Rounding and Chaining LLL: Finding Faster Small Roots of Univariate Polynomial Congruences

Jingguo Bi[1], Jean-Sébastien Coron[2], Jean-Charles Faugère[3,4,5], Phong Q. Nguyen[6,1], Guénaël Renault[4,3,5], and Rina Zeitoun[7,4,3,5]

[1] Tsinghua University, Institute for Advanced Study, Beijing 100084, China
jingguobi@mail.tsinghua.edu.cn
[2] University of Luxembourg
jean-sebastien.coron@uni.lu
[3] INRIA, POLSYS, Centre Paris-Rocquencourt, F-78153, Le Chesnay, France
[4] Sorbonne Universités, UPMC Univ Paris 06, Équipe POLSYS, LIP6 UPMC, F-75005, Paris, France
[5] CNRS, UMR 7606, LIP6 UPMC, F-75005, Paris, France
jean-charles.faugere@inria.fr
guenael.renault@lip6.fr
[6] INRIA, France
http://www.di.ens.fr/~pnguyen
[7] Oberthur Technologies, 420 rue d'Estienne d'Orves, CS 40008, 92705 Colombes, France
r.zeitoun@oberthur.com

Abstract. In a seminal work at EUROCRYPT '96, Coppersmith showed how to find all small roots of a univariate polynomial congruence in polynomial time: this has found many applications in public-key cryptanalysis and in a few security proofs. However, the running time of the algorithm is a high-degree polynomial, which limits experiments: the bottleneck is an LLL reduction of a high-dimensional matrix with extra-large coefficients. We present in this paper the first significant speedups over Coppersmith's algorithm. The first speedup is based on a special property of the matrices used by Coppersmith's algorithm, which allows us to provably speed up the LLL reduction by rounding, and which can also be used to improve the complexity analysis of Coppersmith's original algorithm. The exact speedup depends on the LLL algorithm used: for instance, the speedup is asymptotically quadratic in the bit-size of the small-root bound if one uses the Nguyen-Stehlé L^2 algorithm. The second speedup is heuristic and applies whenever one wants to enlarge the root size of Coppersmith's algorithm by exhaustive search. Instead of performing several LLL reductions independently, we exploit hidden relationships between these matrices so that the LLL reductions can be somewhat chained to decrease the global running time. When both speedups are combined, the new algorithm is in practice hundreds of times faster for typical parameters.

Keywords: Coppersmith's Algorithm, Small Roots of Polynomial Equations, LLL, Complexity, Speedup, RSA.

During the preparation of this paper, J. Bi and P. Q. Nguyen were supported in part by NSFC's Key Project Grant 61133013, China's 973 Program, Grant 2013CB834205, and J. Bi was also supported by NSFC Grant 61272035 and China Postdoctoral Science Foundation Grant 2013M542417. Part of this work was also supported by the HPAC grant (ANR-11-BS02-013) and by the EXACTA grant (ANR-09-BLAN-0371-01) of the French National Research Agency.

1 Introduction

At EUROCRYPT '96, Coppersmith [7, 6, 8] showed how to find efficiently all small roots of polynomial equations (modulo an integer, or over the integers). The simplest (and perhaps most popular) result is the following: Given an integer N of unknown factorization and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ, Coppersmith's lattice-based algorithm finds all integers $x_0 \in \mathbb{Z}$ such that $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq N^{1/\delta}$ in time polynomial in $\log N$ and δ. This has many applications in public-key cryptanalysis (*e.g.* attacking special cases of RSA and factoring with a hint), but also in a few security proofs (such as in RSA-OAEP [22]). Accordingly, Coppersmith's seminal work has been followed up by dozens of articles (see May's survey [14] for references), which introduced new variants, generalizations, simplifications and applications.

All these small-root algorithms are based on the same idea of finding new polynomial equations using lattice basis reduction: it reduces the problem of finding small roots to finding LLL-short vectors in a lattice. This can theoretically be done in polynomial time using the LLL algorithm [13], but is by no means trivial in practice: the asymptotic running time is a high-degree polynomial, because the lattice is huge. More precisely, May's recent survey [14] gives for Coppersmith's lattice-based algorithm the complexity upper bound $O(\delta^5 \log^9 N)$ using the Nguyen-Stehlé L^2 algorithm [18] as the reduction algorithm. A careful look gives a slightly better upper bound: asymptotically, one may take a matrix of dimension $O(\log N)$, and bit-size $O((\log^2 N)/\delta)$, resulting in a complexity upper bound $O((\log^9 N)/\delta^2)$ using L^2. In typical applications, δ is small ≤ 9 but $\log N$ is the bit-size of an RSA modulus, *i.e.* at least 1024 bits, which makes the theoretical running time daunting: $\log^9 N$ is already at least 2^{90}. For more powerful variants of Coppersmith's algorithm, the running time is even worse, because the lattice dimension and/or the bit-size increase: for instance, Coron [9] gives the upper bound $O(\log^{11} W)$ for finding small roots over bivariate equations over the integers (W plays a role similar to N in the univariate congruence case), using L^2.

The bottleneck of all Coppersmith-type small-root algorithms is the LLL reduction. Despite considerable attention, no significant improvement on the running time has been found, except that LLL algorithms have improved since [8], with the appearance of L^2 [18] and $\tilde{L}^1$ [20]. And this issue is reflected in experiments (see [10]): in practice, one settles for sub-optimal parameters, which means that one can only find small roots up to a bound lower than the asymptotic bound. To illustrate this point, the celebrated Boneh-Durfee attack [1] on RSA with short secret exponent has the theoretical bound $d \leq N^{1-1/\sqrt{2}} \approx N^{0.292}$, but

the largest d in the Boneh-Durfee experiments is only $d \approx N^{0.280}$ with a 1000-bit N, and much less for larger N, *e.g.* $d \approx N^{0.265}$ for 4000-bit N.

OUR RESULTS. We present two speedups over Coppersmith's algorithm for finding small roots of univariate polynomial congruences, which can be combined in practice.

The first speedup is provable and depends on the LLL algorithm used: if one uses L^2 [18], the total bit-complexity is upper bounded by $O(\log^7 N)$, which gives a speedup $\Theta((\log^2 N)/\delta^2)$ quadratic in the bit-size of the small-root bound $N^{1/\delta}$; and if one uses $\tilde{L}^1$, the total complexity is upper bounded by $O(\log^{6+\varepsilon} N)$ for any $\varepsilon > 0$ using fast integer arithmetic, which gives a speedup $O((\log N)/\delta)$ linear in the bit-size of the small-root bound $N^{1/\delta}$. This speedup comes from combining LLL reduction with rounding: instead of LLL-reducing directly a matrix with huge entries, we suitably round the coefficients before LLL reduction to make them much smaller, and show that the LLL output allows to derive sufficiently short vectors in the original lattice. In practice, this means that for any instantiation of Coppersmith's algorithm achieving a small-root bound X, we can drastically reduce the size of the coefficients of the matrix to be LLL-reduced and achieve essentially the same small-root bound: asymptotically, the bit-size is reduced by a factor $(\log N)/\delta$, which implies that the speedup is quadratic when using the popular L^2 algorithm, or quasi-linear using the more theoretical $\tilde{L}^1$ algorithm. This rounding strategy is very natural, but it is folklore that it fails in the worst case: when an arbitrary non-singular matrix is rounded, it may even become singular, and the situation is worse for LLL reduction. However, we show that a well-chosen rounding strategy surprisingly works for the special matrices used by Coppersmith's algorithm: this is because the matrices to be reduced are triangular matrices whose diagonal entries are reasonably balanced, which can be exploited. Interestingly, this peculiar property can also be used to improve the complexity upper bound of Coppersmith's original algorithm, without changing the algorithm: if one uses $\tilde{L}^1$ [20], one can obtain the same complexity upper bound as in our rounding-based algorithm, up to constants.

Our second speedup is heuristic and applies whenever one wants to enlarge the root size X of Coppersmith's algorithm by exhaustive search: it is well-known that any root size X can be extended to mX by applying m times the algorithm on "shifted" polynomials. This enlargement is necessary when one wants to go beyond Coppersmith's bound $N^{1/\delta}$, but it is also useful to optimize the running time below $N^{1/\delta}$: beyond a certain root size below $N^{1/\delta}$, it is folklore that it is faster to use exhaustive search than Coppersmith's algorithm with larger parameters. In this setting, one applies Coppersmith's algorithm with the same modulus N but different polynomials which are all "shifts" of the initial

polynomial $f(x)$: $f_t(x) = f(X \cdot t + x)$ for varying t, where $0 \leqslant t < N^{1/\delta}/X$. We show that this creates hidden relationships between the matrices to be LLL reduced, which can be exploited in practice: instead of performing LLL reductions independently of say, matrices B_1 and B_2, we chain the LLL reductions. More precisely, after LLL reducing B_1 into a reduced basis C_1, we reduce a matrix of the form $C_1 \times P$ for some well-chosen matrix P, instead of the matrix B_2. And this process can be iterated to drastically reduce the global running time.

