Lecture 4

Tel Aviv University, Fall 2004
Lattices in Computer Science

Attack on RSA with Low Public Exponent

Lecturer: Oded Regev
Scribe: Ishay Haviv

The well-known RSA public key cryptosystem is nowadays used in a wide variety of applications ranging from web browsers to smart cards. Since its initial publication in 1977, many researchers have tried to look for vulnerabilities in the system. Some clever attacks have been found. However, none of the known attacks is devastating and the RSA system is still considered secure.

In this lecture we present one such attack, originally due to Håstad and then greatly refined by Coppersmith. This attack can be mounted when RSA is used with a low public exponent. The attack is based on an algorithm for finding small solutions to low degree polynomials, which is in turn based on the LLL algorithm. This root finding algorithm is interesting on its own and is also used in other attacks on the RSA system.

Let us describe a simple version of the RSA cryptosystem. Let $N = p \cdot q$ be the product of two large primes of roughly the same size. Let r, s be two integers satisfying $r \cdot s = 1 \pmod{\varphi(N)}$, where $\varphi(N) = (p-1)(q-1)$ is the order of the multiplicative group $\mathbb{Z}_N^*$. We call N the RSA *modulus*, r the *public exponent*, and s the *private exponent*. The pair (N, r) is the *public key*. As its name suggests, it is public and is used to encrypt messages. The pair (N, s) is called the *secret key* or *private key*, and is known only to the recipient of encrypted messages. The secret key enables decryption of ciphertexts. A *message* is an integer $M \in \mathbb{Z}_N^*$. To encrypt M, one computes $C = M^r \pmod{N}$. To decrypt the ciphertext, the legitimate receiver computes $C^s \pmod{N}$. Indeed, $C^s = M^{r \cdot s} = M \pmod{N}$, where the last equality follows by Euler's theorem.

1 Low Public Exponent RSA

In many practical applications, the encryption process is performed by some limited device, such as a smart card. In such cases, raising M to a high power might be quite costly in terms of battery power, time, etc. In an attempt to simplify the encryption process, one might be tempted to modify the RSA cryptosystem by fixing the public exponent to be some small number, say $r = 3$. So now the encryption process simply involves raising a number to the power 3, which can be done using two multiplications.

At first it seems that this modification does not harm the security of the system. Indeed, it is not obvious how to invert the function $M^3 \mod N$ without knowing the factorization of N. However, as we shall see next, this modification allows for some clever attacks. Let us first recall the Chinese Remainder Theorem.

LEMMA 1 (CHINESE REMAINDER THEOREM)
Given r equalities $x \equiv a_i \pmod{m_i}$ such that $m_1, \ldots, m_r$ are pairwise relatively prime, there exists a unique $x \pmod{m_1 \cdot m_2 \cdots m_r}$ that satisfies these equalities, and this x can be found efficiently.

To demonstrate the danger in using a low public exponent, consider the following scenario. Alice wishes to send the same message to Bob, Charlie, and Dave. She encrypts her message with each of their public keys N_B, N_C, N_D. The ciphertexts are

$$c_B = m^3 \bmod N_B, \quad c_C = m^3 \bmod N_C, \text{ and } c_D = m^3 \bmod N_D.$$

We now show that an eavesdropper that receives c_B, c_C, and c_D can easily recover m. Without loss of generality, we can assume that N_B, N_C, and N_D are pairwise relatively prime (otherwise the eavesdropper finds a non-trivial factor of one of the public keys and can decrypt the message directly). By using the Chinese Remainder Theorem, the eavesdropper computes a number c such that $c = m^3 \bmod N_B N_C N_D$. Since $m < N_B, N_C, N_D$, we have $m^3 < N_B N_C N_D$ and hence $c = m^3$ where this is an equality over the integers. The message m can now be easily recovered by computing the cubic root of c over the integers.

It seems that one can avoid this problem by never sending the same message to more than one person. For instance, consider the following solution. Each person, as part of his public key, has some unique ID.

