Applications of Diophantine approximations algorithms in cryptanalysis of RSA

Andrej Dujella

Department of Mathematics, Faculty of Science,
University of Zagreb, Croatia
Center of Excellence QuantiXLie
Croatian Academy of Sciences and Arts
Doctor Honoris Causa of University of Debrecen
e-mail: duje@math.hr
URL: http://web.math.pmf.unizg.hr/~duje/

Public Key Cryptography

The classical situation in cryptography is that two persons - ALICE and BOB - wish to perform some form of communication, while an eavesdropper - EVE - wishes to spy the communication between Alice and Bob. There is no assumption that Alice and Bob (or Eve) are human, they may be computers on some network.

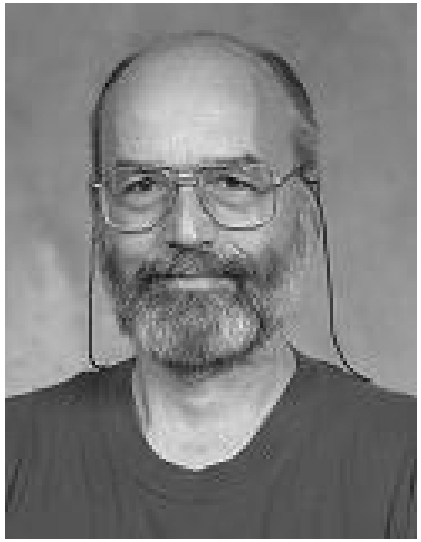
In classical model of cryptography, Alice and Bob secretly choose the key K, which then gives an encryption rule e_K and a decryption rule d_K, and d_K is either the same as e_K, or easily derived from it. Cryptosystems of this type are called *private-key* systems or *symmetric* systems. One drawback of this system is that it requires prior communication of the key K between Alice and Bob, using a secure channel. This may be very difficult to achieve.

The idea of a *public-key* cryptography is that it might be possible to find a cryptosystem where it is computationally infeasible to determine d_K from e_K. If so, then the encryption rule e_K could be made public by publishing it in a directory. Now Alice (or anyone else) can send an encrypted message to Bob (without prior communication of a secret key) by using public encryption rule e_K. But Bob will be the only person that can decrypt the ciphertext, using his secret (private) decryption rule d_K.

The idea of a public-key system was introduced in 1976 by **Diffie** and **Hellman**.

The first realization of a public-key system was proposed in 1977 by **Rivest**, **Shamir** and **Adleman** – **RSA Cryptosystem**, and its security is based on the problem of factorization of large integers.

Ronald Rivest



Adi Shamir

Leonard Adleman

Description of RSA:

Each user chooses two primes p and q, and sets $n = p \cdot q$.
Knowing factorization of n, it is easy to compute

$$\varphi(n) = (p-1)(q-1) = n + 1 - p - q.$$

Next, the user chooses an integer e between 1 and $\varphi(n)$ such that $\gcd(e, \varphi(n)) = 1$.
He computes the multiplicative inverse of e modulo $\varphi(n)$ by Euclidean algorithm:

$$d \cdot e \equiv 1 \pmod{\varphi(n)}.$$

The values n and e are public, while the values p, q and d are secret.
Encryption: $x \mapsto x^e \bmod n$.
Decryption: $y \mapsto y^d \bmod n$.

If the RSA Cryptosystem is to be secure, it is necessary that $n = pq$ must be large enough that factoring it will be computationally infeasible. Hence, it is recommended that one should choose p and q to each be primes having at least 150 digits.

There are basically two types of factoring algorithms: special-purpose (use special features of the number n) and general-purpose (depend only on the size of n). The special-purpose algorithms suggest which kind of numbers n (i.e. p and q) should be avoided. E.g. if p and q are very close to each other, then they can be discovered by testing numbers near $\sqrt{n}$ (Fermat's factorization). Also, if $p-1$ or $q-1$ have only small prime factors (they are "smooth"), then Pollard's $p-1$ factorization algorithm can be efficient.

Diophantine approximations: how well a given (irrational) number can be approximated by rational numbers, when the quality of approximation is given in terms of denominator of the rational numbers.

The classical **Dirichlet's theorem** says that for any irrational number α there exist infinitely many rational numbers p/q such that $|\alpha - p/q| < 1/q^2$.

The rational approximations p/q with this property can be obtained using the continued fractions.

Idea for cryptanalytic attacks: build α from the public key components and p/q from the secret key components, and use results and algorithms from Diophantine approximations (continued fractions and generalizations) to determine (candidates for) p/q from α and the quality of approximation.

