Cryptography Fundamentals **Fall 2017**

Lecture 3: Modular e-th Root in RSA

Lecturer: Wenfei Wu *Scribes: Kefan Dong, Ce Jin*

3.1 Introduction

The main problem we study is how to compute e-th root in $\mathbb{Z}_N$. Given e, c and N, our task is to find x such that

$$x^e \equiv c \pmod N.$$

This problem has great importance in cryptography since efficiently solving e-th root leads to attacks on RSA, one of the most widely used public-key cryptosystems.

In this report, we will study some special cases of this problem which can be more efficiently solved, and discuss the hardness of the general problem and its relation to Integer Factorization problem.

3.2 When N is prime

First we consider the case of $N = p$, where p is an odd prime number. We will show that in this case e-th root can be solved for small e.

3.2.1 Case $e = 2$

The most basic case is when $e = 2$. We need to find x satisfying

$$x^2 \equiv c \pmod p.$$

If such x exists, we say c is a *quadratic residue* of p; otherwise, c is a *quadratic nonresidue* of p. Both residues and nonresidues exist. In fact, among $\{1, 2, \dots, p-1\}$, exactly half of them are quadratic residues, namely $1^2, 2^2, \dots, \left(\frac{p-1}{2}\right)^2$. This fact can be easily shown using $x^2 \equiv (-x)^2$ and $x^2 - y^2 \equiv (x+y)(x-y)$.

Before solving x, we first need to tell whether c is a quadratic residue or not. This can be done using Euler's criterion.

Theorem 3.1 (Euler's criterion) *Let p be an odd prime and c an integer coprime to p. Then*

$$c^{(p-1)/2} \equiv \begin{cases} 1 \pmod p & \textit{if } c \textit{ is a quadratic residue} \\ -1 \pmod p & \textit{if } c \textit{ is a quadratic nonresidue} \end{cases}$$

Proof: By Fermat's theorem, $c^{p-1} \equiv 1$. Then $c^{(p-1)/2} = \pm 1$. If $c \equiv a^2$ is a quadratic residue, then $c^{(p-1)/2} \equiv a^{(p-1)} \equiv 1$. The equation $f(x) = x^{(p-1)/2} \equiv 1$ has at most $\deg(f(x)) = (p-1)/2$ roots, which are all the (non-zero) quadratic residues of p. So for the remaining nonresidue elements c, there must be $c^{(p-1)/2} \equiv -1$. ■

We can compute $c^{(p-1)/2} \bmod p$ using repeated squaring algorithm in $O(\text{poly}(\log p))$ time.

Now we proceed to find the modular square roots of c given that c is a quadratic residue of p. We use Cipolla's algorithm.

Cipolla's algorithm:

Step 1: Find an a such that $a^2 - c$ is a quadratic nonresidue. Define $\omega = \sqrt{a^2 - c}$.

Step 2: Return $\pm(a - \omega)^{(p+1)/2}$.

In step 1 we constructed an extension field $\mathrm{F}_p(\omega) \cong \mathrm{F}_{p^2}$. Step 2 is done in this field. Elements of this field are expressed as $a + b\omega$, where $a, b \in \{0, 1, \ldots, p-1\}$. Arithmetic operations are (denote $u = \omega^2 = a^2 - c$)

$$\begin{aligned}(a+b\omega)+(c+d\omega) &= (a+c)+(b+d)\omega,\\ (a+b\omega)-(c+d\omega) &= (a-c)+(b-d)\omega,\\ (a+b\omega)(c+d\omega) &= (ac+ubd)+(ad+bc)\omega,\\ (a+b\omega)/(c+d\omega) &= \frac{(ac-ubd)+(bc-ad)\omega}{c^2-ud^2}.\end{aligned}$$

First we show the correctness of step 2, i.e., the square of the returned number equals c.

