Public Key Cryptography '99
(March 1–3, 1999, Kamakura, Kanagawa, Japan)
H. Imai and Y. Zheng (Eds.), vol. ???? of Lecture Notes in Computer Science, pages ???–???

The Effectiveness of Lattice Attacks Against Low-Exponent RSA

Christophe Coupé[1], Phong Nguyen[2], and Jacques Stern[2]

[1] coupe@ens-lyon.fr, École Normale Supérieure de Lyon, Département de Mathématiques et d'Informatique, 46 allée d'Italie, 69364 Lyon Cedex 07, France
[2] {pnguyen,stern}@ens.fr, http://www.dmi.ens.fr/~{pnguyen,stern}/
École Normale Supérieure, Laboratoire d'Informatique,
45 rue d'Ulm, 75230 Paris Cedex 05, France

Abstract. At Eurocrypt '96, Coppersmith presented a novel application of lattice reduction to find small roots of a univariate modular polynomial equation. This led to rigorous polynomial attacks against RSA with low public exponent, in some particular settings such as encryption of stereotyped messages, random padding, or broadcast applications *à la* Haståd. Theoretically, these are the most powerful known attacks against low-exponent RSA. However, the practical behaviour of Coppersmith's method was unclear. On the one hand, the method requires reductions of high-dimensional lattices with huge entries, which could be out of reach. On the other hand, it is well-known that lattice reduction algorithms output better results than theoretically expected, which might allow better bounds than those given by Coppersmith's theorems. In this paper, we present extensive experiments with Coppersmith's method, and discuss various trade-offs together with practical improvements. Overall, practice meets theory. The warning is clear: one should be very cautious when using the low-exponent RSA encryption scheme, or one should use larger exponents.

1 Introduction

One longstanding open problem in cryptography is to find an efficient attack against the RSA public key cryptosystem [13]. In the general case, the best-known method is factoring, although the equivalence of factorization and breaking RSA is still open (note that recent results [3] suggest that breaking RSA might be easier than factoring). However, under certain conditions, more efficient attacks are known (for a survey, see [2]). One of these conditions is when the public exponent is small, *e.g.* 3. This is the so-called low-exponent RSA, which is quite popular in the real world.

The most powerful known attack against low-exponent RSA is due to Coppersmith [5, 6]. At Eurocrypt '96, Coppersmith presented two applications [5, 4] of a novel use of the celebrated LLL algorithm [12]. Both applications were searches for small roots of certain polynomial equations: one for univariate modular equations, the other for bivariate integer equations. Instead of using lattice reduction algorithms as shortest vector oracles, Coppersmith applied the LLL algorithm to determine a subspace containing all reasonably short lattice points. He then deduced rigorous polynomial attacks, as opposed to traditional heuristic lattice-based attacks.

Finding small integer roots of a modular polynomial equation has great practical significance, for instance with the low-exponent RSA encryption scheme, or the KMOV cryptosystem (see [1]). More precisely, in the case of low-exponent RSA, such roots are related to the problems of encryption of stereotyped messages, random padding and broadcast applications.

However, Coppersmith did not deal with practical issues: the practical behaviour of his attack was unclear. On the one hand, the method would *a priori* require reductions of high-dimensional lattices with huge entries, in order to achieve the theoretical bounds. For instance, with a small example such as 512-bit RSA and a public exponent of 3, Coppersmith's proofs suggest to reduce matrices of dimension over 300, and 17000-digit entries. Obviously, some adjustements need to be made. On the other hand, it is well-known that lattice reduction algorithms output better results than theoretically expected. Moreover, one could apply improved reduction algorithms such as [14], instead of LLL. Thus, if one uses smaller parameters than those suggested by Coppersmith's theorems, one might still obtain fairly good results.

In this paper, we present extensive experiments with Coppersmith's method applied to the low-exponent RSA case, and discuss various trade-offs together with practical improvements. To our knowledge, only limited experiments (see [1, 9]) had previously been carried out. Our experiments tend to validate Coppersmith's approach. Most of the time, we obtained experimental bounds close to the maximal theoretical bounds. For instance, sending e linearly related messages to participants with the same public exponent e is theoretically insecure. This bound seems unreachable in practice, but we were able to reach the bound $e+1$ in a very short time. The warning is clear: one should be very cautious when using low-exponent RSA encryptions, or one should use larger exponents.

The remainder of the paper is organized as follows. In Section 2, we review Coppersmith's method. In Section 3, we recall applications of this method to the low-exponent RSA encryption scheme. We describe our implementation, and discuss practical issues in Section 4. Finally, Section 5 presents the experiments, which gives various trade-offs.