When both speedups are combined, the new algorithm is in practice hundreds of times faster for typical parameters. Finally, our work helps to clarify the asymptotic complexity of Coppersmith's algorithm for univariate polynomial congruences. Despite the importance of the algorithm, it seems that the dependence on the polynomial degree δ was not well-understood: as previously mentioned, May's survey [14] gave an upper bound including a factor δ^5, and Coppersmith's journal article [8] gave an upper bound growing exponentially in δ. Our final complexity upper bound is independent of δ: it only depends on the bit-size of the modulus N.

Surprisingly, our improvements only apply for now to Coppersmith's algorithm for finding all small roots of polynomial univariate equations, and not to more sophisticated variants such as the gcd generalization used for factoring with a hint. This seems to be the first significant difference between Coppersmith's algorithm and its gcd generalization. It is an interesting open problem to obtain significant speedup for other small-root algorithms.

RELATED WORK. Our first speedup is based on rounding. Rounding has been used in lattice reduction before: for instance, Buchmann [2] used rounding to rigorously estimate when a computation with real lattices can be alternatively performed using integer bases; and the $\tilde{L}^1$ [20] algorithm is also based on rounding. However, it seems that none of the previous work identified the special structure of matrices which we exploit. Our second speedup is based on chaining. Chaining has also been used in lattice reduction before, *e.g.* in the MIMO context [15], but our technique and analysis seem to be a bit different. Thus, both rounding and chaining are folklore strategies, but our work seems to be their first application to Coppersmith's algorithm.

ROADMAP. In Sect. 2, we recall background on lattices and Coppersmith's small-root algorithm. In Sect. 3, we present and analyze our first speedup of Coppersmith's algorithm: rounding LLL. In Sect. 4, we present and analyze our second speedup of Coppersmith's algorithm: chaining LLL. In Sect. 5, we present experimental results with both speedups. Finally, we discuss the case of other small-root algorithms in Sect. 6.

We refer the reader to the full Eprint version of this paper for further details, especially for all missing proofs.

2 Background and Notation

We use row representation for matrices: vectors are row vectors denoted by bold lowercase letters, matrices are denoted by uppercase letters, and their coefficients are denoted by lowercase letters. All logarithms are in base 2. Let $\|\|$ and $\langle , \rangle$ be the Euclidean norm and inner product of $\mathbb{R}^n$. The Euclidean norm is naturally extended to polynomials as follows: if $f(x) = \sum_{i=0}^{n} f_i x^i \in \mathbb{R}[x]$, then $\|f\| = (\sum_{0 \le i \le n} f_i^2)^{1/2}$. We use the following matrix norms: if $M = (m_{i,j})$ is an $n \times m$ matrix, then $\|M\|_2 = \max_{\|\mathbf{x}\| \neq 0} \frac{\|\mathbf{x}M\|}{\|\mathbf{x}\|}$, and $\|M\|_\infty = \max_{1 \le j \le m} \sum_{i=1}^{n} |m_{i,j}|$. Then: $\|M\|_2 \le \sqrt{n}\|M\|_\infty$. If $x \in \mathbb{R}$, we denote by $\lceil x \rfloor$ a closest integer to x.

2.1 Lattices

LATTICES. A lattice L is a discrete subgroup of $\mathbb{R}^m$: there exist $n (\le m)$ linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ s.t. that L is the set $\mathscr{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of all integral linear combinations of the $\mathbf{b}_i$'s. Here, we mostly consider full-rank lattices, *i.e.* $n = m$. The (co-)volume of L is $\mathrm{vol}(L) = \sqrt{\det(BB^t)}$ for any basis B of L, where B^t denotes B's transpose. If B is square, then $\mathrm{vol}(L) = |\det B|$, and if B is further triangular, then $\mathrm{vol}(L)$ is simply the product of the diagonal entries of B in absolute value.

GRAM-SCHMIDT ORTHOGONALIZATION. Let $\mathbf{b}_1, \cdots, \mathbf{b}_n \in \mathbb{R}^m$ be linearly independent vectors. The Gram-Schmidt orthogonalization is the family $(\mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star)$ defined recursively as: $\mathbf{b}_1^\star = \mathbf{b}_1$ and for $i \ge 2$, $\mathbf{b}_i^\star$ is the component of the vector $\mathbf{b}_i$ which is orthogonal to the linear span of $\mathbf{b}_1, \cdots, \mathbf{b}_{i-1}$. Then $\mathbf{b}_i^\star = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^\star$, where $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^\star \rangle / \|\mathbf{b}_j^\star\|^2$ for $1 \le j < i \le n$.

SIZE-REDUCTION. A basis $B = (\mathbf{b}_1, \cdots, \mathbf{b}_n)$ is *size-reduced* if its Gram-Schmidt orthogonalization satisfies $|\mu_{i,j}| \le 1/2$, for all $1 \le j < i \le n$. There is a classical (elementary) algorithm which size-reduces a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of an integer lattice $L \subseteq \mathbb{Z}^m$, in polynomial time, without ever modifying the Gram-Schmidt vectors $\mathbf{b}_i^\star$: this algorithm is included in the original LLL algorithm [13] (e.g. it is the sub-algorithm `RED` in the description of LLL in [4]). In the special case that the input basis is (square) lower-triangular, the running-time of this size-reduction algorithm is $O(n^3 b^2)$ without fast integer arithmetic, and $n^3 \tilde{O}(b)$ using fast-integer arithmetic, where $b = \max_{1 \le i \le n} \log \|\mathbf{b}_i\|$.

LLL AND SHORT LATTICE VECTORS. Coppersmith's small-root method requires the ability to efficiently find reasonably short vectors in a lattice. This

can be achieved by the celebrated LLL algorithm [13] which outputs a non-zero $\mathbf{v} \in L$ s.t. $\|\mathbf{v}\| \leq 2^{\frac{n-1}{4}} \mathrm{vol}(L)^{1/n}$. Nguyen and Stehlé [18] introduced the L^2 algorithm, a faster variant of LLL which can output similarly short vectors in time $O(n^4 m(n+b)b)$ without fast integer arithmetic. The recent $\tilde{L}^1$ algorithm by Novocin *et al.* [20] can output similarly short vectors for a full-rank lattice in time $O(n^{5+\varepsilon}b + n^{\omega+1+\varepsilon}b^{1+\varepsilon})$ for any $\varepsilon > 0$ using fast integer arithmetic, where $\omega \leq 2.376$ is the matrix multiplication complexity constant. However, this algorithm is considered to be mostly of theoretical interest for now: $\tilde{L}^1$ is currently not implemented anywhere, as opposed to L^2. When assessing the complexity of LLL reduction, it is therefore meaningful to mention two complexities: one (closer to the real world) using L^2 without fast integer arithmetic, and another using $\tilde{L}^1$ using fast integer arithmetic and fast linear algebra.

The complexity upper bound of LLL reduction can sometimes be decreased by some polynomial factor. In particular, when the Gram-Schmidt norms of the input basis are balanced, the LLL algorithm requires fewer loop iterations than in the worst case. More precisely, [11, Th. 1.1] showed that the classical upper bound $O(n^2 b)$ on the number of iterations can be replaced by $O\left(n^2 \log \frac{max\|\mathbf{b}_i^\star\|}{min\|\mathbf{b}_i^\star\|}\right)$.

2.2 Coppersmith's method for finding small roots

At EUROCRYPT '96, Coppersmith [7, 6, 8] showed how to find efficiently all small roots of polynomial equations (modulo an integer, or multivariate over the integers), which is surveyed in [14, 16]. We now review the simplest result, following the classical Howgrave-Graham approach [12]: In Sect. 6, we will discuss the main variants of this result.