Proof:

$$\begin{aligned}&(a-\omega)^{p+1}\\ =&(a-\omega)^p(a-\omega)\\ =&(a^p-\omega^p)(a-\omega)\\ =&(a-(a^2-c)^{(p-1)/2}\omega)(a-\omega)\\ =&(a+\omega)(a-\omega)\\ =&c.\end{aligned}$$

■

Next we show that step 1 can be implemented efficiently by random guessing.

Theorem 3.2 *Suppose c is a (nonzero) quadratic residue of p. Let a be uniformly randomly drawn from $\{0, 1, \ldots, p-1\}$. Then $\Pr[a^2 - c$ is a quadratic nonresidue of $p] = (p-1)/2p$.*

Proof: Polynomial $x^2 + 2ax + c$ can be written as $(x-\alpha)(x-\beta) \in \mathrm{F}_p[x]$ iff $\sqrt{(2a)^2 - 4c}$ exists in F_p, i.e., $a^2 - c$ is a quadratic residue. So we count the number of irreducible polynomials $x^2 + 2ax + c$ where a runs through F_p. The reducible polynomials have the form $x^2 + 2ax + c = (x-\alpha)(x - c/\alpha)$. So the number of reducible polynomials equals the number of unordered pairs $(\alpha, c/\alpha)$, which is $(p-3)/2 + 2 = (p+1)/2$. Hence the probabilty that $x^2 + 2ax + c$ is irreducible is $1 - \frac{(p+1)/2}{p} = (p-1)/2p$. ■

Therefore the expected number of random tries in step 1 is $O(1)$. The total complexity of Cipolla's algorithm is $O(\text{poly}(\log p))$.

3.2.2 Larger e

Now we consider larger e. Again, we assume c is coprime to p.

The easiest case is when $\gcd(e, p-1) = 1$. In this case, we can easily find d such that $de \equiv 1 \pmod{p-1}$. Then by Fermat's theorem, $(c^d)^e = c^{de} \equiv c \pmod p$, so c^d is an e-th root of c. Moreover, for any x such that $c \equiv x^e \pmod p$, we have $c^d \equiv (x^e)^d = x^{ed} \equiv x \pmod p$. Hence, c^d is the unique e-th root of c. The total time complexity is $O(\mathrm{poly}(p))$.

It becomes harder when $\gcd(e, p-1) > 1$, but it is still solvable when e is not too large. If e is composite, we can factorize it into $e = mn (m, n \geq 2)$. Suppose x is an e-th root of c. Then e^m must be an n-th root of c. So we can compute all n-th roots of c, and for each of them we compute its m-th roots, and then all e-th roots of c are found. Therefore, we only focus on how to find e-th root of c when e is a small prime. Then, $\gcd(e, p-1) > 1$ implies $e \mid p-1$.

First we need to generalize some results in the previous section. We say c is an e-th power residue of p if exists x such that $x^e \equiv c \pmod p$; otherwise c is an e-th power nonresidue.

Theorem 3.3 *Let p be a prime number, and e a prime factor of $p-1$. Then there are $(p-1)/e$ nonzero e-th power residues of p.*

Proof: Let g be a primitive root of p (i.e., a generator of the cyclic multiplicative group F_p^*). Then g^i is an e-th power residue iff $(p-1) \mid ie$. There are $(p-1)/e$ such i among $\{0, 1, \ldots, p-2\}$. ■

Theorem 3.4 *Let p be a prime number, and e a prime factor of $p-1$. Let c be an integer coprime to p. Then c is an e-th power residue of p iff $c^{(p-1)/e} \equiv 1 \pmod p$.*

Proof: Similar to the case of $e = 2$. ■

Now we use a generalization of Cipolla's algorithm.

Step 1: Find an a such that $a^e - c$ is an e-th power nonresidue. Let $\omega = \sqrt[e]{a^e - c}$.

Step 2: Return $(a - \omega)^{(p^{e-1} + p^{e-2} + \cdots + 1)/e}$ as an e-th root of c.