2 Coppersmith's Method

In this section, we recall Coppersmith's method, as presented in [6]. Let N be a large composite integer of unknown factorization, and $p(x) = x^\delta + p_{\delta-1}x^{\delta-1} + \cdots + p_2x^2 + p_1x + p_0$, be a monic integer polynomial. We wish to find an integer x_0 such that, for some $\varepsilon > 0$:

$$p(x_0) \equiv 0 \pmod{N} \tag{1}$$

$$|x_0| < X = \frac{N^{(1/\delta)-\varepsilon}}{2} \tag{2}$$

(2) means that we look for a reasonably short solution. We select an integer h such that:

$$h \geq \max\left(\frac{\delta - 1 + \varepsilon\delta}{\varepsilon\delta^2}, \frac{7}{\delta}\right) \tag{3}$$

Let $n = h\delta$. For $(i, j) \in [0..\delta-1] \times [1..h-1]$, let the polynomial $q_{i,j}(x) = x^i p(x)^j$, for which $q_{i,j}(x_0) \equiv 0 \pmod{N^j}$.

A rational triangular matrix M is built using the coefficients of the polynomials $q_{i,j}(x)$, in such a way that an integer linear combination of the rows of M corresponding to powers of x_0 and y_0 will give a vector with relatively small Euclidean norm. Multiplying by the least common denominator produces an integer matrix on which lattice basis reduction can be applied. This will disclose a certain linear relation satisfied by all sufficiently short vectors. Finally, this relation will translate to a polynomial relation on x_0 over $\mathbb{Z}$ (not mod N) of degree at most n, which we can solve over $\mathbb{Z}$ to discover x_0.

The matrix M of size $(2n - \delta) \times (2n - \delta)$ is broken into four blocks:

$$M = \begin{pmatrix} A & B \\ 0 & C \end{pmatrix}.$$

The $n \times (n - \delta)$ block B has rows indexed by $g \in [0..n - 1]$, and columns indexed by $\gamma(i, j) = n + i + (j - 1)\delta$ with $(i, j) \in [0..\delta] \times [1..h - 1]$, so that $n \leq \gamma(i, j) < 2n - \delta$. The entry at $(g, \gamma(i, j))$ is the coefficient of x^g in the polynomial $q_{i,j}(x)$. The $(n - \delta) \times (n - \delta)$ block C is a diagonal matrix, with the value N^j in each column $\gamma(i, j)$. The $n \times n$ block A is a diagonal matrix, whose value in row g is a rational approximation to $X^{-g}/\sqrt{n}$, where X is defined by (2).

The rows of M span a lattice. In that lattice, we are interested in a target vector $\mathbf{s}$, related to the unknown solution x_0. Namely, we define $\mathbf{s} = \mathbf{r}M$, where $\mathbf{r}$ is a row vector whose left-hand elements are $r_g = x_0^g$, and whose right-hand elements are $r_{\gamma(i,j)} = -x_0^i y_0^j$ with $y_0 = p(x_0)/N$. The vector $\mathbf{r}$ and the matrix M were constructed in order to make $\mathbf{s}$ a short lattice point, with norm strictly less than 1. Indeed, $\mathbf{s}$ has left-hand elements given by $s_g = (x_0/X)^g/\sqrt{n}$, and right-hand elements equal to zero, as $s_{\gamma(i,j)} = q_{i,j}(x_0) - x_0^i y_0^j N^j$. In other words, the blocks B and C translate the polynomial modular equations $q_{i,j}(x)$. The fact that x_0 satisfies these equations makes the right-hand elements of $\mathbf{s}$ equal to zero. And the upper bound of (2) on the root x_0 is expressed by the block A. The diagonal coefficients "balance" the left-hand elements of $\mathbf{s}$.

In traditional lattice-based attacks, one would reduce the matrix M, and hope that the first vector of the reduced basis is equal to the target vector $\pm\mathbf{s}$. But Coppersmith notices that computing this vector explicitly is not necessary. Indeed, it suffices to confine the target vector in a subspace, which we now detail.

As the right-hand elements $n - \delta$ of the desired vector $\mathbf{s}$ are 0, we restrict our attention to the sublattice $\widehat{M}$ of M consisting of points with right-hand elements 0, namely $M \cap (\mathbb{R}^n \times \{0\}^{n-\delta})$. It is possible to compute explicitly this sublattice, by taking advantage of the fact that $p(x)$ and hence $q_{i,j}(x)$ are monic polynomials: certain $n-\delta$ rows of the block B form an upper triangular matrix with 1 on the diagonal. Thus, we can do elementary row operations on M to produce a block matrix $\widetilde{M}$ of the form:

$$\widetilde{M} = \begin{pmatrix} ? & 0 \\ ? & I \end{pmatrix},$$

where I is the $(n-\delta) \times (n-\delta)$ identity matrix. The $n \times n$ upper-left block represents the desired sublattice: an n-dimensional lattice, of which $\mathbf{s}$ is one relatively short element. In particular, M and $\widehat{M}$ have the same volume.