Continued fractions:

$$\alpha = [a_0; a_1, a_2, \ldots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{}{\ddots}}}.$$

$a_0 = \lfloor \alpha \rfloor$, $\alpha = a_0 + \frac{1}{\alpha_1}$, $a_1 = \lfloor \alpha_1 \rfloor$, etc.

Convergents of continued fraction expansion:
$\frac{p_m}{q_m} = [a_0, a_1, \ldots, a_m]$.

$p_m = a_m p_{m-1} + p_{m-2}$, $q_m = a_m q_{m-1} + q_{m-2}$.

Lagrange: If $|\alpha - \frac{p}{q}| < \frac{1}{2q^2}$, then $\frac{p}{q} = \frac{p_m}{q_m}$ for an integer $m \geq 0$.

To speed up the RSA encryption or decryption one may try to use small public encryption exponent e or secret decryption exponent d. The choice of a small e or d is especially interesting when there is a large difference in computing power between two communicating devices, e.g. in communication between a smart card and a larger computer.

In this situation, it would be desirable:

smart card - small secret exponent d

larger computer - small public exponent e

to reduce the processing required in the smart card.

Attacks on RSA with small secret exponent

Wiener's attack (1990):

$ed - k\varphi(n) = 1, \quad \varphi(n) \approx n \quad \Rightarrow \quad \frac{k}{d} \approx \frac{e}{n}$

Assume that $p < q < 2p$. If $d < \frac{1}{3}n^{0.25}$, then

$$\left|\frac{e}{n} - \frac{k}{d}\right| < \frac{1}{2d^2}.$$

By classical Legendre's theorem, d is the denominator of some convergent p_m/q_m of the continued fraction expansion of e/n, and therefore d can be computed efficiently from the public key (n, e).

Total number of convergents is of order $O(\log n)$; a convergent can be tested in polynomial time.

Verheul & van Tilborg (1997): An extension of Wiener's attack that allows the RSA cryptosystem to be broken when d is a few bits longer than $n^{0.25}$. For $d > n^{0.25}$ their attack needs to do an exhaustive search for about $2t + 8$ bits (under reasonable assumptions on involved partial convergents), where $t = \log_2(d/n^{0.25})$.

Boneh & Durfee (1999), **Blömer & May (2001)**: Attacks based on **Coppersmith's** lattice-based technique for finding small roots of modular polynomials equations using LLL-algorithm. The attacks works (heuristically and asymptotically) if $d < n^{0.292}$ (and practically if $d < n^{0.278}$).

Advice: avoid $d < n^{0.5}$ in typical version of RSA.

D. (2004): A modification of the Verheul and van Tilborg attack, based on a result of **Worley (1981)** on Diophantine approximations, which implies that all rationals p/q satisfying the inequality

$$\left|\alpha - \frac{p}{q}\right| < \frac{c}{q^2},$$

for a positive real number c, are given by

$$\frac{p}{q} = \frac{rp_{m+1} \pm sp_m}{rq_{m+1} \pm sq_m}$$

for some $m \geq -1$ and nonnegative integers r and s such that $rs < 2c$.

D. & Ibrahimpašić (2008): Worley's result is sharp, in the sense that the condition $rs < 2c$ cannot be replaced by $rs < (2 - \varepsilon)c$ for any ε.

In both mentioned extensions of Wiener's attack, the candidates for the secret exponent are of the form $d = rq_{m+1} + sq_m$. We test all possibilities for d, and number of possibilities is roughly (number of possibilities for r) $\times$ (number of possibilities for s), which is $O(D^2)$, where $d = Dn^{0.25}$.

More precisely, number of possible pairs (r, s) in Verheul and van Tilborg attack is $O(D^2A^2)$, where $A = \max\{a_i : i = m+1, m+2, m+3\}$, while in our variant number of pairs is $O(D^2 \log A)$ (and also $O(D^2 \log D)$).

Testing:

There are two principal methods for testing:

1) compute p and q assuming d is correct guess:

$$\varphi(n) = (de - 1)/k, \quad p + q = n + 1 - \varphi(n),$$

$$(p - q)^2 = (p + q)^2 - 4n;$$

2) test the congruence $(M^e)^d \equiv M \pmod{n}$,
say for $M = 2$.

D. (2009): apply "meet-in-the-middle" attack to this second test.

We want to test whether
$2^{e(rq_{m+1}+sq_m)} \equiv 2 \pmod{n}$.
Note that m is (almost) fixed. Let m' be the largest odd integer such that

$$\frac{p_{m'}}{q_{m'}} > \frac{e}{n} + \frac{2.122e}{n\sqrt{n}}.$$

Then $m \in \{m', m'+1, m'+2\}$.

Let $2^{eq_{m+1}} \bmod n = a$, $(2^{eq_m})^{-1} \bmod n = b$. Then we test the congruence $a^r \equiv 2b^s \pmod{n}$.