Theorem 1 (Coppersmith [7, 8]). *There is an algorithm which, given as input an integer N of unknown factorization and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ and coefficients in $\{0, \ldots, N-1\}$, outputs all integers $x_0 \in \mathbb{Z}$ such that $f(x_0) \equiv 0 \,(mod\, N)$ and $|x_0| \leq N^{1/\delta}$ in time polynomial in $\log N$ and δ.*

In all the paper, we consider polynomials verifying $2 < \delta + 1 < (\log N)/2$ since other cases are trivial. Furthermore, Coppersmith's algorithm does not directly achieve the bound $N^{1/\delta}$: indeed, it finds efficiently all roots up to some bound X ($< N^{1/\delta}$) depending on an integer parameter $h \geq 2$, chosen asymptotically to be $h = O((\log N)/\delta)$. When h is sufficiently large, then X becomes sufficiently close to $N^{1/\delta}$ so that one can find all roots up to $N^{1/\delta}$. However, it is well-known that the bound $X = N^{1/\delta}$ should not be reached by taking such a large h. Instead, it is faster to use a smaller h, and perform exhaustive search on the most significant bits of the solutions (see Section 4 for more details).

We now explain Coppersmith's algorithm. The core idea consists in reducing the problem to solving univariate polynomial equations over the integers, by

transforming modular roots into integral roots. More precisely, it constructs a polynomial $g(x) \in \mathbb{Z}[x]$ such that: if $x_0 \in \mathbb{Z}$ is such that $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq X$, then $g(x_0) = 0$ and can be solved easily over $\mathbb{Z}$. To do so, it uses the following elementary criterion:

Lemma 1 (Howgrave-Graham [12]). *Let $g(x) \in \mathbb{Z}[x]$ be a polynomial with at most n non-zero coefficients. Let M be an integer ≥ 1. Assume that $\|g(xX)\| < \frac{M}{\sqrt{n}}$ for some $X \in \mathbb{R}$. If $x_0 \in \mathbb{Z}$ is such that $g(x_0) \equiv 0 \,(mod\, M)$ and $|x_0| \leq X$, then $g(x_0) = 0$.*

Lemma 1 will be used with $M = N^{h-1}$ and $g(x)$ found by lattice reduction. Let $h \geq 2$ be an integer and define the following family of $n = h\delta$ polynomials:

$$g_{i,j}(x) = (x)^j N^{h-1-i} f^i(x) \quad 0 \leq i < h, 0 \leq j < \delta \tag{1}$$

These n polynomials satisfy: if $f(x_0) \equiv 0 \pmod{N}$ for some $x_0 \in \mathbb{Z}$, then $g_{i,j}(x_0) \equiv 0 \pmod{N^{h-1}}$. In order to apply Lemma 1 for a bound $X \geq 1$ to be determined later, Coppersmith's algorithm constructs the n-dimensional lattice L spanned by the rows of the $n \times n$ matrix B formed by the n coefficient vectors of $g_{i,j}(xX)$, where the polynomials are ordered by increasing degree (*e.g.* in the order $(i,j) = (0,0),(0,1),\cdots,(0,\delta-1),(1,0),\cdots(h-1,\delta-1)$) and the coefficients are ordered by increasing monomial degree: the first coefficient is thus the constant term of the polynomial. The matrix B is lower triangular, and its n diagonal entries are:

$$\left(N^{h-1}, N^{h-1}X, \ldots, N^{h-1}X^{\delta-1}, \ldots, N^0 X^{\delta h-\delta}, \ldots, N^0 X^{\delta h-2}, N^0 X^{\delta h-1}\right), \tag{2}$$

because $f(x)$ is monic. In other words, the exponent of X increases by one at each row, while the exponent of N decreases by one every δ rows. It follows that $\mathrm{vol}(L) = \det(B) = N^{\frac{1}{2}n(h-1)} X^{\frac{1}{2}n(n-1)}$. The LLL algorithm is applied to the matrix B, which provides a non-zero polynomial $v(x) \in \mathbb{Z}[x]$ such that $\|v(xX)\| \leq 2^{\frac{n-1}{4}} \mathrm{vol}(L)^{\frac{1}{n}} = 2^{\frac{n-1}{4}} N^{\frac{h-1}{2}} X^{\frac{n-1}{2}}$. It follows that the polynomial $v(x)$ satisfies Lemma 1 with $M = N^{h-1}$ and $g(x) = v(x)$ if $X \leq \frac{1}{\sqrt{2}} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}}$. The dimension of B is $n = h\delta$, and the entries of the matrix B have bit-size $O(h \log N)$, therefore the running time of L^2 without fast integer arithmetic is $O(\delta^6 h^7 \log N + \delta^5 h^7 \log^2 N)$, which is $O(\delta^5 h^7 \log^2 N)$ because $\delta + 1 < (\log N)/2$, and the running time of $\tilde{L}^1$ is $O(h^{6+\varepsilon} \delta^{5+\varepsilon} \log N + h^{\omega+2+2\varepsilon} \delta^{\omega+1+\varepsilon} \log^{1+\varepsilon} N)$ for any $\varepsilon > 0$ using fast integer arithmetic and $\tilde{L}^1$, where $\omega \leq 2.376$ is the matrix multiplication complexity constant. We obtain the following concrete version of Th. 1:

Corollary 2 *Coppersmith's algorithm of Th. 1 with* $h = \lfloor \log N/\delta \rfloor$ *and* $X = \lfloor 2^{-1/2} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}} \rfloor$ *runs in time* $O((\log^9 N)/\delta^2)$ *without fast integer arithmetic using* L^2*, or* $O((\log^{7+\varepsilon} N)/\delta)$ *for any* $\varepsilon > 0$ *using fast integer arithmetic and* $\tilde{L}^1$*.*

Sketch of Proof: One can show that the cost of the root computation step performed at the end of Coppersmith's algorithm is less than the one of the LLL reduction. Moreover the number of loop iterations performed by Coppersmith's algorithm to find all solutions smaller than $N^{1/\delta}$ by exhaustive search is at most $O(N^{1/\delta}/X)$ which can be shown to be $O(1)$. Thus, from the analysis preceding Cor. 2, the asymptotic complexity of Coppersmith's algorithm is the one of one call to LLL (L^2 or $\tilde{L}^1$), with $h = \lfloor \log N/\delta \rfloor$. □

We will later see that the complexity upper bounds of Cor. 2 with $L^2$and $\tilde{L}^1$can actually be decreased. Indeed, we will uncover a special property of Coppersmith's matrix (see Lemma 2), which implies that $O\left(\frac{max\|\mathbf{b}_i^\star\|}{min\|\mathbf{b}_i^\star\|}\right) = O(N)$, so that the number of loop iterations $O\left(n^2 \log \frac{max\|\mathbf{b}_i^\star\|}{min\|\mathbf{b}_i^\star\|}\right)$ on the input basis used by Coppersmith's algorithm is $O(n^2 \log N)$ instead of the all-purpose bound $O(n^2 h \log N)$ [11]. By taking this observation into account, the upper bounds $O((\log^8 N)/\delta)$ and $O(\log^{6+\varepsilon} N)$ are respectively achieved for the $L^2$and $\tilde{L}^1$algorithms. In the sequel, we present another method improving Cor. 2, based on the same special property of Coppersmith's matrix, and which can be easily implemented.

3 Speeding up Coppersmith's Algorithm by Rounding

Our first main results is the following speedup over Coppersmith's algorithm:

Theorem 3. *There is an algorithm (namely, Alg. 1) which, given as input an integer* N *of unknown factorization and a monic polynomial* $f(x) \in \mathbb{Z}[x]$ *of degree* δ *and coefficients in* $\{0, \ldots, N-1\}$*, outputs all integers* $x_0 \in \mathbb{Z}$ *such that* $f(x_0) \equiv 0 \,(mod\, N)$ *and* $|x_0| \leq N^{1/\delta}$ *in time* $O(\log^7 N)$ *without fast integer arithmetic using the* L^2 *algorithm [18], or* $O(\log^{6+\varepsilon} N)$ *for any* $\varepsilon > 0$ *using fast integer arithmetic and the* $\tilde{L}^1$ *algorithm [20] in Step 7.*

3.1 Rounding for Coppersmith's Algorithm

The bottleneck of Coppersmith's algorithm is the LLL reduction of the matrix B, whose dimension is $n = h\delta$, and whose entries have bit-size $O(h \log N)$. Asymptotically, we have $h = O(\log N/\delta)$ so the dimension is $O(\log N)$ and the bit-size

is $O((\log^2 N)/\delta)$. We will modify Coppersmith's algorithm in such a way that we only need to LLL-reduce a matrix of the same dimension but with much smaller entries, namely bit-length $O(\log N)$.