Next, we compute an LLL-reduced basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of the matrix $\widehat{M}$. From the theoretical bounds of the LLL algorithm and the value of the volume of $\widehat{M}$ (which can be bounded thanks to (3) and (2)), Coppersmith proved that any lattice point of norm strictly less than 1 must lie in the hyperplane spanned by $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n-1}$. In particular, $\mathbf{s}$ is such a lattice point. In terms of the larger matrix M, there is an n-dimensional space of vectors $\mathbf{r}$ such that $\mathbf{r}M = \mathbf{s}$ has 0's in its right-hand $n-\delta$ entries. And those integer vectors $\mathbf{r}$ which additionally satisfy $\mathbf{s} < 1$ must lie in a space of dimension one smaller, namely dimension $n-1$. This gives rise to a linear equation on the entries $r_g, 0 \leq g < n$. That is, we compute coefficients c_g such that: for any integer vector $\mathbf{r} = (r_g, r_{\gamma(i,j)})$ such that $\mathbf{s} = \mathbf{r}M$ has right-hand entries 0 and $\|\mathbf{s}\| < 1$, we must have $\sum c_g r_g = 0$. In particular:

$$\sum_{g=0}^{n-1} c_g x_0^g = 0.$$

This is a polynomial equation holding in $\mathbb{Z}$, not just modulo N. We can solve this polynomial for x_0 easily, using known techniques for solving univariate polynomial equations over $\mathbb{Z}$ (for instance, the Sturm sequence [11] suffices). This shows:

Theorem 1 (Coppersmith). *Let $p(x)$ be a polynomial of degree δ in one variable modulo an integer N of unknown factorization. Let X be the bound on the desired solution x_0. If $X < \frac{1}{2}N^{1/\delta-\varepsilon}$, then in time polynomial in $(\log N, \delta, 1/\varepsilon)$, we can find all integers x_0 with $p(x_0) \equiv 0 \pmod{N}$ and $|x_0| < X$.*

Corollary 2 (Coppersmith). *With the same hypothesis, except that $X \leq N^{1/\delta}$, then in time polynomial in $(\log N, 2^\delta)$, we can find all integers x_0 such that $p(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq X$.*

Proof. See [6]. The result is obtained by applying the previous theorem four times, with $\varepsilon = 1/\log_2 N$. $\square$

This is a major improvement over the bound $N^{2/[\delta(\delta+1)]}$ which was previously obtained in [17]. But, theoretically, one would *a priori* need the following parameters in order to achieve the theoretical bound $N^{1/\delta}$: $\varepsilon = 1/\log_2 N$ and $h \approx (\delta-1)\log_2 N/\delta^2$. For example, if we take $\delta = 3$ and a 512-bit number N, this means reducing several 341×341 matrices with entries at least as large as N^{h-1}, that is 17000-digit numbers ! Unfortunately, that appears to be a drawback of Coppersmith's improvement. Indeed, instead of using only the polynomial $p(x)$ (such as in [17]), Coppersmith introduced shifts and powers of this polynomial. This enlarges the volume of the lattice M, which is what makes the target vector more and more short compared to other lattice points, but at the expense of the size of the entries. In other words, the larger the entries are, the better the bound is supposed to be, and the more expensive the reduction is. This leads to several questions: is Coppersmith's method of any use in real life ? How much can we achieve in practice ? How do the practical bounds compare with the theoretical bounds ? We will answer these questions in Sections 4 and 5.

3 Applications to Low-Exponent RSA

We briefly review some applications of Coppersmith's method. More can be found in [6].

3.1 Stereotyped messages

Suppose the plaintext m consists of two pieces: a known piece $B = 2^k b$, and an unknown piece x. If this is RSA-encrypted with an exponent of 3, the ciphertext c is given by $c = m^3 = (B + x)^3 \pmod N$. If we know B, c and N we can apply the previous results to the polynomial $p(x) = (B + x)^3 - c$, and recover x_0 satisfying

$$p(x_0) = (B + x_0)^3 - c \equiv 0 \pmod N,$$

as long as such an x_0 exists with $|x_0| < N^{1/3}$. The attack works equally well if the unknown x_0 lies in the most significant bits of the message m rather than the least significant bits.

3.2 Random padding

Suppose two messages m and m' satisfy an affine relation, say $m' = m + r$. Suppose we know the RSA-encryptions of the two messages with an exponent of 3:

$$\begin{aligned} c &\equiv m^3 \pmod N \\ c' &\equiv (m')^3 \equiv m^3 + 3m^2 r + 3mr^2 + r^3 \pmod N \end{aligned}$$

We can eliminate m from the two equations above by taking their resultant, which gives a univariate polynomial in r of degree 9, modulo N:

$$r^9 + (3c - 3c')r^6 + (3c^2 + 21cc' + 3(c')^2)r^3 + (c - c')^3.$$

Thus, if $|r| < N^{1/9}$, we can theoretically recover r, from which we can derive the message $m = r(c' + 2c - r^3)/(c' - c + 2r^3) \pmod N$ (see [7]).