Instead of encrypting M, encrypt $M + 2^k \cdot ID$, where k is the length of the message M in bits and ID is the ID of the recipient. In this way we never send the same message to more than one person.

In the next section we prove that this modification suffers from a similar attack. In fact, we show a much more general attack: assume the public key is of the form (N, g) where g is some polynomial in M. To encrypt a message M, we send $g(M) \pmod N$ where g is the polynomial of the recipient. In the above special case, we had $g(M) = (M + 2^k ID)^3$.

Before presenting the attack, let us mention that low public exponent RSA is still considered secure when used carefully. Namely, the current wisdom says that one should use a moderate public exponent, say $r = 2^{16} + 1$ and pad the message with some random bits. Finally, we also mention that much stronger attacks are known on low private exponent s using different techniques.

2 The Attack

THEOREM 2
Let $N_1, \ldots, N_k$ be pairwise relatively prime integers, and let $N_{min} = \min_i N_i$. Let $g_i \in \mathbb{Z}_{N_i}[x]$ be k polynomials of maximum degree d. Suppose there exists a unique $M < N_{min}$ such that $g_i(M) = C_i \pmod{N_i}$ for all $i = 1, \ldots, k$. Then, if $k \geq d$, one can efficiently find M given $(N_i, g_i, C_i)_{i=1}^k$.

The main tool in the proof is the following theorem, whose proof is given in the next section.

THEOREM 3
Let N be an integer and $f \in \mathbb{Z}_N[x]$ a monic polynomial of degree d (i.e., the coefficient of x^d is 1). Then we can efficiently find all $x \in \mathbb{Z}$ such that $|x| \leq B$ and $f(x) = 0 \pmod N$ for $B = N^{1/d}$.

REMARK 1 This theorem is quite powerful, and is used in other attacks on the RSA system.

REMARK 2 Notice that the factorization of N is unknown. Otherwise, there are much better root finding algorithms.

PROOF: (of Theorem 2) Define $h_i = g_i - C_i$ for $1 \leq i \leq k$. Then we are looking for M such that $h_i(M) = 0 \pmod{N_i}$ for $i = 1, \ldots, k$. We can assume without loss of generality, that all h_i are monic (otherwise multiply them by the inverse mod N_i of their leading coefficients; if this coefficient is not invertible, then we have a factor of N_i and we're done). We can also assume without loss of generality that all h_i are of degree d (by multiplying by x^j for some j).

Next, we use the Chinese Remainder Theorem to combine the polynomials h_i into a polynomial h. Namely, we define

$$h(x) = \sum_{i=1}^{k} T_i h_i(x) \pmod{N_1 \cdot N_2 \cdots N_k}$$

where the T_i's are chosen using the Chinese Remainder Theorem to satisfy that for each i, $T_i \pmod{N_i} = 1$ and for each $i \neq j$, $T_i \pmod{N_j} = 0$. Then the polynomial $h(x)$ is

1. of degree d,
2. monic, since its x^d coefficient is 1 modulo any N_i, and
3. $h(M) = 0 \pmod{N_1 \cdot N_2 \cdots N_k}$.

We can now apply Theorem 3 to find M, since

$$M < N_{min} \leq (N_1 \cdot N_2 \cdots N_k)^{\frac{1}{k}} \leq (N_1 \cdot N_2 \cdots N_k)^{\frac{1}{d}}$$

and this completes the proof. □

3 Finding Roots of Low Degree Polynomials

In this section we prove Theorem 3. We do this in several steps. First, we prove the theorem for $B = \Theta(N^{2/(d(d+1))})$ where the Θ hides a constant that depends only on d. Then, we improve it to $B = \Theta(N^{\frac{1}{2d-1}})$, and mention how it can be improved to $B = N^{1/d}$. For simplicity, one can think of d as being a small constant, say, $d = 3$.

