Computing the RSA Secret Key is Deterministic Polynomial Time Equivalent to Factoring

Alexander May

Faculty of Computer Science, Electrical Engineering and Mathematics
University of Paderborn
33102 Paderborn, Germany
alexx@uni-paderborn.de

Abstract. We address one of the most fundamental problems concerning the RSA cryptoscheme: Does the knowledge of the RSA public key/ secret key pair (e, d) yield the factorization of $N = pq$ in polynomial time? It is well-known that there is a *probabilistic* polynomial time algorithm that on input (N, e, d) outputs the factors p and q. We present the first *deterministic* polynomial time algorithm that factors N provided that $e, d < \phi(N)$ and that the factors p, q are of the same bit-size. Our approach is an application of Coppersmith's technique for finding small roots of bivariate integer polynomials.

Keywords: RSA, Coppersmith's method

1 Introduction

One of the most important tasks in public key cryptography is to establish the polynomial time equivalence of

- the problem of computing the secret key from the public information to
- a well-known hard problem P that is believed to be computational infeasible.

This reduction establishes the security of the secret key under the assumption that the problem P is computational infeasible. On the other hand, such a reduction does not provide any security for a public key system itself, since there might be ways to break a system without computing the secret key.

Now let us look at the RSA scheme. We briefly define the RSA parameters: Let $N = pq$ be a product of two primes of the same bit-size. Furthermore, let e, d be integers such that $ed = 1 \bmod \phi(N)$, where $\phi(N)$ is Euler's totient function.

For the RSA scheme, we know that there exists a *probabilistic* polynomial time equivalence between the secret key computation and the problem of factoring the modulus N. The proof is given in the original RSA paper by Rivest, Shamir and Adleman [9] and is based on a work by Miller [8].

In this paper, we present a *deterministic* polynomial time algorithm that on input (N, e, d) outputs the factors p, q, provided that p and q are of the same bit-size and that

$$ed \leq N^2.$$

In the normal RSA-case we have $e, d < \phi(N)$, since e, d are defined modulo $\phi(N)$. This implies that $ed \leq N^2$ as required. Thus, our algorithm establishes the *deterministic* polynomial time equivalence between the secret key computation and the factorization problem in the most common RSA case. We reduce the problem of factoring N to the problem of computing d, the reduction in the opposite direction is trivial.

Our approach is an application of Coppersmith's method [4] for finding small roots of bivariate integer polynomials. We want to point out that some cryptanalytic results [1, 2] are based on Coppersmith's technique for solving *modular* bivariate polynomial equations. In contrast to these, we make use of Coppersmith's algorithm for bivariate polynomials with a small root over the integers. Therefore, our result does not depend on the usual heuristic for modular multivariate polynomial equations but is rigorous.

To the best of our knowledge, the only known application of Coppersmith's method for bivariate polynomials with a root over the integers is the so-called "factoring with high bits known" [4]: Given half of the most significant bits of p, one can factor N in polynomial time. Howgrave-Graham [6] showed that this problem can be solved alternatively using an univariate modular approach (see also [5]).

Since our approach directly uses Coppersmith's method for bivariate integer polynomials, the proof of our reduction is brief and simple.

The paper is organized as follows. First, we present in Sect. 2 a deterministic polynomial time algorithm that factors N on input (N, e, d) provided that $ed \leq N^{\frac{3}{2}}$. This more restricted result is interesting, since RSA is frequently used with small e in practice. Additionally, we need only elementary arithmetic in order to prove the result. As a consequence, the underlying algorithm has running time $\mathcal{O}(\log^2 N)$.

Second, we show in Sect. 3 how to improve the previous result to the desired bound $ed \leq N^2$ by applying Coppersmith's method for solving bivariate integer polynomials. We conclude by giving experimental results in Sect. 4.

2 An Algorithm for $ed \leq N^{\frac{3}{2}}$

In this work, we always assume that N is a product of two different prime factors p, q of the same bitsize, wlog $p < q$. This implies

$$p < N^{\frac{1}{2}} < q < 2p < 2N^{\frac{1}{2}}.$$

We obtain the following useful estimates:

$$p + q < 3N^{\frac{1}{2}} \quad \text{and} \quad \phi(N) = N + 1 - (p + q) > \frac{1}{2}N.$$

Let us denote by $\lceil k \rceil$ the smallest integer greater or equal to k. Furthermore, we denote by $\mathbb{Z}^*_{\phi(N)}$ the ring of invertible integers modulo $\phi(N)$.