We first prove that $x^e - (a^e - c)$ is irreducible over F_p

Theorem 3.5 *Let K be a field, p a prime number. Element $a \in K$ is not a p-th power in K. Then $x^p - a$ is irreducible over K.*

Proof: Let L be the splitting field for $x^p - a$ over K, and $b \in L$ such that $b^p = a$ and $b \notin K$. Then we must have $x^p - a = (x-b)(x-br)(x-br^2)\ldots(x-br^{p-1})$, where $r \in L$ is a p-th unit root, $r^p = 1$. Suppose $x^p - a = f(x)g(x)$, $1 \leq d = \deg(f) < p$ and $f, g \in K[x]$. Then $f(x)$ can be expressed as the product of some $(x - br^i)$ factors. The constant term of $f(x)$ has the form $(-b)^d r^s \in K$. Pick integer m such that $dm = kp + 1$, then $((-b)^d r^s)^m = (-1)^{dm} a^k b r^{sm} \in K$, which implies $br^{sm} \in K$. Hence a can be expressed as a p-th power in K, $a = (br^{sm})^p$, a contradiction. ■

Hence, $x^e - (a^e - c)$ is irreducible over F_p. Arithmetic operations can be carried out in the field $\mathrm{F}_p(\omega) = \mathrm{F}_p[x]/(x^e - (a^e - c))$.

Now we prove the correctness of step 2.

Proof:

$$\begin{aligned}&(a-\omega)^{p^{e-1}+p^{e-2}+\cdots+1}\\=&(a-\omega)^{p^{e-1}}(a-\omega)^{p^{e-2}}\dots(a-\omega)^1\\=&\left(a-\omega^{p^{e-1}}\right)\left(a-\omega^{p^{e-2}}\right)\dots(a-\omega^1),\end{aligned}$$

For any $0 \le i \le e-1$, $(\omega^{p^i})^e = (\omega^e)^{p^i} = (a^e-c)^{p^i} = a^e - c$, so ω^{p^i} is a root of polynomial $x^e-(a^e-c)$.

Moreover, we can show that these roots are distinct. Suppose there exist $0 \le i < j \le e-1$ such that $\omega^{p^i} = \omega^{p^j}$. Then $\omega^{p^j-p^i} = 1$. Since $\gcd(p^j-p^i, p^e-1) = \gcd(p^{j-i}-1, p^e-1) = 1$, [1] we can find t such that $(p^j-p^i)t \equiv 1 \pmod{p^e-1}$. Then $1 = 1^t = \omega^{t(p^j-p^i)} = \omega$, contradicting $\omega \notin \mathrm{F}_p$.

Hence they are all e roots of the polynomial $x^e-(a^e-c)$. We factor it into $x^e-(a^e-c) = (x-\omega^{p^{e-1}})\dots(x-\omega^1)$. Then

$$\left(a-\omega^{p^{e-1}}\right)\left(a-\omega^{p^{e-2}}\right)\dots(a-\omega^1) = a^e-(a^e-c) = c.$$

■

By experiment, the failing probability in step 1 is very low. However it's still not clear how to bound this probability.

The time complexity of this algorithm is $O(\mathrm{poly}(e, \log p))$.

3.3 When N is prime power

When $N = p^k$, where p is a prime, we can use power lifting technique. We use $e = 2$ to demonstrate this technique.

We are going to solve $x^2 \equiv c \pmod N$. When $c \equiv 0 \pmod{p^k}$, the solution is $x \equiv 0 \pmod{p^{\lceil k/2 \rceil}}$

Otherwise, let $c = p^r a, \gcd(p,a) = 1, 0 \le r < k$. When x exists, r must be even. Let $x = p^{r/2}x'$, then $x'^2 \equiv a \pmod{p^{k-r}}$

So we only need to consider the case when $\gcd(p,c) = 1$.