3.3 Broadcast attacks

As was pointed out in [15, 1], Coppersmith's result improves known results of Håstad [8]. We consider the situation of a broadcast application, where a user sends linearly related messages m_i to several participants with public exponent e_i and public modulus N_i. That is, $m_i \equiv \alpha_i m + \beta_i \pmod{N_i}$, for some unknown m and known constants α_i and β_i. This precisely happens if one sends a similar message with different (known) headers or time-stamps which are part of the encryption block.

Let $e = \max e_i$. If k such messages m_i are sent, the attacker obtains k polynomial equations $p_i(m) \equiv 0 \ (\mathrm{mod}\, N_i)$ of degree $e_i \leq e$. Then we use the Chinese Remainder Theorem to derive a polynomial equation of degree e:

$$p(m) \equiv 0 \pmod{N}, \text{where } N = \prod_{i=1}^{k} N_i.$$

And thus, by Coppersmith's method, we can theoretically recover m if $|m| < N^{1/e}$. In particular, this is satisfied if $k \geq e$. This improves the previous bound $k > e(e+1)/2$ obtained by Håstad.

4 Implementation

In Section 2, we saw that Coppersmith's method required reductions of high-dimensional lattices with huge entries. This is because the proof uses the parameter ε which induces a choice of h. Actually, ε is only of theoretical interest, as h is the natural parameter. In practice, one would rather choose h and ignore ε, so that the matrix and its entries are not too large. To compute the theoretical maximal rootsize (for a fixed h), one needs to look back at Coppersmith's proof. However, we will obtain this maximal rootsize from another method, due to Howgrave-Graham (see [9]). It can be shown that from a theoretical point of view, the two methods are strictly equivalent: they provide the same bounds, and they have the same complexity. But Howgrave-Graham's method is simpler to implement and to analyze, so that the practical behaviour of Coppersmith's method is easier to explain with this presentation.

4.1 Howgrave-Graham's method

We keep the notations of Section 2: a monic polynomial $p(x)$ of degree δ; a bound X for the desired solutions modulo N; and h a fixed integer. In both methods, one computes a polynomial $r(x)$ of degree at most $n = h\delta$ for which small modular roots of $p(x)$ are also integral roots of $r(x)$. In Coppersmith's method, such a polynomial is deduced from the hyperplane generated by the first vectors of a reduced basis of a certain n-dimensional lattice. In Howgrave-Graham's method, any sufficiently short vector of a certain n-dimensional lattice can be transformed into such a polynomial. Actually, these two lattices are related to each other by duality. Coppersmith uses lattice reduction to find a basis for which sufficiently short vectors are confined to the hyperplane generated by the first vectors of the basis. But this problem can also be viewed as a traditional short vector problem in the dual lattice, a fact that was noticed by both Howgrave-Graham [9] and Jutla [10].

Given a polynomial $r(x) = \sum a_i x^i \in \mathbb{Z}[x]$, define $\|r(x)\| = \sqrt{\sum a_i^2}$.

Lemma 3 (Howgrave-Graham). *Let $r(x) \in \mathbb{Z}[x]$ of degree n, and let X be a positive integer. Suppose $\|r(xX)\| < M/\sqrt{n}$. If $r(x_0) \equiv 0 \pmod M$ and $|x_0| < X$, then $r(x_0) = 0$ holds over the integers.*

Proof. Notice that $|r(x_0)| = |\sum a_i x_0^i| \leq \sum |a_i X^i| \leq \|r(xX)\|\sqrt{n} < M$. Since $r(x_0) \equiv 0 \pmod M$, it follows that $r(x_0) = 0$. □

The lemma shows that a convenient $r(x) \in \mathbb{Z}[x]$ is a polynomial with small norm having the same roots as $p(x)$ modulo N. We choose such a polynomial as an integer linear combination of the following polynomials (similar to the $q_{i,j}$'s of Coppersmith's method):

$$q_{u,v}(x) = N^{h-1-v} x^u f(x)^v.$$

Since x_0 is a root of $q_{u,v}(x)$ modulo N^{h-1}, $r(xX)$ must have norm less than $N^{h-1}/\sqrt{n}$ to use the lemma. But this can be seen as a short vector problem in the lattice corresponding to the $q_{u,v}(xX)$. So we define a lower triangular $n \times n$ matrix M whose i-th row consists of the coefficients of $q_{u,v}(xX)$, starting by the low-degree terms, where $v = \lfloor (i-1)/\delta \rfloor$ and $u = (i-1) - \delta v$. It can be shown that:

$$\det(M) = X^{n(n-1)/2} N^{n(h-1)/2}.$$

We apply an LLL-reduction to the lattice spanned by the rows of M. The first vector of the reduced basis corresponds to a polynomial of the form $r(xX)$. And its Euclidean norm is equal to $\|r(xX)\|$.