To explain the intuition behind our method, let us first take a closer look at the matrix B an uncover one of its special property:

Lemma 2. *Let $X \leq N^{1/\delta}$. The maximal diagonal coefficient of Coppersmith's matrix B is $N^{h-1}X^{\delta-1} < N^h$, the minimal diagonal coefficient is $X^{h\delta-\delta} \leq N^{h-1}$, and $\frac{N^{h-1}X^{\delta-1}}{X^{h\delta-\delta}} \geq N^{1-1/\delta}$ if $h \geq 2$. Furthermore, if $X \geq \Omega(N^{\frac{h-1}{n-1}})$, $h \geq 2$ and $h\delta = O(\log N)$ then $X^{h\delta-\delta} \geq N^{h-O(1)}$*

Proof. The ratio $\frac{N^{h-1}X^{\delta-1}}{X^{h\delta-\delta}}$ is exactly $N^{h-1}/X^{h\delta-2\delta+1}$ which is clearly $\geq N^{1-1/\delta}$ if $X \leq N^{1/\delta}$ and $h \geq 2$. Now, let $X_0 = N^{\frac{h-1}{n-1}}$ so that $X = \Omega(X_0)$. We have $N^{1/\delta}/N^{\frac{h-1}{n-1}} \leq N^{1/(h\delta-1)}$, therefore $X_0 \geq N^{1/\delta-1/(h\delta-1)} = N^{(h\delta-1-\delta)/(\delta(h\delta-1))}$. Hence $X_0^\delta \geq N^{(h\delta-1-\delta)/(h\delta-1)} = N^{1-\delta/(h\delta-1)}$ and thus $X_0^{h\delta-\delta} > N^{h-2}$. Since $X = \Omega(X_0)$ and $h\delta = O(\log N)$, we obtain $X^{h\delta-\delta} \geq N^{h-O(1)}$. □

This implies that the diagonal coefficients of B are somewhat balanced: the matrix B is not far from being reduced. In fact, the first row of B has norm N^{h-1} which is extremely close to the bound $N^{h-1}/\sqrt{n}$ required by Lemma 1: intuitively, this means that it should not be too difficult to find a lattice vector shorter than $N^{h-1}/\sqrt{n}$.

To take advantage of the structure of B, we first size-reduce B to make sure that the subdiagonal coefficients are smaller than the diagonal coefficients. Then we round the entries of B so that the smallest diagonal coefficient becomes $\lfloor c \rfloor$ where $c > 1$ is a parameter. More precisely, we create a new $n \times n$ triangular matrix $\tilde{B} = (\tilde{b}_{i,j})$ defined by:

$$\tilde{B} = \left\lfloor cB/X^{h\delta-\delta} \right\rfloor \tag{3}$$

By Lemma 2, we have: $b_{i,i} \geq X^{h\delta-\delta}$ and $\tilde{b}_{i,i} \geq \lfloor c \rfloor$. We LLL-reduce the rounded matrix $\tilde{B}$ instead of B: let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ be the first vector of the reduced basis obtained. If we applied to B the unimodular transformation that LLL-reduces $\tilde{B}$, we may not even obtain an LLL-reduced basis in general. However, because of the special structure of B, it turns out that $\mathbf{v} = \mathbf{x}B$ is still a short non-zero vector of L, as shown below:

Lemma 3. *Let $B = (b_{i,j})$ be an $n \times n$ lower-triangular matrix over $\mathbb{Z}$ with strictly positive diagonal. Let $c > 1$. If $\tilde{B} = \lfloor cB/\min_{i=1}^n b_{i,i} \rfloor$ and $\mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, then $0 < \|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right) 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}}$.*

Proof. Let $\alpha = \min_{i=1}^{n} b_{i,i}/c$, so that $\tilde{B} = \lfloor B/\alpha \rfloor$. Define the matrix $\bar{B} = \alpha\tilde{B}$ whose entries are $\bar{b}_{i,j} = \alpha\tilde{b}_{i,j}$. Then $0 \leq b_{i,j} - \bar{b}_{i.j} < \alpha$, therefore $\|B - \bar{B}\|_2 < n\alpha$. We have:

$$\|\mathbf{x}B\| \leq \|\mathbf{x}(B-\bar{B})\| + \|\mathbf{x}\bar{B}\| \leq \|\mathbf{x}\| \times \|B-\bar{B}\|_2 + \alpha\|\mathbf{x}\tilde{B}\| < n\|\mathbf{x}\|\alpha + \alpha\|\mathbf{x}\tilde{B}\|.$$

Let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$. Then $\|\mathbf{x}\| \leq \|\tilde{\mathbf{v}}\|\|\tilde{B}^{-1}\|_2$, and we obtain $\|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right)\alpha\|\tilde{\mathbf{v}}\|$. The matrix $\tilde{B}$ is lower-triangular with all diagonal coefficients strictly positive because $c > 1$. Since $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, and $\tilde{B}$ is non-singular, $\mathbf{x}B \neq 0$ and we have:

$$\alpha\|\tilde{\mathbf{v}}\| \leq \alpha 2^{\frac{n-1}{4}} \det(\tilde{B})^{\frac{1}{n}} = 2^{\frac{n-1}{4}} \det(\bar{B})^{\frac{1}{n}} \leq 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}},$$

where we used the fact that matrices $\tilde{B}$, $\bar{B}$ and B are lower-triangular. The result follows by combining both inequalities. □

If $\mathbf{x}B$ is sufficiently short, then it corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$ satisfying Lemma 1, and the rest proceeds as in Coppersmith's algorithm. The whole rounding algorithm is given in Alg. 1, which will be shown to admit a lower complexity upper-bound than Coppersmith's algorithm to compute all roots up to $N^{1/\delta}$.

Algorithm 1 Coppersmith's Method with Rounding

Input: Two integers $N \geq 1$ and $h \geq 2$, a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0,\ldots,N-1\}$ and $2 < \delta + 1 < (\log N)/2$.
Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$.
1: Let $n = h\delta$, X the bound given in Th. 4, $c = (3/2)^n$ and $t = 0$.
2: **while** $Xt < N^{1/\delta}$ **do**
3: $f_t(x) = f(Xt + x) \in \mathbb{Z}[x]$.
4: Build the $n \times n$ matrix B whose rows are the $g_{i,j}(xX)$'s defined by (1).
5: Size-reduce B without modifying its diagonal coefficients.
6: Compute the matrix $\tilde{B} = \lfloor cB/X^{h\delta-\delta} \rfloor$ obtained by rounding B.
7: Run the L^2 algorithm [18] on the matrix $\tilde{B}$.
8: Let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ be the first vector of the reduced basis obtained.
9: The vector $\mathbf{v} = \mathbf{x}B$ corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
10: Compute all the roots x_0' of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
11: Output $x_0 = x_0' + Xt$ for each root x_0' which satisfies $f_t(x_0') \equiv 0 \,(\mathrm{mod}\, N)$ and $|x_0'| \leq X$.
12: $t \leftarrow t + 1$.
13: **end while**

We now justify the bound X given in Alg. 1. In order for Lemma 3 to be useful, we need to exhibit an upper bound for $\|\tilde{B}^{-1}\|_2$:

Lemma 4. *Let $B = (b_{i,j})$ be an $n \times n$ size-reduced lower-triangular matrix over $\mathbb{Z}$ with strictly positive diagonal. Let $c > 1$. If $\tilde{B} = \lfloor cB / \min_{i=1}^{n} b_{i,i} \rfloor$, then $\|\tilde{B}^{-1}\|_2 \leq \sqrt{n} \left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor$.*

By combining Lemmas 3 and 4, we obtain the following small-root bound X for Alg. 1:

Theorem 4. *Given as input two integers $N \geq 1$ and $h \geq 2$, a rational $c > 1$, and a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$, one loop of Alg. 1, corresponding to $t < N^{1/\delta}/X$, outputs all $x_0 = Xt + x_0' \in \mathbb{Z}$ s.t. $|x_0'| \leq X$ and $f(x_0) = 0 \mod N$, where $n = h\delta$ and*

$$X = \left\lfloor \frac{N^{\frac{h-1}{n-1}} \kappa_1^{-2/(n-1)}}{\sqrt{2} n^{1/(n-1)}} \right\rfloor \quad \textit{with} \quad \kappa_1 = n^{3/2} \left(\frac{3c-2}{2c-2}\right)^{n-1} \lfloor c \rfloor^{-1} + 1 \,.$$

Proof. Combining Lemma 4 with Lemma 3 where $\det(B)^{1/n} = N^{\frac{h-1}{2}} X^{\frac{n-1}{2}}$, we get $0 < \|\mathbf{x}B\| < \kappa_1 2^{\frac{n-1}{4}} N^{\frac{h-1}{2}} X^{\frac{n-1}{2}}$. It follows that Lemma 1 is satisfied with $M = N^{h-1}$ and $v(xX)$ corresponding to $\mathbf{x}B$ if $\|\mathbf{x}B\| \leq N^{h-1}/\sqrt{n}$. This gives the following condition on the bound X: $X \leq N^{(h-1)/(n-1)} 2^{-1/2} n^{-1/(n-1)} \kappa_1^{-2/(n-1)}$. □

The bound X of Th. 4 is never larger than that of Cor. 2. However, if one selects $c \geq (3/2)^n$, then the two bounds are asymptotically equivalent. This is why Alg. 1 uses $c = (3/2)^n$.