3.3.1 When p is odd

Theorem 3.6 *Solution exists iff c is a quadratic residule of p. When solution exists, there are exactly two solutions, $\pm x_k$.*

Proof: We use induction. Suppose $x_k^2 \equiv c \pmod{p^k}$. Let

$$\begin{aligned}x_{k+1} =& x_k + tp^k \pmod{p^{k+1}}, t = 0, 1, \cdots, p-1,\\x_{k+1}^2 =& x_k^2 + t^2p^{2k} + 2x_ktp^k\\\equiv& x_k^2 + 2x_ktp^k \pmod{p^{k+1}}\\\equiv& c \pmod{p^{k+1}}\end{aligned}$$

[1]When $m < n$ are coprime, $\gcd(a^m-1, a^n-1) = \gcd(a^m-1, (a^n-1)-(a^m-1)) = \gcd(a^m-1, a^m(a^{n-m}-1) = \gcd(a^m-1, a^{n-m}-1) = \cdots = 1$

Then

$$2x_k t \equiv (c - x_k^2)/p^k \pmod{p}.$$

Solve t and we get the value of x_{k+1}. ■

3.3.2 When $p = 2$

Case $N = 4$ is trivial. Now we assume $N \geq 8$.

Every odd x satisfies $x^2 \equiv 1 \pmod 8$, so we must have $c \equiv 1 \pmod 8$.

Theorem 3.7 *For $N \geq 8$, solution exists iff $c \equiv 1 \pmod 8$. In this case there are exactly four solutions, $\pm x_k, \pm(N/2 - x_k)$.*

Proof: The statement holds for $N = 8$.

We use induction. The four solutions for $x^2 \equiv c \pmod{2^k}$ are $\pm x_k, \pm(2^{k-1} - x_k)$. Then

$$\begin{aligned} & x_k^2 - (2^{k-1} - x_k)^2 \\ =& (2x_k - 2^{k-1})2^{k-1} \\ \equiv& x_k 2^k \pmod{2^{k+1}} \\ \equiv& 2^k \pmod{2^{k+1}} \end{aligned}$$

Hence, exactly one of $x_k, 2^{k-1} - x_k$ can be used as x_{k+1}, such that $x_{k+1}^2 \equiv c \pmod{2^{k+1}}$.

Then we can easily check that $(2^k - x_{k+1})^2 \equiv x_{k+1}^2 (\mathrm{mod} 2^{k+1})$. So we have found the four new solutions $\pm x_k, \pm(2^k - x_k)$. ■

3.4 Relation with Integer Factorization

When N is composite, we can factorize N and compute the e-th root modulo prime powers, and combine them using CRT. So computing e-th root is easy given that we can factorize big integers. The converse is also true: an efficient algorithm for finding modulo square root can be used to factorize N.

We randomly generate $x \in \{0, 1, \ldots, N-1\}$. Let $c \equiv x^2 \bmod N$. We use the algorithm to find a square root y of c. Then $x^2 \equiv y^2$. So $(x+y)(x-y) \equiv 0 \pmod N$. When $x \neq \pm y$, we can compute $\gcd(x+y, N), \gcd(x-y, N)$ to find a proper factor of N.

We roughly explain why the success probability high. Suppose $N = PQ$, where $P > 1, Q > 1, \gcd(P, Q) = 1$. Denote $C(a, b) = n$ such that $n \equiv a \pmod P, n \equiv b \pmod Q$ (i.e., combining using CRT). Then $C(a, b), C(a, -b), C(-a, b), C(-a, -b)$ are distinct, but their squares are the same. Suppose the x you pick is $C(a, b)$. If the algorithm returns $C(a, -b)$ or $C(-a, b)$ then you will succeed.

3.5 Attack on RSA

According to Dan [B99], it is still open that whether breaking RSA as hard as factoring. Formally, the concrete open problem is stated below.

Open Problem 3.8 *Given integers N and e satisfying* $\gcd(e, N) = 1$ *, define the function $f_{e,N} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ by $f_{e,N} = x^{1/e} \mod N$. Is there a polynomial-time algorithm A that computes the factorization of N given N and access to an "oracle" $f_{e,N}(x)$ for some e?*

Therefore in this section, we only consider attacking RSA as recovering the private key. In the original paper of RSA [RSA78], the key generation phase is defined as follows.