One the one hand, to apply the lemma, we need :

$$\|r(xX)\| \leq N^{h-1}/\sqrt{n}.$$

On the other hand, the theoretical bounds of the LLL algorithm guarantee that the norm of the first vector satisfies:

$$\|r(xX)\| \leq 2^{(n-1)/4} \det(M)^{1/n} \leq 2^{(n-1)/4} X^{(n-1)/2} N^{(h-1)/2}.$$

Therefore, a sufficient condition for the method to work is:

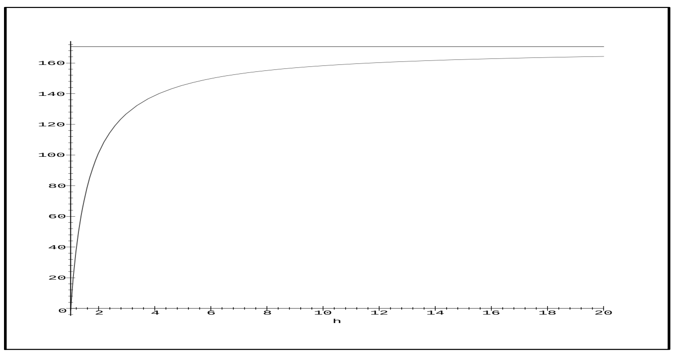
$$2^{(n-1)/4} X^{(n-1)/2} N^{(h-1)/2} \leq N^{h-1}/\sqrt{n}.$$

Hence, for a given h, the method is guaranteed to find modular roots up to X if:

$$X \leq \frac{1}{\sqrt{2}} N^{(h-1)/(n-1)} n^{-1/(n-1)} \tag{4}$$

This is also the expression found by Coppersmith in [6] (p. 241). And the limit of this expression, when h grows to ∞, is $\frac{1}{\sqrt{2}} N^{1/\delta}$. But what is worth noticing is that the logarithm of that expression, as a function of h, is quite concave (see Figure 1). This means that small values of h should already give results close to the limits. And hopefully, with a small h, the lattice is low-dimensional and its entries are not excessively large. This indicates that Coppersmith's method should be useful in real life. Fortunately, we will see that experiments confirm this prediction.

Fig. 1. Bit-length of the bound X for $\delta = 3$ and RSA-512, as a function of h.



4.2 Limits of the method

It is well-known that lattice reduction algorithms perform better in practice than theoretically expected. And when the LLL algorithm does not provide sufficiently short vectors, one can turn to improved lattice reduction algorithms such as Schnorr's [14]. However, a simple argument shows that Coppersmith's method and its variants are inherently limited, no matter how good the reduction algorithm is.

Indeed, if we assume that the lattice M to be reduced is "random", there are probably no lattice points of M significantly shorter than $\det(M)^{1/n}$, that is $X^{(n-1)/2} N^{(h-1)/2}$. And therefore, since the conditions of lemma 3 are quite tight, any lattice reduction algorithm will not detect roots much larger than:

$$N^{(h-1)/(n-1)} n^{-1/(n-1)}.$$

Compared to (4), only the factor $1/\sqrt{2}$ is removed, which is a very small improvement. Thus, it is likely that when the LLL algorithm fails to provide the solution, other lattice reduction algorithms will not help. The bound provided by (4) is probably tight.

4.3 Complexity

In both Coppersmith's method and Howgrave-Graham's method, the most expensive step is the lattice reduction step. The matrices to be reduced have the same dimension $n = h\delta$, and the size of their entries are similar. Therefore, from a theoretical point of view, the methods have the same complexity. We assume that X is chosen less than $N^{1/\delta}$.

The worst-case complexity of the LLL algorithm is $O(n^5 d \log^3 R)$ where n is the lattice dimension, d is the space dimension and R an upper bound for the squared norms of the basis vectors. So the method has worst-case complexity $O(n^6 \log^3 R)$ where R is an upper bound for all the $\|q_{u,v}(xX)\|^2$. We have:

$$\|q_{u,v}(xX)\|^2 = N^{2(h-1-v)} X^{2u} \|p(xX)^v\|^2.$$

All the coefficients of $p(xX)$ are less than N^2. It follows that:

$$\|p(xX)^v\|^2 \leq N^{4v} \|(1 + x + \cdots + x^\delta)^v\|^2 \leq N^{4v} (\delta + 1)^{2v}.$$

Therefore:

$$\|q_{u,v}(xX)\|^2 \leq N^{2(h-1+v)} X^{2u} (\delta + 1)^{2v} \leq N^{2h-4} X^{2\delta-2} (\delta + 1)^{2h-2}.$$

Thus, the complexity is $O(n^6[(2h - 4 + (2\delta - 2)/\delta) \log N + (2h - 2)(\delta + 1)]^3)$, that is:

$$O(h^9 \delta^6 [\log^3 N + \delta \log^2 N + \delta^2 \log N + \delta^3]).$$

For large N compared to δ, this is $O(h^9 \delta^6 \log^3 N)$. And that means large values of h and δ are probably not realistic. It also means that the running time of the method should be more sensitive to an increase of h, than an increase of δ, or an increase of the size of the modulus N.