3.2 Running time: proof of Theorem 3

The original matrix B had entries whose bit-size was $O(h \log N)$. Let $\beta = \frac{N^h X^{\delta-1}}{X^{h\delta-\delta}}$ be the ratio between the maximal diagonal coefficient and the minimal diagonal coefficient of $\tilde{B}$. If B is size-reduced, the entries of the new matrix $\tilde{B} = \left\lfloor cB / X^{h\delta-\delta} \right\rfloor$ are upper bounded by $c\beta$.

By Lemma 2, we know that if $h \geq 2$, then $\beta \geq N^{1-1/\delta}$, and if further $X \geq \Omega(N^{\frac{h-1}{n-1}})$ and $h\delta = O(\log N)$, then $\beta = N^{O(1)}$. Hence, the bit-size of $\tilde{B}$'s entries is $\leq \log c + O(\log N)$. And the dimension of $\tilde{B}$ is the same as B, *i.e.* $h\delta$. It follows that the running time of L^2 in Step 7 is $O(\delta^6 h^6 (\log c + \log N) + \delta^5 h^5 (\log c + \log N)^2)$ without fast integer arithmetic, which is $O(\delta^5 h^5 (\log c + \log N)^2$ because $\delta < (\log N)/2 - 1$, and is $O((h\delta)^{5+\varepsilon}(\log c + \log N) + (h\delta)^{\omega+1+\varepsilon}(\log c + \log N)^{1+\varepsilon})$ for any $\varepsilon > 0$ using fast integer arithmetic and $\tilde{L}^1$ in Step. 7, where $\omega \leq 2.376$ is the matrix multiplication complexity constant.

This leads to our main result (Th. 3), a variant of Coppersmith's algorithm with improved complexity upper bound. More precisely, as in Coppersmith's algorithm, one can easily prove that the number of loops performed in Alg. 1

is at most constant. Indeed, when $c = (3/2)^n$, then $\kappa_1^{\frac{-2}{n-1}}$ converges to 1. This means that the bound X achieved by Th. 4 is asymptotically equivalent to the one achieved by Cor. 2, which completes the proof of Th. 3, because $\log c = O(\log N)$ when $c = (3/2)^n$.

Remark: Surprisingly, Lemma 2 also allows to prove that the $\tilde{L}^1$algorithm, when carefully analyzed using the balancedness of the Gram-Schmidt norms, already achieves the complexity bound $O(\log^{6+\varepsilon} N)$ given in Th. 3. Indeed, using Th. 6 from [20] which gives the $\tilde{L}^1$complexity upper bound $O(n^{3+\varepsilon}\tau) = O(\log^{3+\varepsilon} N\tau)$ where τ is the total number of iterations, and combining it with [11] applied to Coppersmith's matrix (Lemma 2), which gives $\tau = O(n^2 \log N) = O(\log^3 N)$, allows to retrieve the above complexity $O(\log^{6+\varepsilon} N)$. However, we propose in this paper a direct improvement of Coppersmith's method based on elementary tools and which can therefore be easily implemented on usual computer algebra systems (*e.g.* Sage, Magma, NTL) with immediate practical impact on cryptanalyses. Furthermore, we are not aware of any implementation of the $\tilde{L}^1$algorithm for the time being, which makes a practical comparison tricky.

In the sequel, we present a method that allows to speed up the exhaustive search which is performed to reach Coppersmith's bound $N^{1/\delta}$.

4 Chaining LLL

As recalled in Section 2.2, in order to find all solutions which are close to the bound $N^{1/\delta}$, one should not use a very large dimension (*i.e.* $n = O(\log N)$). Instead, it is better to use a lattice of reasonable dimension and to perform exhaustive search on the most significant bits of x until finding all solutions. Namely, we consider polynomials $f_t(x) = f(X \cdot t + x)$ where $0 \leqslant t < \frac{N^{1/\delta}}{X}$ and $X = \lfloor 2^{\frac{-1}{2}} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}} \rfloor$. Thus, an initial solution x_0 that can be written $x_0 = X \cdot t_0 + x'_0$ is obtained by finding the solution x'_0 of the polynomial f_{t_0}. In this case, this solution satisfies $|x'_0| < X$ and it has a correct size for LLL to find it using a lattice of dimension n. For each polynomial f_t, one runs LLL on a certain matrix (Step 4 of Alg. 1). In Section 4.1, we describe a method that allows to take advantage of the LLL performed for the case $t = i$ to reduce (in practice) the complexity of the LLL performed for the case $t = i+1$. Thereafter, in Section 4.2 we combine this improvement with the rounding approach described in Section 3. The proofs of the results presented in this section can be found in the full version of this paper.

4.1 Exploiting Relations Between Consecutive Lattices

The following proposition discloses a surprising connection between the lattice used for the case $t = i$ and the next lattice used for $t = i+1$. This connection is based on the well-known *Pascal matrix* $P = (p_{s,t})$ defined as the $n \times n$ lower-triangular matrix whose non-zero coefficients are the binomials: $p_{s,t} = \binom{s}{t}$ for $0 \leq t \leq s \leq n-1$.

Proposition 1. *Let B be a basis of the n-dimensional lattice used by Coppersmith's algorithm to find all small roots of the polynomial $f_i(x) = f(X \cdot i + x)$, where X is the small-root bound. Then $B \cdot P$ is a basis of the "next" lattice used for the polynomial $f_{i+1}(x)$.*

Proof. Because all lattice bases are related by some unimodular matrix, it suffices to prove the statement for a special basis B. We thus only consider the special basis $B = B_i$ formed by the n shifted polynomials constructed from $f_i(x)$ and written in the basis $\mathscr{B} = \left(1, xX^{-1}, (xX^{-1})^2, \ldots, (xX^{-1})^{n-1}\right)$. For the case $t = i+1$, one tries to solve the polynomial

$$f_{i+1}(x) = f(X \cdot (i+1) + x) = f(X \cdot i + x + X) = f_i(x + X)\,.$$

Therefore, the shifted polynomials constructed from f_{i+1} are the same as for the case $t = i$, but written in the different basis $\mathscr{B}' = (1, xX^{-1}+1, (xX^{-1}+1)^2, \ldots, (xX^{-1}+1)^{n-1})$. Yet, we need to return to the original representation of the polynomials, *i.e.* in the basis $\mathscr{B}$. To this end, we use the following property regarding the lower triangular Pascal matrix P: $\mathscr{B}'^T = P \cdot \mathscr{B}^T$. As a consequence, left-multiplying each side of this equality by the matrix B_i proves that the matrix $B_i \cdot P$ is a basis of the lattice used for finding small roots of the polynomial $f_{i+1}(x)$. □

The proposition allows us to use different matrices to tackle the polynomial $f_{i+1}(x)$ than the one initially used by Coppersmith's method. In particular, we can use a matrix of the form $B^R \cdot P$ where B^R is an LLL-reduced basis of the previous lattice used to solve $f_i(x)$: intuitively, it might be faster to LLL-reduce such matrices than the initial Coppersmith's matrix. Although we are unable to prove this, we can show that the vectors of such a matrix are not much longer than that of B:

Corollary 5 *Let B_i^R be the LLL-reduced matrix used for solving f_t for $t = i$ and P be the Pascal matrix. The matrix $B_{i+1} = B_i^R \cdot P$ spans the same lattice used for solving the case $t = i+1$. This matrix consists of vectors $\mathbf{b_{i+1},}_j$ whose norms are close to vector norms of the LLL-reduced matrix B_i^R. Namely, for all $1 \leqslant j \leqslant n$ we have: $||\mathbf{b_{i+1},}_j|| < \sqrt{n} \cdot 2^{n-1} \cdot ||\mathbf{b_i^R}_{,j}||$. In particular, for the case $i = t_0$ the first vector of B_{i+1} has a norm bounded by $2^{n-1} \cdot N^{h-1}$.*

Cor. 5 shows us that vectors of B_{i+1} are relatively close to the ones in the LLL-reduced matrix B_i^R. Thus, we intuitively expect the LLL-reduction of B_{i+1} to be less costly than the one of the original Coppersmith's matrix. However, our bounds are too weak to rigorously prove this. Yet, one can use this property iteratively to elaborate a new method which *chains* all LLL reductions as follows. First, one LLL-reduces B_0 for the case $t = 0$. This gives a reduced matrix B_0^R. Then, one iterates this process by performing LLL reduction on $B_{i+1} = B_i^R \cdot P$ (for $i \geq 0$) to obtain B_{i+1}^R and so forth until all solutions are found (each time by solving the polynomial corresponding to the first vector of B_i^R).