Definition 3.9 *A RSA key generation phase runs as follows,*

- *Choose two large prime numbers p, q. Compute $N = pq$.*
- *Choose the public key d, where $(d, \phi(N)) = 1$. Private key e is computed by $ed \equiv 1 \mod \phi(N)$.*

However, due to computation aspects, a alternative approach is proposed by using Chinese Reminder Theorem to speed up computation. The key generation phase of this CRT-RSA is defined as follows.

Definition 3.10 *A CRT-RSA key generation phase runs as follows,*

- *Choose two large prime numbers p, q. Compute $N = pq$.*
- *Choose the public key $d = d_p d_q$, where $(d_p, p-1) = 1$ and $(d_q, q-1) = 1$. Private key e is computed by $e = e_p e_q$ where $e_p d_p \equiv 1 \mod p-1$ and $e_q d_q \equiv 1 \mod q-1$.*

Note that in CRT-RSA, we have $ed \equiv 1 \mod \mathrm{LCM}(p-1, q-1)$, where $\mathrm{LCM}(x, y) = xy/\gcd(x, y)$ is the least common multiple of x and y.

In this section, we will show that,

- private key recovering in both original RSA and CRT-RSA is as hard as factoring,
- RSA is not secure when $d \le N^{0.25}$,
- it is not secure to choose $\min\{d_p, d_q\}$ very small (which is a good idea in respect of computation),
- both RSA and CRT-RSA are not secure when $d \le N^{0.292}$.

3.5.1 Hardness of private key recovering

First we define private key recovering problem formally.

Definition 3.11 *Private key recovering problem. Given $N = pq$ and e, where p, q are primes and $(e, \phi(N)) = 1$, it is required to find $1 \le d < \phi(N)$ such that $ed \equiv 1 \mod \phi(N)$.*

Note that if such d is found, it also satisfies $ed \equiv 1 \mod \mathrm{LCM}(p-1, q-1)$ because $\mathrm{LCM}(p-1, q-1) \mid \phi(N)$.

Next we prove that private key recovering is as hard as factoring.

On the one hand, if one can factor $N = pq$, $\phi(N) = (p-1)(q-1)$ can be easily computed. Therefore d can be found by extended Euclidean algorithm.

On the other hand, if one can recover d, p and q can be found by using the probabilistic algorithm stated in question 10 of midterm exam. A stronger conclusion is shown by Miller [G75], assuming extended Riemann's Hypothesis is true.

Theorem 3.12 *Define $\lambda'(n)$ to be $\lambda'(n) = \mathrm{LCM}(p_1 - 1, \cdots, p_m - 1)$, where $p_1, p_2, \cdots, p_m$ are all distinct prime factors of n. Let g be any function that*

- $\lambda'(n) \mid g(n)$,
- $|g(n)| = \mathcal{O}(|n|^k)$,

where $|n| = \lceil \log n \rceil$ is the number of bits of n. Then given g, factoring n can done in polynomial time.

Proof: Let $\#_2(n) = \max\{t | t \geq 0, 2^t \mid n\}$ be the number of 2s in n. Consider the following algorithm,

- Check whether n is a perfect power, i.e., $n = p^\alpha$. If so, factorization is done. Terminate.
- Let $f(n)$ be a function of n where $1 \leq f(n) < n$. For $2 \leq a \leq f(n)$, and $1 \leq k \leq \#_2(g(n))$,
  - if $a \mid n$, factorization is done. Terminate.
  - calculate $g = \gcd(n, a^{g(n)/2^k} - 1)$. If g is a nontrivial factor of n, i.e., $g \neq 1$ and $g \neq n$, the factorization is done. Terminate.

We want to show that, for some $f(n)$ bounded by $|n|^c$, where c is a constant, the algorithm will terminate.