Our goal is to find solutions x to $f(x) = 0 \pmod N$ such that $|x| \leq B$. Write

$$f(x) = x^d + a_{d-1}x^{d-1} + \ldots + a_1 x + a_0.$$

Note that if it so happens that the coefficients of the polynomial f satisfy

$$\forall i = 0, \ldots, d.\ |a_i B^i| < \frac{N}{d+1} \tag{1}$$

then $|f(x)| \leq \sum_{i=0}^{d} |a_i B^i| < N$ for $|x| \leq B$. Hence, any small solution $|x| \leq B$ to the modular equation $f(x) = 0 \pmod N$ is also a solution to $f(x) = 0$ over the integers. Such solutions can be found efficiently using standard techniques (for example, we can find all real roots since there are at most d of them and then throw all the non-integer ones).

However, there is no reason for (1) to hold in general. The main idea is that even if (1) does not hold for f, it might hold for some integer multiple of f reduced modulo N. Intuitively, each multiple of f 'reshuffles' the coefficients and since there are roughly N possible multiples to consider, we might expect one of them to yield a polynomial all of whose coefficients are small.

More formally, the idea is to find a polynomial g that has property (1) and has the same roots as f. Then we can find all the roots of g directly and hence also all the small roots of f. In order to find such a g, consider the following set of polynomials:

$$\mathcal{Z}_1 = \left\{ N, Nx, Nx^2, \ldots, Nx^{d-1}, f(x) \right\}.$$

Notice that any integer combination of these polynomials has the same roots as f modulo N. Hence, it is enough to find an integer combination of these polynomials that satisfies property (1). This naturally leads us to consider the lattice

$$\mathcal{L}_1 = \begin{pmatrix} N & 0 & \cdots & 0 & a_0 \\ 0 & BN & \ddots & 0 & Ba_1 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & B^{d-1}N & B^{d-1}a_{d-1} \\ 0 & \cdots & \cdots & 0 & B^d \end{pmatrix}_{(d+1)\times(d+1)}.$$

Each column corresponds to one polynomial in $\mathcal{Z}_1$. The ith row, $i = 0, \ldots, d$, corresponds to the coefficient of x^i multiplied by B^i.

We would like to show that we can find a short vector in this lattice. So first, let us calculate $\det(\mathcal{L}_1)$,

$$\det(\mathcal{L}_1) = N \cdot BN \cdot \ldots \cdot B^{d-1}N \cdot B^d = N^d \cdot B^{1+2+\ldots+d} = N^d \cdot B^{d(d+1)/2}.$$

Using LLL, we find an integer combination of the columns v (i.e., a lattice vector) whose length satisfies

$$\|v\| \leq O(\lambda_1(\mathcal{L}_1)) \leq O\left((\det(\mathcal{L}_1))^{\frac{1}{d+1}}\right) = O\left(N \cdot \frac{B^{d/2}}{N^{\frac{1}{d+1}}}\right)$$

where the $O()$ hides a constant that depends only on d and the second inequality follows from Minkowski's theorem. This is less than $\frac{N}{d+1}$ if we take $B \leq c_1(d) N^{\frac{2}{d(d+1)}}$ for a small enough $c_1(d)$ that depends only on d. In particular, each of v's coordinates is less than $\frac{N}{d+1}$. Hence, if we take the corresponding combination of the polynomials in $\mathcal{Z}_1$, we obtain a polynomial

$$g(x) = b_d x^d + b_{d-1} x^{d-1} + \ldots + b_1 x + b_0$$

such that $\forall i = 0, \ldots, d,\ |b_i B^i| < \frac{N}{d+1}$, and g has the same roots as f. Now we can find the small roots of g over the integers using standard methods. This completes the description of the basic idea.