5 Experiments

Our implementation uses the NTL library [16] of Victor Shoup. Due to the size of the entries, we had to use the floating point versions of reduction algorithms with extended exponent. Timings are given for a 500 MHz DEC Alpha. We used two sorts of computers: 64-bit 500 MHz DEC Alpha using Linux and 64-bit 270 MHz Sparc Ultra-2i using Solaris. It is worth noticing that for large reductions, the Alpha was about 6 times faster than the Ultra. In part, this is because we were able to use a 64-bit compiler for the Alpha, but not for the Ultra; and the clock frequency of the Alpha is twice as high than the one of the Ultra.

We implemented both Coppersmith's method and its variant by Howgrave-Graham. The running times and the results are very similar, but Howgrave-Graham's method is simpler to implement. Therefore, the tables given here hold for both methods.

5.1 Running times

Tables 1, 2 and 3 show the running time of the reduction stage, as a function of the parameter h and the polynomial degree δ, for different sizes of moduli. The polynomial was randomly chosen. The other parts of the method, such as computing the integral roots of the polynomial found, are negligible compared to the reduction stage.

In Section 4, we saw that the worst-case complexity was $O(h^9\delta^6\log^3 N)$. The running times confirm that an increase in h is more expensive than an increase in δ. But the dominant factor is $n = h\delta$. If δ is not small, only small values of h are realistic. And if h is chosen large, only small values of δ are possible.

Doubling the size of the modulus from RSA-512 to RSA-1024 roughly multiplies by 5 the running times. And doubling the size of the modulus from RSA-1024 to RSA-2048 roughly multiplies by 5.5 the running times. From the complexity, one would expect a multiplication by 8. It turns out that the method is practical even for very large N. And therefore, one would expect broadcast attacks with small exponent to be practical, as they multiply the size of the modulus by the number of linearly related messages, but keep the (low) polynomial degree unchanged.

Table 1. Running time (in seconds), as a function of h and δ, for RSA-512.

Parameter h	Polynomial degree δ								
	2	3	4	5	6	7	8	9	10
2	0	0.04	0.12	0.29	0.57	0.98	1.71	2.8	4.4
3	0.07	0.34	1.02	2.66	5.71	11	21	36	56
4	0.27	1.48	5.09	14	33	64	120	191	318
5	0.84	4.99	19	53	123	242	455	773	1170
6	2.21	14	55	161	368	764	1395	2341	3773
7	5.34	37	150	415	919	1868	3417	6157	9873
8	11	82	331	912	2146	4366	7678	13725	21504
9	21	166	646	1838	4464	8777	17122	27314	42212
10	38	323	1234	3605	8343	15997	30600	47612	84864
11	70	598	2239	6989	15881	30488	50866	91100	
12	126	994	4225	11650	25637	48249	91824		
13	194	1582	6598	17892	44616	82290	158051		
14	311	2498	10101	29785	65267				
15	474	4097	15835	41856					
16	714	5857	25059	64158					
17	1066	9296	34365	96503					
18	1437	12431	50095						

Table 2. Running time (in seconds), as a function of h and δ, for RSA-1024.

Parameter h	Polynomial degree δ								
	2	3	4	5	6	7	8	9	10
2	0.03	0.13	0.37	0.83	1.68	3.02	5.17	8.53	13
3	0.24	1.19	3.76	9.19	21	42	76	128	209
4	1.16	5.89	21	57	134	270	492	813	1306
5	3.69	21	82	238	541	1111	2030	3426	5745
6	9.68	66	264	752	1736	3423	6272	11064	17040
7	25	175	699	2017	4521	9266	17746		
8	53	392	1623	4748	10858	21662			
9	103	815	3277	9800	22594	44712			
10	204	1605	6512	18608					
11	364	2813	11933						
12	627	5191	20947						
13	1028	8530							

Table 3. Running time (in seconds), as a function of h and δ, for RSA-2048.