Recall that $\lambda'(n) = \mathrm{LCM}(p_1 - 1, \cdots, p_m - 1)$, we have $\#_2(\lambda'(n)) = \max\{\#_2(p_1 - 1) \mid 1 \leq i \leq m\}$. Then consider two cases.

a. when $\#_2(\lambda'(n)) > \#_2(p_x - 1)$ for some x. Because $\lambda'(n) = \mathrm{LCM}(p_1 - 1, \cdots, p_m - 1)$, there exists y such that $\#_2(\lambda'(n)) = \#_2(p_y - 1)$. Let $k = \#_2(g(n)/\lambda'(n)) + 1$ and a be a quadratic non-residue mod p_y. Then we have,

$$a^{g(n)/2^k} \equiv 1 \mod p_x, a^{g(n)/2^k} \equiv -1 \mod p_y.$$

The first equation is trivial because of Fermat's theorem. For the second equation, notice that under $\mathbb{Z}^*_{p_y}$, we can define index of a as $ind(a) = \min\{x | g^x \equiv a \mod p\}$ where g is an generator of $\mathbb{Z}^*_{p_y}$ (i.e., primitive root mod p_y). Then we have the following lemma,

Lemma 3.13 *For a prime p and integer q, where $q \mid p - 1$, for any $a \in \mathbb{Z}^*_p$, a is a q-th residue mod p if and only if $q \mid ind(a)$.*

The proof of this lemma is relatively easy and can be found in appendix A.

Now if, on contradiction, $a^{g(n)/2^k} \equiv 1 \mod p_y$. This gives us

$$q - 1 \mid ind(a) \cdot \frac{g(n)}{2^k},$$

which is not true because $2 ind(a)$. And because $a^{g(n)/2^{k-1}} \equiv 1 \mod p_y$, we have $a^{g(n)/2k} \equiv -1 \mod p_y$.

Due to Chinese Reminder Theorem, we can see that $a^{g(n)/2^k}$ has a non-trivial greatest common divisor with n, therefore the algorithm will terminate for quadratic non-residue a in this case.

b. when $\#_2(\lambda'(n)) = \#_2(p_1 - 1) = \cdots = \#_2(p_m - 1)$. For arbitrary $1 \leq x, y \leq m$, where $x \neq y$, consider p_x, p_y. For $a \in \mathbb{Z}^*_n$ where

$$\left(\frac{a}{p_x p_y}\right) = -1,$$

we have $(a/p_x)(a/p_y) = -1$, where (a/p) is Legendre symbol[2] . W.L.O.G., we assume $(a/p_x) = 1$ and $(a/p_y) = -1$. In this case, a is a quadratic residue mod p_x and is not a quadratic residue mod p_y. Similar arguments can be applied to conclude that $a^{g(n)/2^k}$ has a non-trivial greatest common divisor with n.

Assuming extended Riemann's Hypothesis is true, it is shown[A52] that $f(n)$ is bounded by $\mathcal{O}\left(|n|^2\right)$, i.e., we only need to check $\mathcal{O}\left(|n|^2\right)$ times before the factorization is done, which means that calculate such $g(n)$ (in our case, a multiple of $\phi(n)$ obtained by $ed - 1$) is easer than factorization. ∎

3.5.2 Wiener's attack when $d \le N^{0.25}$

3.5.2.1 High level idea

In 1990, Wiener shown that for small private key d where $d \le N^{0.25}$, there exists a polynomial time algorithm that can recover d [W89]. The one main tools Wiener uses is as follows.

Theorem 3.14 *For any $f \in \mathbb{Q}$. Given estimation f', where $f' = f(1-\delta)$ and a polynomial oracle $O(t)$, which returns whether $f = t$, one can guess f correctly for some small δ in polynomial time.*