We can do it by computing $a^r \bmod n$ for all r, sorting the list of results, and then computing $2b^s \bmod n$ for each s one at a time, and checking if the result appears in the sorted list.

This decrease the time complexity of testings phase to $O(D \log D)$ (with the space complexity $O(D)$).

We have implemented the proposed attack in PARI and C++ (with **Petričević**), and it works efficiently for values of D up to 2^{30}, i.e. for $d < 2^{30} n^{0.25}$.

For larger values of D the memory requirements become too demanded.

$\log_2 n$	$\log_2(2^{30} n^{0.25})$	$\log_2(n^{0.292})$	$\log_2(n^{0.278})$
768	222	224	214
1024	286	299	285
2048	542	598	569

A space-time tradeoff might be possible, by using unsymmetrical variants of Worley's result (with different bounds on r and s).

The attack can be improved:

- by using better approximations to $\frac{k}{d}$, e.g. $\frac{e}{n+1-2\sqrt{n}}$ instead of $\frac{e}{n}$.

- by using hash functions/tables instead of sorting.

With these improvements for 1024-bits RSA modulus n, the range in which our attack can be applied is comparable with known attacks based on LLL-algorithm.

Let $f(x) \in \mathbb{Z}[x]$ be a polynomial of degree δ and suppose that there is a "small" solution of the congruence $f(x) \equiv 0 \pmod{N}$, i.e. solution x_0 such that $|x_0| < N^{1/\delta}$. Can we find x_0 efficiently?
Coppersmith (1997): answer is yes.

Idea: construct new polynomial $h(x)$ which also satisfies $h(x_0) \equiv 0 \pmod{N}$, but this new polynomial should be small in the sense that the norm of its coefficients $\|h(x)\| := \left(\sum_{i=0}^{n} h_i^2\right)^{1/2}$ is small. If for a positive integer X holds $\|h(xX)\| < \frac{N}{\sqrt{n}}$, and $|x_0| < X$ satisfies the congruence $h(x_0) \equiv 0 \pmod{N}$, then $h(x_0) = 0$ over the integers and not just modulo N.

The polynomial $h(x)$ can be constructed using LLL-algorithm, which finds small vectors in lattices
(**A. K. Lenstra, H. W. Lenstra, L. Lovász (1982)**).

Boneh & Durfee (1999): extension of Wiener's attack to $d < n^{0.292}$. As with Wiener's attack, the starting points is $ed - k\varphi(n) = 1$, which can be written as

$$ed - k(n + 1 - p - q) = 1.$$

Put $s = p + q$, $a = n + 1$. Then finding small secret exponent d, say $d < n^{\delta}$, is equivalent to finding the two small solutions k and s to the following congruence

$$f(k, s) = k(s - a) \equiv 1 \pmod{e}.$$

Indeed,

$$|s| < 3\sqrt{n} \approx e^{0.5}, \quad |k| < \frac{de}{\varphi(n)} \approx e^{\delta}.$$

Situation is similar as in Coppersmith's result, but the application is not straightforward since here we need a two-variable analogue of that result.

Assuming that two polynomials corresponding to two shortest vectors outputted by LLL-algorithm are algebraically independent (which appears to be true in practice, but has not yet been proved rigorously), their common roots can be extracted using resultants. Under this assumption, the attack works theoretical up to asymptotical bound $\delta < 1 - 2^{-1/2} \approx 0.292$.

However, finding short vectors in lattices of dimension > 500 is hard computational problem, so it seems that at the moment the practical bound for the attack is $\delta < 0.278$, as reported recently by **Miller, Narayanan & Venkatesan (2017)**.

Attacks on generalizations of RSA

Koyama, Maurer, Okamoto & Vanstone (1991): KMOV (uses elliptic curves over $\mathbb{Z}/n\mathbb{Z}$)

Smith & Lennon (1993): LUC (uses Lucas sequences)

Pinch (1995): extended Wiener's attack

Ibrahimpašić (2009): extended Verheul & van Tilborg attack

Bunder, Nitaj, Susilo & Tonien (2017): attack on key equation of the form $ed - k(p^2 - 1)(q^2 - 1) = z$; the attack combines the continued fraction algorithm and Coppersmith's technique

CRT-RSA

Quisquater & Couvreur (1982):

Chinese Remainder Theorem (CRT) can be used to speed up the decryption in the RSA algorithm.

For a given ciphertext $c = m^e \bmod n$, the standard decryption algorithm for RSA recovers the plaintext by computing $m = c^d \bmod n$. Since the primes in the modulus are known to the decrypting party, the plaintext can also be recovered by first computing partial decryptions of the ciphertext modulo p and modulo q and then combining these with the Chinese remainder theorem to compute the plaintext.