Next, we would like to improve the bound B. The main idea is to notice that we can consider a larger set of polynomials. Consider the set of polynomials

$$\mathcal{Z}_2 = \left\{ N, Nx, \ldots, Nx^{d-1} \right\} \bigcup \left\{ f(x), xf(x), \ldots, x^{d-1} f(x) \right\}.$$

For any integer combination g of these polynomials, the roots of g contain all the roots of f. [1] Consider the lattice

$$\mathcal{L}_2 = \begin{pmatrix} N & 0 & 0 & 0 & a_0 & 0 & \ldots & 0 \\ 0 & BN & \ddots & \vdots & \vdots & Ba_0 & \ddots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots & \ddots & 0 \\ \vdots & \vdots & \ddots & B^{d-1}N & B^{d-1}a_{d-1} & \vdots & \vdots & B^{d-1}a_0 \\ \vdots & \vdots & \ddots & 0 & B^d & B^d a_{d-1} & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & B^{d+1} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & B^{2d-2}a_{d-1} \\ 0 & 0 & \ldots & \ldots & 0 & 0 & 0 & B^{2d-1} \end{pmatrix}_{(2d)\times(2d)}$$

As before, using LLL we obtain a vector v whose length satisfies

$$\|v\| \leq O\left((\det(\mathcal{L}_2))^{\frac{1}{2d}} \right) = O\left((N^d \cdot B^{d(2d-1)})^{\frac{1}{2d}} \right) = O\left(N \cdot \frac{B^{d-1/2}}{\sqrt{N}} \right).$$

Hence, choosing $B \leq c_2(d) N^{\frac{1}{2d-1}}$ for some $c_2(d)$ makes this smaller than $\frac{N}{2d}$. Let g be the integer combination of the polynomials in $\mathcal{Z}_2$ that corresponds to v. This is a degree $2d-1$ polynomial whose coefficients satisfy that for all $i = 0, \ldots, 2d-1$, $|b_i B^i| < \frac{N}{2d}$. We can now find the small roots of g by considering it as a polynomial over the integers.

We now sketch how further improvements to the bound B can be achieved. For some $h \geq 2$, consider the set of polynomials

$$\mathcal{Z}_3 = \{ N^{h-i-1} f(x)^i x^j \mid 0 \leq j < d,\ 0 \leq i < h \}.$$

For $h = 2$ this is exactly $\mathcal{Z}_2$. Notice that for any integer combination g of these polynomials and any x such that $f(x) = 0 \pmod{N}$ it holds that $g(x) = 0 \pmod{N^{h-1}}$. Following a similar argument, one can obtain $B = N^{1/d-\epsilon}$ for any constant $\epsilon > 0$ by setting h to be a large enough constant. Taking h to be a slowly growing function of N yields a bound of the form $N^{1/d}/C$ for some constant C. Using this, it is easy to find all roots in a slightly larger range, say $B = 10N^{1/d}$, by partitioning this range into $10C$ smaller ranges and using the previous algorithm on each of them.

[1] Notice that g might have more roots than f (for example, 0 is always a root of $xf(x)$). This does not raise any difficulty since we anyway find all roots of g.

REMARK 3 Our proof also gives an answer to the following purely mathematical question: how many roots modulo N can a degree d polynomial have in the range $x \in \{-B, \ldots, 0, \ldots, B\}$? The first proof gives an upper bound of d for $B = c_1(d)N^{\frac{2}{d(d+1)}}$ and the second gives an upper bound of $2d-1$ for $B = c_2(d)N^{\frac{1}{2d-1}}$.

REMARK 4 It seems that this technique cannot be improved much beyond $B = N^{1/d}$. Consider, for example, $N = p^2$ and $f(x) = x^2$. Then in the range $|x| < N^{\frac{1}{2}+\varepsilon}$ this polynomial has roughly N^{ε} roots so we cannot hope to output all of them.

References

[1] Dan Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society (AMS)*, 46(2):203–213, 1999.

[2] Don Coppersmith. Finding small solutions to small degree polynomials. *Lecture Notes in Computer Science*, 2146:20–31, 2001.