The crucial identity that uses this tool is,

$$ed = K \cdot \text{LCM}(p-1, q-1) + 1 = \frac{K}{G}(p-1)(q-1) + 1 = \frac{k}{g}(p-1)(q-1) + 1, \tag{3.1}$$

where $G = \gcd(p-1, q-1)$, and k/g is the irreducible fraction of K/G, i.e., $\gcd(k, g) = 1$ and $k/g = K/G$. Reformulate this identity we have,

$$\frac{e}{pq} = \frac{k}{dg}\left(1 - \frac{p+q-1-g/k}{pq}\right). \tag{3.2}$$

Let $f = \frac{k}{dg}$ is the faction that we are guessing, $f' = \frac{e}{pq}$ is the estimation that we know, $\delta = \frac{p+q-1-g/k}{pq}$. The bound given by Wiener is,

$$kdg < \frac{pq}{\frac{3}{2}(p+q)}.$$

For original RSA, we can regard g as 1 and this gives us

$$kd \le \frac{pg}{3/2(p+q)}.$$

According to equation 3.2, we have $d \le k$. Therefore this attack will success when

$$d^2 \le \frac{pq}{3/2(p+q)},$$

which means $d = \mathcal{O}\left(n^{0.25}\right)$.

3.5.2.2 Prove of Theorem 3.14

In Wiener's paper, continued fractions are used to guess f. A continued fraction $\langle q_0; q_1, q_2, \cdots, q_m\rangle$ is defined as,

$$\langle q_0; q_1, q_2, \cdots, q_m\rangle = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{1}{\cdots}}}.$$

[2]`https://en.wikipedia.org/wiki/Legendre_symbol`

Note that given f, its continued faction representation can be computed in the following way:

- $q_0 = \lfloor f \rfloor, r_0 = f - q_0$,
- $q_i = \lfloor \frac{1}{r_{i-1}} \rfloor, r_i = \frac{1}{r_{i-1}} - q_i, \quad \text{for } i = 1, 2, \cdots, m.$

For a rational number $f \in \mathbb{Q}$, there are exactly two representations for f, which is $\langle q_0, q_1, \cdots, q_m \rangle$ generated by the above algorithm, and $\langle q_0, q_1, \cdots, q_m - 1, 1 \rangle$. Here we only consider the representation where $q_m > 1$, i.e., the representation generated by our algorithm.

The basic idea of Wiener's attack is that, when f' and f are close, their continued fraction representations are close. Therefore we can guess f by writing down the continued fraction representation of f', and twist it a little bit. The algorithm runs as follows,

- Repeat generating the next quotient q_i for f', i.e., the current representation is $\langle q_0, q_1, \cdots, q_i \rangle$
- Construct $g = \begin{cases} \langle q_0, q_1, \cdots, q_i \rangle, \text{if } i \text{ is odd}, \\ \langle q_0, q_1, \cdots, q_i + 1 \rangle, \text{if } i \text{ is even}. \end{cases}$.
- Use the oracle O to check whether g is correct. If not, continue. Otherwise terminate.

Note that the guess g will always no less than f'.

To analyze when the algorithm can success, the following fact is needed.

Fact 3.15 *For any $f, f' \in \mathbb{R}$, with their continued fraction representation $\langle q_0, q_1, \cdots, q_m, \cdots \rangle$ and $\langle q'_0, q'_1, \cdots, q'_m, \cdots \rangle$. Suppose $q_i = q'_i$ for $0 \le i \le m$ and $q_m \ne q'_m$. W.L.O.G., suppose $q_m < q'_m$, then we have*

- *$f < f'$, when m is even.*
- *$f > f'$, when m is odd.*

Therefore we can easily see that Wiener's algorithm will success when,

$$\begin{cases} \langle q_0, q_1, \cdots, q_{m-1}, q_m - 1 \rangle < f' < \langle q_0, \cdots, q_m \rangle, & \text{when } m \text{ is even}, \\ \langle q_0, q_1, \cdots, q_{m-1}, q_m + 1 \rangle < f' < \langle q_0, \cdots, q_m \rangle, & \text{when } m \text{ is odd}. \end{cases}$$

Let $f = n/d$ where $\gcd(n, d) = 1$. Based on the above condition we have, Wiener's algorithm will success when

$$\delta < \frac{1}{\frac{3}{2}nd}.$$

The concrete calculation is tedious and therefore omitted. It can be found in [W89].