Parameter h	Polynomial degree δ								
	2	3	4	5	6	7	8	9	10
2	0.09	0.46	1.29	3.01	5.93	11	19	31	48
3	1.04	5.11	16	42	93	187	343	598	928
4	5.21	29	97	277	635	1308	2386	4151	6687
5	19	107	405	1185	2780	5616	10584	17787	28458
6	52	337	1355	3922	9129	18776			
7	123	920	3729	10697	25087				
8	282	2122	8697	25089	58258				
9	555	4503	18854	53345					
10	1072	9313	36468						
11	2008	16042	68669						
12	3499	28187							
13	5796								

5.2 Experimental bounds

For a given choice of h and δ, one can theoretically find roots as large as $X = N^{(h-1)/(n-1)}n^{-1/(n-1)}/\sqrt{2}$, where $n = h\delta$. However, in practice, one has to use floating point versions of lattice reduction algorithms, because exact versions (using only integer arithmetic) are quite expensive, especially with this size of entries. This means that the basis obtained is not guaranteed to be LLL-reduced, and therefore, the upper bound X cannot be guaranteed either. But, in practice, in all our experiments, the basis obtained was always LLL-reduced, and thus, we have always been able to find roots as large as the bound. Approximation problems occur only when the lattice dimension is very high (larger than say, 150), which was not the case here. When the LLL algorithm failed to provide a sufficiently short vector, we applied improved lattice reduction algorithms. But as expected (see the previous section), it did not help: the method is inherently limited by the value of the lattice determinant.

We only made experiments with the case of an RSA encryption using 3 as a public exponent. Coppersmith-like attacks are useful only for a very small exponent such as 3, because the polynomial degree must be very small for efficiency, and the roots cannot be much larger than the size of the modulus divided by the polynomial degree. For instance, a public exponent of 65537 is not threatened by Coppersmith's method. One should also note that these attacks do not recover the secret factorization: they can only recover the plaintext under specific conditions.

Stereotyped messages. This case corresponds to $\delta = 3$. Table 4 give the bounds obtained in practice, and the corresponding running times. The bound of (4) is tight: we never obtained an experimental bound X more than twice as large as the theoretical bound. There is a value of h which gives the best compromise between the maximal rootsize and the running time. Of course, this value depends on the implementation. If one wants to compute roots larger than the corresponding rootsize, one should treat the remaining bits by exhaustive search, rather than by increasing h. Here, this value seems to be slightly larger than 13.

Table 4. Bounds and running time for stereotyped messages

Size of N	Data type	Parameter h												
		2	3	4	5	6	7	8	9	10	11	12	13	∞
512	Size of X	102	128	139	146	150	153	156	157	159	160	161	162	170
	Seconds	0.05	0.36	1.54	5	15	36	82	161	308	542	910	1501	
768	Size of X	153	192	209	219	226	230	234	236	238	240	241	242	256
	Seconds	0.09	0.76	3.39	12	35	90	211	418	853	1490	2563	4428	
1024	Size of X	204	256	279	292	301	307	311	315	318	320	322	323	341
	Seconds	0.14	1.28	6	23	66	179	393	823	1634	3044	5254	9224	

Random padding. This case corresponds to $\delta = 9$. Table 5 give the bounds obtained in practice, and the corresponding running times. Note that for this case, the experimental bound X had a few bits more than the theoretical bound for small values of h, which is why we added new data in the table. Again, there is a value of h which gives the best compromise between the maximal rootsize and the running time. This value seems to be $h = 6$ for RSA-512 and RSA-768, and $h = 7$ for RSA-1024. In all these cases, the running time is less than than a few minutes, and the corresponding rootsize is not far from the maximal theoretical rootsize (corresponding to $h = \infty$).

Note that the running time is significantly less than the one given in tables 1, 2 for $\delta = 9$. This is because the polynomial of degree 9 is of particular form here, as it is quite sparse.

Table 5. Bounds and running time for random padding

Size of N	Data type	Parameter h									
		2	3	4	5	6	7	8	9	10	∞
512	Experimental size of X	34	42	46	48	50	51	51	52	52	
	Theoretical size of X	30	39	44	46	48	49	50	51	52	57
	Seconds	0.28	2.07	8	29	76	190	396	769	1307	
768	Experimental size of X	51	63	69	73	75	76	77	78	79	
	Theoretical size of X	45	59	66	70	72	75	76	77	77	85
	Seconds	0.46	3.76	17	55	163	396	835	1713	3095	
1024	Experimental size of X	68	85	93	97	100	102	103	104	105	
	Theoretical size of X	60	79	88	93	96	99	101	102	103	114
	Seconds	0.74	6	28	97	298	733	1629	3468	6674	

Broadcast applications. We consider the situation of a broadcast application, where a user sends k linearly related messages m_i (built from an unknown message m) to several participants with public exponent $e_i \leq e$ and public modulus N_i. Theoretically, Coppersmith's method should recover the message m, as soon as $k \geq e$. The problem is that the case $k = e$ corrresponds to a large value of h, which is unrealistic in practice, as shown in Table 3. Table 6 give the bounds obtained in practice, and the corresponding running times for a public exponent of 3 (which corresponds to $\delta = 3$), depending on the number of linearly related messages and the size of the modulus N. When one allows $e + 1$ messages, the attack becomes practical. We have always been able to recover the message when $e = 3$ and 4 messages are sent, with a choice of $h = 4$ (the value is $h = 3$ is a bit tight). The corresponding running time is only a few minutes, even with RSA-1024. For larger exponents (and thus, a larger number of necessary messages), the method does not seem to be practical, as the running time is very sensitive to the polynomial degree δ and the parameter h.