In the following theorem, we present a very efficient algorithm that on input (N, e, d) outputs the factors of N provided that $ed \leq N^{\frac{3}{2}}$.

Theorem 1 *Let $N = pq$ be an RSA-modulus, where p and q are of the same bit-size. Suppose we know integers e, d with $ed > 1$,*

$$ed = 1 \bmod \phi(N) \quad \textit{and} \quad ed \leq N^{\frac{3}{2}}.$$

Then N can be factored in time $\mathcal{O}(\log^2 N)$.

Proof: Since $ed = 1 \bmod \phi(N)$, we know that

$$ed = 1 + k\phi(N) \quad \text{for some } k \in \mathbb{N}.$$

Next, we show that k can be computed up to a small constant for our choice of e and d. Therefore, let us define $\tilde{k} = \frac{ed-1}{N}$ as an underestimate of k. We observe that

$$\begin{aligned} k - \tilde{k} &= \frac{ed-1}{\phi(N)} - \frac{ed-1}{N} \\ &= \frac{N(ed-1) - (N-p-q+1)(ed-1)}{\phi(N)N} \\ &= \frac{(p+q-1)(ed-1)}{\phi(N)N} \end{aligned}$$

Using the inequalities $p + q - 1 < 3N^{\frac{1}{2}}$ and $\phi(N) \geq \frac{1}{2}N$, we conclude that

$$k - \tilde{k} < 6N^{-\frac{3}{2}}(ed-1). \tag{1}$$

Since $ed \leq N^{\frac{3}{2}}$, we know that $k - \tilde{k} < 6$. Thus, one of the six values $\lceil \tilde{k} \rceil + i$, $i = 0, 1, \ldots 5$ must be equal to k. We test these six candidates successively. For the right choice k, we can compute

$$N + 1 + \frac{1-ed}{k} = p + q.$$

From the value $p + q$, we can easily find the factorization of N.

Our approach uses only elementary arithmetic on integers of size $\log(N)$. Thus, the running time is $\mathcal{O}(\log^2 N)$ which concludes the proof of the theorem. □

3 The Main Result

In this section, we present a polynomial time algorithm that on input (N, e, d) outputs the factorization of N provided that $ed \leq N^2$. This improves upon the result of Theorem 1. However, the algorithm is less efficient, especially when we get close to the bound N^2.

Our approach makes use of the following result of Coppersmith [4] for finding small roots of bivariate integer polynomials.

Theorem 2 (Coppersmith) *Let $f(x, y)$ be an irreducible polynomial in two variables over $\mathbb{Z}$, of maximum degree δ in each variable separately. Let X, Y be bounds on the desired solution (x_0, y_0). Let W be the absolute value of the largest entry in the coefficient vector of $f(xX, yY)$. If*

$$XY \leq W^{\frac{2}{3\delta}}$$

then in time polynomial in $\log W$ and 2^δ we can find all integer pairs (x_0, y_0) with $f(x_0, y_0) = 0$, $|x_0| \leq X$ and $|y_0| \leq Y$.

Now, let us prove our main theorem.

Theorem 3 *Let $N = pq$ be an RSA-modulus, where p and q are of the same bit-size. Suppose we know integers e, d with $ed > 1$,*

$$ed = 1 \bmod \phi(N) \quad \textit{and} \quad ed \leq N^2.$$

Then N can be factored in time polynomial in the bit-size of N.

Proof: Let us start with the equation

$$ed = 1 + k\phi(N) \quad \text{for some } k \in \mathbb{N}. \tag{2}$$

Analogous to the proof of Theorem 1, we define the underestimate $\tilde{k} = \frac{ed-1}{N}$ of k. Using (1), we know that

$$k - \tilde{k} < 6N^{-\frac{3}{2}}(ed - 1) < 6N^{\frac{1}{2}}.$$

Let us denote $x = k - \tilde{k}$. Therefore, we have an approximation $\tilde{k}$ for the unknown parameter k in (2) up to an additive error of x.