We can first compute $m_p = c^d \bmod p$, $m_q = c^d \bmod q$, and then, since the primes are relatively prime to each other, combine these using the Chinese remainder theorem to obtain the plaintext. The plaintext is computed as

$$m = m_p + p((m_q - m_p)p^{-1} \bmod q).$$

Since the RSA primes are secret, this method for decryption cannot be extended to encryption. Encryption for CRT-RSA is the same as that for standard RSA.

When the private exponent is greater than $n^{1/2}$, a reduced exponent can be used in each of the partial decryptions. Let d_p and d_q be defined as $d_p = d \bmod (p-1)$, $d_q = d \bmod (q-1)$. It follows from Fermat's little theorem, that d_p and d_q can be used instead of d in the partial decryptions $m_p = c^{d_p} \bmod p$, $m_q = c^{d_q} \bmod q$. We call d_p and d_q the CRT-exponents.

There are several ways the exponents can be chosen for CRT-RSA, depending on whether the situation demands fast encryption or decryption. Decryption costs can be reduced by using a small private exponent but the private exponent should be at least $N^{0.292}$ (and probably larger) to be secure.

With CRT decryption, it is possible to use CRT-exponents much smaller than $n^{0.25}$ without being insecure. In the key generation algorithm, choosing different small CRT-exponents d_p and d_q first, and then computing the public exponent e, via the Chinese remainder theorem, will result in a public and secrete exponents that are, with high probability, full sized. Thus, all the small exponents attacks do not apply. The CRT-exponents cannot be chosen too small, as there attacks applicable in this situation.

CRT-RSA generated in this manner is called *Rebalanced RSA*. With rebalanced RSA, the CRT decryption costs are minimized at the expense of maximizing the encryption. Typical RSA with small public exponent and rebalanced RSA represent the extremes for minimizing (and maximizing) encryption or decryption costs.

Attacks on RSA with small public exponent

Suppose that we have three users with different public moduli n_1, n_2, n_3. In addition suppose they all have the same small public exponent $e = 3$. Assume now that someone sends them the same message m. Then their attacker sees the following ciphertexts:

$$c_1 \equiv m^3 \pmod{n_1},\ c_2 \equiv m^3 \pmod{n_2},\ c_3 \equiv m^3 \pmod{n_3}.$$

Now the attacker, using the Chinese Remainder Theorem, computes the solution of the system of linear congruences

$$x \equiv c_1 \pmod{n_1},\ x \equiv c_2 \pmod{n_2},\ x \equiv c_3 \pmod{n_3},$$

and obtains x such that $x \equiv m^3 \pmod{n_1 n_2 n_3}$. But since $m^3 < n_1 n_2 n_3$, we have in fact here the equality $x = m^3$, and the attacker can recover m by taking the real cube root of x.

We can protect against this attack by insisting that before encrypting a message m we first pad the message with some random user-specific data. Thus we will never send completely identical messages to different users.

However, one can still break this system using an attack due to Håstad, which uses the following result of Coppersmith.

Coppersmith (1997): Let $f \in \mathbb{Z}[x]$ be a monic polynomial of degree δ and n a positive integer. If there is some x_0 such that $f(x_0) \equiv 0 \pmod{n}$ and $|x_0| \leq X = n^{1/\delta - \varepsilon}$, then x_0 can be found in time a polynomial in $\log n$ and $1/\varepsilon$.

Håstad's attack (1985):

Suppose that, before encrypting, we first pad with some user-specific data at the beginning of the message:

$$c_i = (i \cdot 2^h + m)^e \bmod n_i, \quad i = 1, \ldots, k.$$

We have k polynomials $g_i(x) = (i \cdot 2^h + x)^e - c_i$, and we want to find m such that

$$g_i(m) \equiv 0 \pmod{n_i}.$$

Let $n = n_1 n_2 \cdots n_k$. Using Chinese Remainder Theorem we can find t_i such that

$$g(x) = \sum_{i=1}^{k} t_i g_i(x) \quad \text{and} \quad g(m) \equiv 0 \pmod{n}$$

($t_i \equiv 1 \pmod{n_i}$, $t_i \equiv 0 \pmod{n_j}$ for $j \neq i$).

Since g has degree e and is monic, using Coppersmith's theorem we can recover m in polynomial time, as long as we have at least as many ciphertexts as the encryption exponent, i.e. if $k \geq e$, since then $m < \min_i n_i < n^{1/k} \leq n^{1/e}$.

To avoid attacks on RSA with small public exponent, but still to use advantages of relatively small exponent with only few non-zero binary digits (to speed up modular exponentiation), the exponent

$$e = 65537 = 2^{16} + 1$$

can be recommended.

Thank you very much for your attention.