm that breaks RSA. However, if we know how to factor N in polynomial time, that is enough to break RSA.

In particular, the RSA encryption forms a simple polynomial equation, namely

$$(m + x)^e - c \equiv 0 \pmod{N}.$$

The strength of the Coppersmith method is that it allows us to solve the polynomial equation even without factoring N, which remains hard.

However, there is a fallback. The Coppersmith method only allows us to find small solutions, so we can only find x if $|x| \leq N^{1/e}$, as the degree of the desired polynomial is $d = e$. But if that is the case, then the encryption scheme is broken.

In particular, the exponent $e = 3$ was used widely in the past, and in that case, we are trying to solve the monic polynomial equation

$$f(x) \coloneqq (m + x)^3 - c = x^3 + 3mx^2 + 3m^2x + m^3 - c \equiv 0 \pmod{N}$$

with degree $\deg f = d = 3$. If we are only missing $\log_2(N^{1/3}) = \frac{1}{3}\log_2 N$ bits of the message in consecutive positions, we can let those missing bits be x, and we can recover them by solving such an equation as the solution x satisfies $|x| \leq N^{1/e} = N^{1/3}$.

To better understand the feasibility of such a bound, we need to look at the historical standards for N and e. Historically, the suggested values of N have been increasing as our computational power increases. Arjen Lenstra and Eric R. Verheul published a timeline of different years (in the past and future) and suggested bit size of N in [LV01, Table 1]. In particular, starting at 1982, the suggestion is 417 bits, in 2000 the suggestion is 952 bits, in 2010 the suggestion is 1369 bits, and in 2020 the suggestion is 1881 bits.

However, the exponent e is normally desired to be large enough to avoid possible attacks (such as the Coppersmith method), but not too large to increase the computational runtime of encrypting and decrypting messages. The usage of $e = 2^{16} + 1$ is widespread as $2^{16} + 1$ is a prime number with only two ones in the binary expansion, making its exponentiation quick to compute.

Nowadays, NIST indeed suggests using an odd exponent e such that $65537 = 2^{16} + 1 \leq e < 2^{256}$ and that N should have 2048 bits for 112 security strength, according to [BCR+19, Section 6.2.1] and [Bar20, Table 2].

With these standards of security, the fraction $\frac{1}{e} \leq \frac{1}{65537}$ is sufficiently small, making this use of the Coppersmith method to decrypt the message unrealistic.

Therefore, even though the Coppersmith method might be viable to break RSA with a small exponent, namely $e = 3$, it is no longer a security threat when large exponents are used.

References

[Bar20] Elaine B. Barker. Recommendation for key management: Part 1 – general. NIST Special Publication (SP) 800-57 Part 1 Rev. 5, National Institute of Standards and Technology, May 2020.

[BCR+19] Elaine B. Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, Richard Davis, and Simon Scott. Recommendation for pair-wise key-establishment schemes using integer factorization cryptography. NIST Special Publication (SP) 800-56B Rev. 2, National Institute of Standards and Technology, March 2019.

[Ber70] Elwyn R. Berlekamp. Factoring polynomials over large finite fields. In *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, SYMSAC '71, page 223, New York, NY, USA, 1970. Association for Computing Machinery.

[CHHS16] Ted Chinburg, Brett Hemenway, Nadia Heninger, and Zachary Scherr. Cryptographic applications of capacity theory: On the optimality of coppersmith's method for univariate polynomials. *CoRR*, abs/1605.08065, 2016.

[Cop96] Don Coppersmith. Finding a small root of a univariate modular equation. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 155–165, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Cop97] Don Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, 10(4):233–260, September 1997.

[HG97] Nicholas Howgrave-Graham. Finding small roots of univariate modular equations revisited. In Michael Darnell, editor, *Cryptography and Coding*, pages 131–142, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.

[LV01] Arjen Lenstra and Eric Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14:255–293, 12 2001.

[RSA77] Ronald L. Rivest, Adi. Shamir, and Leonard M. Adleman. *A method for obtaining digital signatures and public-key cryptosystems.* MIT/LCS/TM-82. Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, rev. edition, 1977.

Massachusetts Institute of Technology
Email address: tiago13@mit.edu