In the sequel, we study this *chaining method* by performing similar roundings as in Section 3 before each call of LLL reduction.

4.2 Rounding and Chaining LLL

During the exhaustive search described in Section 4.1, we perform the LLL algorithm on the matrix $B_{i+1} = B_i^R \cdot P$ for $0 \leqslant i < N^{1/\delta}/X$, where B_i^R is LLL-reduced. It is worth noticing that the structure of B_i^R and thereby of B_{i+1}, is different from the original Coppersmith's matrix B_0 (in particular, it is not triangular anymore). Yet, we are able to show that under certain conditions on B_{i+1} verified experimentally, one can combine the rounding technique of Section 3 with the chaining technique of Section 4.1. Indeed, we show that during the chaining loop, one can size-reduce B_{i+1} and then round its elements for all $i \geq 0$ as follows:

$$\tilde{B}_{i+1} = \left\lfloor cB_{i+1} / \min_{1 \leq i \leq n} \|\mathbf{b}_i^\star\| \right\rfloor , \tag{4}$$

where $\mathbf{b}_i^\star$ are Gram-Schmidt vectors of B_{i+1} and c is a rational that will be determined later. Then, one applies LLL on the rounded matrix $\tilde{B}_{i+1}$ as performed in Section 3. We obtain an LLL-reduced matrix $\tilde{B}_{i+1}^R$ and a unimodular matrix $\tilde{U}_{i+1}$ such that $\tilde{U}_{i+1} \cdot \tilde{B}_{i+1} = \tilde{B}_{i+1}^R$. Then one shows that by applying $\tilde{U}_{i+1}$ on B_{i+1}, the first vector of this matrix $\tilde{U}_{i+1} \cdot B_{i+1}$ is a short vector that allows to find the solutions provided that they are smaller than a bound X that will be determined latter. For the sake of clarity, in the sequel we denote by B the matrix B_{i+1}, and by $\mathbf{x}B$, the first vector of matrix $\tilde{U}_{i+1} \cdot B_{i+1}$. We would like to exhibit an upper-bound on $\|\mathbf{x}B\|$. To this end, we will need, as in Section 3, to upper-bound the value $\|\tilde{B}^{-1}\|_2$. This is done in the following lemma:

Lemma 5. *Let $B = (b_{i,j})$ be an $n \times n$ non-singular integral matrix and $\alpha \geq 1$ such that $n\alpha\|B^{-1}\|_2 < 1$. Then the matrix $\tilde{B} = \lfloor B/\alpha \rfloor$ is invertible with $\|\tilde{B}^{-1}\|_2 \leq \alpha\|B^{-1}\|_2(1 - n\alpha\|B^{-1}\|_2)^{-1}$.*

As one can see, this value depends on $\|B^{-1}\|_2$ which is given in Lemma 6.

Lemma 6. *Let B be an $n \times n$ non-singular size-reduced matrix, with Gram-Schmidt vectors $\mathbf{b}_i^\star$. Then $\|B^{-1}\|_2 \leq \sqrt{n}(3/2)^{n-1}/\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|$.*

One can now give an upper-bound on $\|\mathbf{x}B\|$:

Corollary 6 *Let $B = (b_{i,j})$ be an $n \times n$ size-reduced non-singular matrix over $\mathbb{Z}$. Let $\alpha \geq 1$ such that $n^2\alpha\|B^{-1}\|_2 < 1$. Then $\tilde{B} = \lfloor cB/\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|\rfloor = \lfloor B/\alpha\rfloor$ is non-singular. And if $\mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, then: $0 < \|\mathbf{x}B\| < \frac{c^{\frac{n+1}{n}}}{(c-n^{3/2}(3/2)^{n-1})(c-n^{5/2}(3/2)^{n-1})^{1/n}}\, 2^{\frac{n-1}{4}}\, \det(B)^{\frac{1}{n}}$.*

Again, if $\|\mathbf{x}B\|$ is sufficiently short, then it corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$ satisfying Lemma 1. In particular, for the case $t = t_0$, solving this polynomial equation would allow to retrieve the solution x_0. Note that the condition $n^2\alpha\|B^{-1}\|_2 < 1$ specified in Cor. 6 gives a condition on the rational c. Indeed, since $\alpha = \min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|/c$ and using Lemma 6, one gets: $n^2\alpha\|B^{-1}\|_2 \leqslant n^2\,\frac{\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|}{c}\,\frac{\sqrt{n}(3/2)^{n-1}}{\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|} \leqslant \frac{n^{5/2}(3/2)^{n-1}}{c} < 1$ that is c should be such that $c > n^{5/2}(3/2)^{n-1}$. The whole chaining and rounding algorithm is depicted in Algorithm 2. Note that in practice, we do not need to perform Step 8 of Alg. 2 and that $\min_{1\leq i\leq n}\|\mathbf{b}_{\mathbf{t}+\mathbf{1}_i}^\star\|$ can be estimated instead of being computed in Step 9 (see Section 4.3 for more details).

In the following, we give a small-root bound X on the solution x_0' sufficient to guarantee success:

Theorem 7. *Given as input two integers $N \geq 1$ and $h \geq 2$, a rational $c > n^{5/2}(3/2)^{n-1}$, and a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0,\ldots,N-1\}$, one loop of Alg. 2, corresponding to $t < N^{1/\delta}/X$, outputs all $x_0 = Xt + x_0' \in \mathbb{Z}$ s.t. $|x_0'| \leq X$ and $f(x_0) = 0 \mod N$, and $n = h\delta$, where*

$$X = \left\lfloor \frac{N^{\frac{h-1}{n-1}}\,\kappa_2^{\frac{-2}{n-1}}}{\sqrt{2}\,n^{1/(n-1)}} \right\rfloor \quad \text{and} \quad \kappa_2 = \frac{c^{\frac{n+1}{n}}}{(c-n^{3/2}(3/2)^{n-1})(c-n^{5/2}(3/2)^{n-1})^{1/n}}.$$

The bound X of Th. 7 is never larger than that of Cor. 2. However, if one selects $c > n^{5/2}(3/2)^{n-1}$, then the two bounds are asymptotically equivalent. This is why Alg. 2 uses $c = n^{5/2}(3/2)^n$.

4.3 Complexity Analysis: A Heuristic Approach

The complexity of Alg. 2 relies on the complexity of the LLL-reduction performed in Step 10. The cost of this reduction depends on the size of coefficients

Algorithm 2 Coppersmith's Method with Chaining and Rounding

Input: Two integers $N \geq 1$ and $h \geq 2$, a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0,\ldots,N-1\}$ and $2 < \delta + 1 < (\log N)/2$.

Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$.