Table 6. Bounds and running time for broadcast attacks with public exponent 3

Size of N	Messages	Data type	Parameter h					
			2	3	4	5	6	∞
512	2	Size of X	204	256	279	292	301	341
		Seconds	0.12	1.27	6	22	67	
	3	Size of X	307	384	419	439	452	511
		Seconds	0.28	2.85	15	55	164	
	4	Size of X	409	512	558	585	602	682
		Seconds	0.45	5	28	104	329	
768	2	Size of X	307	384	419	439	452	511
		Seconds	0.27	2.72	15	55	169	
	3	Size of X	460	576	628	658	677	767
		Seconds	0.57	7	36	135	435	
	4	Size of X	614	768	837	877	903	1022
		Seconds	0.93	12	67	272	873	
1024	2	Size of X	409	512	558	585	602	682
		Seconds	0.48	5	29	107	340	
	3	Size of X	614	768	837	877	903	1024
		Seconds	1.41	13	71	283	896	
	4	Size of X	819	1024	1117	1170	1204	1364
		Seconds	1.89	25	144	567	1793	

6 Conclusion

We presented extensive experiments with lattice-based attacks against RSA with low public exponent, which validate Coppersmith's novel approach to find small roots of a univariate modular polynomial equation. In practice, one can, in a reasonable time, achieve bounds fairly close to the theoretical bounds. We also showed that these theoretical bounds are essentially tight, in the sense that one cannot expect to obtain significantly better results in practice, regardless of the lattice reduction algorithm used.

The experiments confirm that sending stereotyped messages with a small public exponent e is dangerous when the modulus size is larger than e times the size of the hidden part (consecutive bits). Random padding with public exponent 3 is also dangerous, as while as the modulus size is larger than 9 times the padding size. Interestingly, Håstad-like attacks are practical: if a user sends 4 linearly related messages encrypted with public exponent 3, then one can recover the unknown message in a few minutes, even for 1024-bit modulus. Note that this improves the former theoretical bound of 7 messages obtained by Håstad. For 3 messages, one can recover the message if the unknown part has significantly less bits than the modulus.

This stresses the problems of the low-exponent RSA encryption scheme. However, it only applies to the case of very small public exponents such as 3. It does not seem to threaten exponents such as 65537. And these attacks do not seem to apply to the RSA signature scheme with a small validating exponent.

Acknowledgements. We would like to thank the anonymous referees for their helpful comments.

References

1. D. Bleichenbacher. On the security of the KMOV public key cryptosystem. In *Proc. of Crypto '97*, volume 1294 of *LNCS*, pages 235–248. Springer-Verlag, 1997.
2. D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 1998. To appear. Available at `http://theory.stanford.edu/~dabo/`.
3. D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In *Proc. of Eurocrypt '98*, volume 1233 of *LNCS*, pages 59–71. Springer-Verlag, 1998.
4. D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Proc. of Eurocrypt '96*, volume 1070 of *LNCS*, pages 178–189. Springer-Verlag, 1996.
5. D. Coppersmith. Finding a small root of a univariate modular equation. In *Proc. of Eurocrypt '96*, volume 1070 of *LNCS*, pages 155–165. Springer-Verlag, 1996.
6. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. of Cryptology*, 10(4):233–260, 1997.
7. D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. In *Proc. of Eurocrypt '96*, volume 1070 of *LNCS*, pages 1–9. Springer-Verlag, 1996.
8. J. Hastad. Solving simultaneous modular equations of low degree. *SIAM J. Comput.*, 17(2):336–341, April 1988.
9. N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding*, volume 1355 of *LNCS*, pages 131–142. Springer-Verlag, 1997.
10. C. S. Jutla. On finding small solutions of modular multivariate polynomial equations. In *Proc. of Eurocrypt '98*, volume 1233 of *LNCS*, pages 158–170. Springer-Verlag, 1998.
11. D. Knuth. *The Art of Computer Programming vol. 2: Seminumerical Algorithms.* Addison-Wesley, 1981. Section 4.6.1.
12. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.
13. R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
14. C.-P. Schnorr. A hierarchy of polynomial lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.
15. H. Shimizu. On the improvement of the Håstad bound. In *1996 IEICE Fall Conference*, volume A-162, 1996. In Japanese.
16. V. Shoup. Number Theory C++ Library (NTL) version 3.1. Can be obtained at `http://www.cs.wisc.edu/~shoup/ntl/`.
17. B. Vallée, M. Girault, and P. Toffin. How to guess ℓ-th roots modulo n by reducing lattice bases. In *Proc. of AAECC-6*, volume 357 of *LNCS*, pages 427–442. Springer-Verlag, 1988.