Next, we also want to find an approximation for the second unknown parameter $\phi(N)$ in (2). Note that

$$N - \phi(N) = p + q - 1 < 3N^{\frac{1}{2}}.$$

That is, $\phi(N)$ lies in the interval $[N - 3N^{\frac{1}{2}}, N]$. We can easily guess an estimate of $\phi(N)$ with additive error at most $\frac{1}{4}N^{\frac{1}{2}}$ by doing a brute-force search on the most significant bits of $\phi(N)$.

More precisely, we divide the interval $[N - 3N^{\frac{1}{2}}, N]$ into 6 sub-interval of length $\frac{1}{2}N^{\frac{1}{2}}$ with centers $N - \frac{2i-1}{4}N^{\frac{1}{2}}$, $i = 1, 2, \ldots, 6$. For the correct choice of i we have

$$\left| N - \frac{2i-1}{4}N^{\frac{1}{2}} - \phi(N) \right| \leq \frac{1}{4}N^{\frac{1}{2}}.$$

Let g denote the term $\frac{2i-1}{4}N^{\frac{1}{2}}$ for the right choice of i. That is, we know $\phi(N) = N - g - y$ for some unknown y with $|y| \leq \frac{1}{4}N^{\frac{1}{2}}$.
Plugging our approximations for k and $\phi(N)$ in (2) leads to

$$ed - 1 - (\tilde{k} + x)(N - g - y) = 0.$$

Let us round $\tilde{k}$ and g to the next integers. Here we omit the rounding brackets $\lceil\tilde{k}\rceil, \lceil g\rceil$ for ease of simplicity. Notice that the effect of this rounding on the bounds of the estimation errors x and y can be neglected (x becomes even smaller). Thus, we assume in the following that $\tilde{k}$, g are integers. Therefore, we can define the following bivariate integer polynomial

$$f(x,y) = xy - (N-g)x + \tilde{k}y - \tilde{k}(N-g) + ed - 1$$

with a root $(x_0, y_0) = (k - \tilde{k}, p + q - 1 - g)$ over the integers.

In order to apply Coppersmith's theorem (Theorem 2), we have to bound the size of the root (x_0, y_0). We define $X = 6N^{\frac{1}{2}}$ and $Y = \frac{1}{4}N^{\frac{1}{2}}$. Then, $|x_0| \leq X$ and $|y_0| \leq Y$.

Let W denote the ℓ_∞-norm of the coefficient vector of $f(xX, yY)$. We have

$$W \geq (N-g)X \geq 3N^{\frac{3}{2}}.$$

By Coppersmith's theorem, we have to satisfy the condition $XY \leq W^{\frac{2}{3}}$. Using our bounds, we obtain

$$XY = \frac{3}{2}N < \left(3N^{\frac{3}{2}}\right)^{\frac{2}{3}} \leq W^{\frac{2}{3}}.$$

Thus, we can find the root (x_0, y_0) in time polynomial in the bit-size of W using Coppersmith's method. Note that the running time is also polynomial in the bit-size of N since $W \leq NX = 6N^{\frac{3}{2}}$. Finally, the term $y_0 = p + q - 1 - g$ yields the factorization of N. This concludes the proof of the theorem. $\square$

We want to point out that Theorem 3 can be easily generalized to the case, where $p + q \leq \mathrm{poly}(\log N) \cdot N^{\frac{1}{2}}$. I.e., we do not necessarily need that p and q are of the same bit-size. All that we have to require is that they are balanced up to some polylogarithmic factor in N.

The following theorem is a direct consequence of Theorem 3. It establishes the polynomial time equivalence of computing d and factoring N in the common RSA case, where $e, d \in \mathbb{Z}^*_{\phi(N)}$.

Theorem 4 *Let $N = pq$ be an RSA-modulus, where p and q are of the same bit-size. Furthermore, let $e \in \mathbb{Z}^*_{\phi(N)}$ be an RSA public exponent.*

Suppose we have an algorithm that on input (N, e) outputs in deterministic polynomial time the RSA secret exponent $d \in \mathbb{Z}^*_{\phi(N)}$ satisfying $ed = 1 \bmod \phi(N)$. Then N can be factored in deterministic *polynomial time.*

4 Experiments

We want to provide some experimental results. We implemented the algorithm introduced in the previous section on an 1GHz Linux-PC. Our implementation of Coppersmith's method follows the description given by Coron [4]. L^3-lattice reduction [7] is done using Shoup's NTL library [10].