1: Performs Step 1 and Steps 3 to 7 of Alg. 1. Step 7 returns $\tilde{B}_0^R$ and $\tilde{U}_0$ such that $\tilde{U}_0 \cdot \tilde{B}_0 = \tilde{B}_0^R$.
2: Let $n = h\delta$, X the bound given in Th. 7, $c = n^{\frac{5}{2}}(\frac{3}{2})^n$, $t = 0$, P is the $n \times n$ lower triangular Pascal matrix.
3: Compute the matrix $\tilde{U}_0 \cdot B_0$, where B_0 is the matrix computed in Step 5 of Alg. 1.
4: The first vector of $\tilde{U}_0 \cdot B_0$ corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
5: Compute and output all roots $x_0 \in \mathbb{Z}$ of $v(x)$ satisfying $f(x_0) \equiv 0 \pmod N$ and $|x_0| \leq X$.
6: **while** $Xt < N^{1/\delta}$ **do**
7: Compute the matrix $B_{t+1} = \tilde{U}_t \cdot B_t \cdot P$.
8: Size-reduce B_{t+1}.
9: Compute the matrix $\tilde{B}_{t+1} = \lfloor cB_{t+1} / \min_{1\leq i\leq n} \|\mathbf{b_{t+1}}_i^\star\| \rfloor$ obtained by rounding B_{t+1}.
10: Run L^2 algorithm on matrix $\tilde{B}_{t+1}$ which returns $\tilde{B}_{t+1}^R$ and $\tilde{U}_{t+1}$ s.t. $\tilde{U}_{t+1} \cdot \tilde{B}_{t+1} = \tilde{B}_{t+1}^R$.
11: Compute the matrix $\tilde{U}_{t+1} \cdot B_{t+1}$.
12: The first vector of $\tilde{U}_{t+1} \cdot B_{t+1}$ corresponds to a polynomial of the form $v(xX)$.
13: Compute all the roots x_0' of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
14: Output $x_0 = x_0' + Xt$ for each root x_0' which satisfies $f(x_0' + Xt) \equiv 0 \pmod N$ and $|x_0'| \leq X$.
15: $t \leftarrow t + 1$.
16: **end while**

in matrix $B = \tilde{B}_{t+1}$, which itself depends on the value $\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\|$. The exact knowledge of this value does not seem straightforward to obtain without computing the Gram-Schmidt matrix explicitly. However, experiments show that the Gram-Schmidt curve is roughly decreasing, *i.e.* $\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\| \approx \|\mathbf{b}_n^\star\|$ and is roughly symmetric: *i.e.* $\log\|\mathbf{b}_i^\star\| - \log\|\mathbf{b}_{n/2}^\star\| \approx \log\|\mathbf{b}_{n/2}^\star\| - \log\|\mathbf{b}_{n-i+1}^\star\|$. If we assume these two experimental facts, we deduce that $\|\mathbf{b}_{n/2}^\star\| \approx |\det(B)|^{1/n}$. By duality, this means that $\|\mathbf{b}_n^\star\| \approx |\det(B)|^{2/n}/\|\mathbf{b}_1^\star\|$. Furthermore, from the definition of the Gram-Schmidt orthogonalization, we know that $\|\mathbf{b}_1^\star\| = \|\mathbf{b}_1\|$, where $\mathbf{b}_1$ is the first vector of matrix B. Therefore we have:

$$\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\| \approx \|\mathbf{b}_n^\star\| \approx |\det(B)|^{2/n}\|\mathbf{b}_1^\star\|^{-1} = N^{h-1}X^{n-1}\|\mathbf{b}_1\|^{-1}, \qquad (5)$$

Thus, we need an estimation on $\|\mathbf{b}_1\|$. Since in practice matrix $B = B_{i+1} = \tilde{U}_i \cdot B_i \cdot P$ is already nearly size-reduced, one can skip Step 8 of Alg. 2. Therefore, vector $\mathbf{b}_1$ is the first vector of matrix $\tilde{U}_i \cdot B_i \cdot P$. Using Cor. 6, one deduces that the first vector matrix $\tilde{U}_i \cdot B_i$ is roughly as short as the first vector of an LLL-reduced matrix. From the well-known experimental behavior of LLL [17], we can model the first vector of the LLL-reduced basis as a "random" vector of norm $\approx 1.02^n |\det(B)|^{1/n}$. Since the Pascal matrix P has a norm smaller than 2^{n-1} , one gets the bound $\|\mathbf{b}_1\| \leqslant \sqrt{n}2^{n-1}1.02^n|\det(B)|^{1/n}$. Therefore, we

deduce that: $\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\| \approx |\det(B)|^{1/n}/(\sqrt{n}2^{n-1}1.02^n)$. In practice, we conjecture that $\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\| > |\det(B)|^{1/n}/\beta^n$, where $\beta < 2$ (see Fig. 5 in Sec. 5).

This discussion leads to the following heuristic approach regarding the method: firstly, one should rather use the estimation (5) in Step 9 of Alg. 2, instead of explicitly computing the Gram-Schmidt matrix; secondly, one can skip Step 8 of Alg. 2. This heuristic version of Algorithm 2 is the one we used during our experiments, all these assumptions were always verified.

To conclude our analysis, it suffices to reduce a rounded matrix such that $\max_{1\leq i\leq n}\|\tilde{\mathbf{b}}_i^\star\| \leq c\max_{1\leq i\leq n}\|\mathbf{b}_i^\star\|/\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\| \leqslant c\beta^{2n}$, instead of being such that $\max_{1\leq i\leq n}\|\tilde{\mathbf{b}}_i^\star\| \leq \beta^n|\det(B)|^{1/n}$. This means that we are trading entries of size $O(n)$. Therefore, by considering $n = O(\log N)$, we obtain the same complexity as in Theorem 3 but in a heuristic way. However, even if both asymptotic complexities are identical, in practice for reasonable dimensions the speed-up brought by using Alg. 2 rather than Alg. 1 is considerable (see Section 5). Indeed, the LLL-reduction of matrix $\tilde{U}_i \cdot B_i \cdot P$ (Step 10 of Alg. 2) performs surprisingly faster than expected. This comes from the fact that for reasonable dimensions, the Gram-Schmidt curve of this matrix remains quite close to the one of matrix $\tilde{U}_i \cdot B_i$, where $\tilde{U}_i \cdot B_i$ turns out to be LLL-reduced (or nearly). Besides, the overall running-time of Alg. 2 is approximately the time spent to perform one LLL-reduction, multiplied by the number of executed loops, *i.e.* by $N^{1/\delta}/X$.

5 Experiments

We implemented Coppersmith's algorithm and our improvements (Algs. 1 and 2) using Shoup's NTL library [21]. However, for the LLL reduction, we used the fplll implementation [3] by Cadé *et al.*, which includes the L^2 algorithm [18]: fplll is much faster than NTL for Coppersmith's matrices. It should be stressed that fplll is a wrapper which actually implements several variants of LLL, together with several heuristics: L^2 is only used as a last resort when heuristic variants fail. This means that there might be a discrepancy between the practical running time and the theoretical complexity upper bound of LLL routines. Our test machine is a 2.93-GHz Intel Core 2 Duo processor E7500 running on Fedora. Running times are given in seconds. Like in [10], we used the case $\delta = 3$, and N an RSA-type modulus: the exact polynomial congruence is derived from RSA encryption with public exponent δ. Then, one loop of Coppersmith's algorithm , with $n = 3h$, can find all roots x_0 as long as $|x_0'| \leq X = \lfloor 2^{-1/2}N^{\frac{h-1}{n-1}}n^{-\frac{1}{n-1}}\rfloor$. For a fixed h, the rounding strategy (Alg. 1) gives a worse bound than X, but the difference can be made arbitrarily small by increasing the parameter c: in our experiments, we therefore chose the smallest value of c such that $\kappa_1^{\frac{-2}{n-1}}$ and $\kappa_2^{\frac{-2}{n-1}}$ are larger than 0.90, so that the new bound is never less than the old bound X by

more than 10%, which is essentially the same. However, we note that the value c can be taken smaller in practice.

Furthermore, it is worth noticing that since the value α is not significant in itself, in order to increase the efficiency, one can round matrices at negligible cost by taking $\alpha := 2^{\lfloor \log_2(\alpha) \rfloor}$ and performing shifts of $\lfloor \log_2(\alpha) \rfloor$ bits. In the same vein, one can increment t by 2 instead of 1 in Coppersmith's algorithm or in Step 12 of Alg. 1, and one can multiply the matrix $\tilde{U}_i \cdot B_i$ by P^2 instead of P in Step 7 of Alg. 2. This comes from the fact that if $0 < x'_0 < X$ (resp. $-X < x'_0 < 0$), then $x'_0 - X$ (resp. $x'_0 + X$) is also a valid solution. This refinement allows to divide by 2 the global timing of Coppersmith's algorithm and Alg. 1. However, it seems to be much less relevant when applied to Alg. 2.

Figures 1 and 2 summary our limited experiments respectively comparing one loop of Coppersmith's algorithm with Alg. 1 and Alg. 2 in practice. They provide the bit-length of X and the corresponding running times of the lattice reduction only, because the cost of solving a univariate equation over $\mathbb{Z}$ turns out to be much less in practice. Running times are given as averages over 5 samples. For a typical case where $\lceil \log N \rceil = 2048$, the whole Coppersmith's algorithm would perform in $((\lfloor 2048/3 - 666 \rfloor)/2) \times 6431.2 \approx 6.7$ years and the new Alg. 2 would perform in $(\lfloor 2048/3 - 666 \rfloor) \times 15.52 \approx 11.8$ days, which is about 207 times faster (see Fig. 2 and 6).

Fig. 1. Bounds and running time of rounding method for cubic congruences

Size of N	Data type	Parameter h				
		10	15	20	25	30
1024	Size of X	318	324	328	331	332
	$T_{original}$	2.54	30.48	216.3	793.4	3720.8
	$T_{rounded}$	0.68	4.49	18.22	48.17	175.9
	Speed-up	3.74	6.79	11.87	16.47	21.16
2048	Size of X	634	650	658	663	666
	$T_{original}$	13.47	150.7	865.7	3078	10146.7
	$T_{rounding}$	3.14	17.79	63.3	166.4	379.8
	Speed-up	4.29	8.40	13.67	18.50	26.72
4096	Size of X	1270	1302	1318	1327	1333
	$T_{original}$	41.45	582.6	3162	11968	42053
	$T_{rounded}$	7.07	43.25	157.5	449.8	1301.5
	Speed-up	5.86	13.47	20.07	26.61	32.31

Fig. 2. Bounds and running time of rounding plus chaining method for cubic congruences

Size of N	Data type	Parameter h				
		10	15	20	25	30
1024	Size of X	316	323	327	330	332
	$T_{original}$	2.14	23.55	161.55	646.37	1955.1
	T_{rc}	0.04	0.42	1.71	5.56	12.71
	$Speed-up_{rc}$	53.5	56.07	94.47	116.25	153.83
2048	Size of X	633	649	657	663	666
	$T_{original}$	8.21	95.12	641.22	2299.5	6431.2
	T_{rc}	0.07	0.55	2.39	7.75	15.52
	$Speed-up_{rc}$	117.28	172.95	268.29	296.71	414.38
4096	Size of X	1270	1302	1318	1327	1333
	$T_{original}$	27.64	378.62	2226	8303.2	25813
	T_{rc}	0.11	0.87	3.73	11.72	29.65
	$Speed-up_{rc}$	251.27	435.19	596.78	708.46	870.6

From Figure 3, we see that we already get significant speedups (say, larger than 10) even for small values of h and typical sizes of N, by using the rounding method (Alg. 1). The speedup grows when $\log N$ or h grows: for fixed N, the speedup grows roughly a bit less than quadratically in h, whereas the theoretical analysis gives a speedup linear in h. From Figure 4, we see that

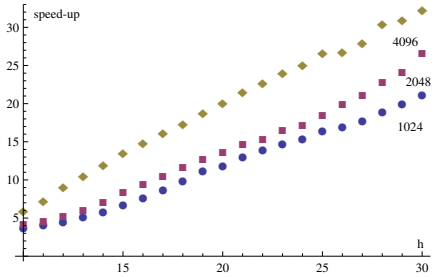

Fig. 3. Speed-up of rounding method

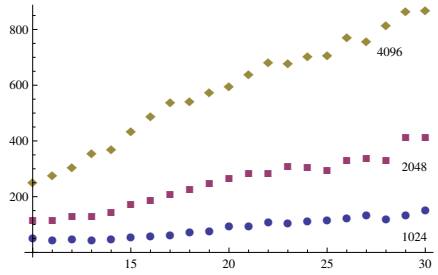

Fig. 4. Speed-up of rounding plus chaining method

we can obtain more speedups as the sizes of N or h increase, by using the rounding and chaining method (Alg. 2). Hence, our improvement is practical and allows to get much closer to the asymptotic small-root bound. Furthermore, we verify the assumption on value $\min_{1\le i\le n}\|b_i^\star\|$ for matrix B. Let $\max_{1\le i\le n}\|\mathbf{b}_i^\star\| \approx \beta_1^n \mathrm{vol}(L)^{1/n}$ and $\min_{1\le i\le n}\|\mathbf{b}_i^\star\| \approx \beta_2^n \mathrm{vol}(L)^{1/n}$. In this paper, we have assumed that $\beta_1 = 1/\beta_2$. We summary the results of our experiments for $\lceil \log N \rceil = 512$ with dimensions $30, 60, 90, 120, 150$ in Table 5. We can see that $\beta_1 \times \beta_2 \approx 1$ and that $\beta_1 \le 2$. This means our assumptions are reasonable.

Fig. 5. Beta values for $\lceil \log N \rceil = 512$

Data type	Parameter h				
	10	20	30	40	50
β_1	1.7582	1.8751	1.9093	1.9218	1.9435
β_2	0.5460	0.5271	0.5155	0.5091	0.5077
product	0.9600	0.9883	0.9842	0.9785	0.9867

Fig. 6. Timings comparisons for the total method

	$\log N$		
	1024	2048	4096
Original	5.8 days	6.7 years	1757782 years
Alg. 2	1.8 hours	11.8 days	4038 years
Speed-up	77	207	435

6 Other Small-Root Algorithms

Other small-root algorithms (see the surveys [14, 16]) are based on the same main ideas where LLL reduction plays a crucial role. Due to the different structure of the matrices in these settings, a direct application of our new approach does not seem to provide the same speedup. We leave it as an open problem to obtain polynomial (non-constant) speedups for these other small-root algorithms: this might be useful to make practical attacks on certain fully homomorphic encryption schemes (see [5]). See the extended version of this paper for a further discussion on these generalizations.

Acknowledgements. We would like to thank the anonymous reviewers of PKC'14 for their valuable comments.

References

1. D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):1339, 2000.
2. J. Buchmann. Reducing lattice bases by means of approximations. In *Algorithmic Number Theory – Proc. ANTS-I*, volume 877 of *Lecture Notes in Computer Science*, pages 160–168. Springer, 1994.
3. D. Cadé, X. Pujol, and D. Stehlé. FPLLL library, version 3.0. Available from `http://perso.ens-lyon.fr/damien.stehle`, Sep 2008.
4. H. Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.
5. H. Cohn and N. Heninger. Approximate common divisors via lattices. *IACR Cryptology ePrint Archive*, 2011:437, 2011.
6. D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.
7. D. Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1996.
8. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997. Journal version of [7, 6].
9. J.-S. Coron. Finding small roots of bivariate integer polynomial equations: A direct approach. In *Advances in Cryptology – Proc. CRYPTO '07*, volume 4622 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2007.
10. C. Coupé, P. Q. Nguyen, and J. Stern. The effectiveness of lattice attacks against low-exponent RSA. In *Public Key Cryptography – Proc. PKC '99*, volume 1560 of *Lecture Notes in Computer Science*, pages 204–218. Springer, 1999.
11. H. Daudé and B. Vallée. An upper bound on the average number of iterations of the lll algorithm. *Theor. Comput. Sci.*, 123(1):95–115, 1994.
12. N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding – Proc. IMA '97*, volume 1355 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 1997.
13. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.
14. A. May. Using LLL-reduction for solving RSA and factorization problems: A survey. 2010. In [19].
15. H. Najafi, M. Jafari, and M.-O. Damen. On adaptive lattice reduction over correlated fading channels. *Communications, IEEE Transactions on*, 59(5):1224–1227, 2011.
16. P. Q. Nguyen. Public-key cryptanalysis. In I. Luengo, editor, *Recent Trends in Cryptography*, volume 477 of *Contemporary Mathematics*. AMS–RSME, 2009.
17. P. Q. Nguyen and D. Stehlé. LLL on the average. In *Algorithmic Number Theory – Proc. ANTS*, LNCS, pages 238–256. Springer, 2006.
18. P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. of Computing*, 39(3):874–903, 2009.
19. P. Q. Nguyen and B. Vallée, editors. *The LLL Algorithm: Survey and Applications*. Information Security and Cryptography. Springer, 2010.
20. A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity: extended abstract. In *Proc. STOC '11*, pages 403–412. ACM, 2011.
21. V. Shoup. Number Theory C++ Library (NTL) version 5.4.1. Available at `http://www.shoup.net/ntl/`.
22. V. Shoup. OAEP reconsidered. *J. Cryptology*, 15(4):223–249, 2002.