We choose $e < \phi(N)$ randomly. Therefore, in every experiment the product ed is very close to the bound N^2. Notice that in Theorem 3, we have to do a small brute-force search on the most significant bits of $\phi(N)$ in order to prove the desired bound. The polynomial time algorithm of Coppersmith given by Theorem 2 requires a similar brute-force search on the most significant bits.

In Table 1, we added a column that states the total number c of bits that one has to guess in order to find a sufficiently small lattice vector. Thus, we have to multiply the running time of the lattice reduction algorithm by a factor of 2^c. As the results indicate, the number c heavily depends on the lattice dimension. Coppersmith's technique yields a polynomial time algorithm when the lattice dimension is of size $\theta(\log W)$. However, we only tested our algorithm for lattices of small fixed dimensions 16, 25 and 36.

Our experiments compare well to the experimental results of Coron [3]: One cannot come close to the bounds of Coppersmith's theorem without reducing lattices of large dimension. Notice that we have to guess a large number of bits. In contrast, by the proof of Coppersmith's theorem (see [4]) the number of bits that one has to guess for lattice dimension $\theta(\log W)$ is a small constant. However, it is a non-trivial task to handle lattices of these dimensions in practice.

Table 1. Results for $ed \approx N^2$

N	c	dim	L^3-time
512 bit	55 bit	16	0.5 min
512 bit	43 bit	25	6 min
512 bit	36 bit	36	53 min
768 bit	80 bit	16	1 min
768 bit	63 bit	25	13 min
768 bit	53 bit	36	128 min
1024 bit	105 bit	16	2.5 min
1024 bit	82 bit	25	26 min
1024 bit	67 bit	36	242 min

One might conclude that our method is of purely theoretical interest. But let us point out that we have a worst case for our approach when the product ed is very close to the bound N^2. In Table 2, we provide some more practical results for the case $ed \approx N^{1.75}$.

Table 2. Results for $ed \approx N^{1.75}$

N	c	dim	L^3-time
512 bit	10 bit	25	6 min
768 bit	13 bit	25	13 min
1024 bit	18 bit	25	26 min

References

1. D. Boneh, G. Durfee, "Cryptanalysis of RSA with private key d less than $N^{0.292}$", IEEE Trans. on Information Theory, Vol. 46(4), pp. 1339–1349, 2000
2. J. Blömer, A. May, "New Partial Key Exposure Attacks on RSA", Advances in Cryptology – Crypto 2003, Lecture Notes in Computer Science Vol. 2729, pp. 27–43, Springer-Verlag, 2003
3. Jean-Sébastien Coron, "Finding Small Roots of Bivariate Integer Polynomial Equations Revisited", Advances in Cryptology – Eurocrypt '04, Lecture Notes in Computer Science Vol. 3027, pp. 492–505, Springer-Verlag, 2004
4. D. Coppersmith, "Small solutions to polynomial equations and low exponent vulnerabilities", Journal of Cryptology, Vol. 10(4), pp. 223–260, 1997.
5. D. Coppersmith, "Finding Small Solutions to Small Degree Polynomials", Cryptography and Lattice Conference (CaLC 2001), Lecture Notes in Computer Science Volume 2146, Springer-Verlag, pp. 20–31, 2001.
6. N. Howgrave-Graham, "Finding small roots of univariate modular equations revisited", Proceedings of Cryptography and Coding, Lecture Notes in Computer Science Vol. 1355, Springer-Verlag, pp. 131–142, 1997
7. A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," Mathematische Annalen, Vol. 261, pp. 513–534, 1982
8. G. L. Miller, "Riemann's hypothesis and tests for primality", Seventh Annual ACM Symposium on the Theory of Computing, pp. 234–239, 1975
9. R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, Vol. 21(2), pp.120–126, 1978
10. V. Shoup, NTL: A Library for doing Number Theory, online available at `http://www.shoup.net/ntl/index.html`