Applying this to equation 3.2, we can get the bound $\mathcal{O}\left(N^{0.25}\right)$.

3.5.3 Attack on small private key in CRT-RSA

In respect of computation, it seems to be a good idea to choose d_p and d_q small, because it will speed up the decryption. In practice, a small public key e is chosen. Usually $e = 65537$. One may think it is helpful to choose d_p and d_q also small. However, we show that this is not secure by showing a attack which runs in $\widetilde{\mathcal{O}}\left(\sqrt{\min\{d_p, d_q\}}\right)$.

Suppose we are given N and e, where $N = pq$ and $ed \equiv 1 \mod p-1$, and d is bounded by $1 \leq d \leq D^2$. Note that p, q, d are not given. Now we want to factor N in $\widetilde{\mathcal{O}}(D)$ time.

For a random x, we can see that with high probability (namely $1 - 1/q$),

$$x^{ed} \equiv x \mod p, \quad x^{ed} \not\equiv x \mod q.$$

Therefore with high probability $\gcd(x^{ed} - x, N) = p$. This fact leads to a attack which runs in $\widetilde{\mathcal{O}}(D^2)$ time. Now we want to speed up this straightforward implementation by a twisted baby-step giant-step method.

Let $d = a + bD$ where $0 \leq a, b < D$. Consider $t = \prod_{i=0}^{D-1}(x^{ei}x^{ebD} - x)$. We have,

$$\gcd(t, N) = \mathrm{LCM}_{i=0}^{D-1}\left\{\gcd(x^{ei}x^{ebD} - x, N)\right\}.$$

According to union bound, the probability that $\gcd(t, N) \neq p$ is at most $\frac{D}{p-1}$, which is negligible when D is small.

Consider the polynomial $f(y) = \prod_{i=0}^{D-1}(x^{ei}y - x)$. We can compute this polynomial by divide and conquer method in $\widetilde{\mathcal{O}}(D)$ time, and evaluate it at D points (namely at $y = x^{ebD}$ for $0 \leq b < D$) in $\widetilde{\mathcal{O}}(D)$ time, using fast evaluation and interpolation algorithms in Chapter 10 of [JGG13].

Given those values, we can calculate their gcd with N, and get the factorization of N with high probability.

3.5.4 BD98's attack when $d \leq N^{0.292}$

In 1998, Boneh *et al.*[BD98] showed an attack when $d \leq N^{0.292}$. Their method based on finding small roots of polynomial in $\mathbb{Z}_n$ (which is not a field when n is composite). Their analysis also implies that an attack when $d \leq N^{0.5}$ is possible. However finding such attack is leaved open till now.

The basic idea of BD98's attack is to reduce key recovering problem to small inverse problem. Specifically, let $N = pq$ and $g = \gcd(p-1, q-1)$. Then there exists a integer k such that

$$ed + k\left(\frac{N+1}{g} - \frac{p+q}{g}\right) = 1.$$

Let $s = -\frac{p+q}{g}$ and $A = \frac{N+1}{g}$, we have,

$$k(A+s) \equiv 1 \mod e, \tag{3.3}$$

where e and A are given. The attack is to find all pair of integers k, s satisfying equation 3.3, such that $|k| < e^{\delta}, |s| < e^{0.5}$. Note that when $\delta < 0.5$, the expectation number of solutions is no more than 1, which means that if we can solve the small inverse problem for $\delta = 0.5$, we can attack on RSA when $d \leq N^{0.5}$.

BD98's attack is able to solve small inverse problem for $\delta = 1 - \frac{\sqrt{2}}{2} = 0.292$.

3.5.4.1 Solving the small inverse problem

Let $f(x, y) = x(A+y) \mod e$. The main tool is stated in the following fact.

Fact 3.16 (HG98 [H97][BD98]) *Given a polynomial $h(x,y) = \sum_{i,j} a_{i,j}x^iy^j$, define $||h(x,y)||^2 = \sum_{i,j}|a_{i,j}^2|$.*

Let $h(x,y) \in \mathbb{Z}[x,y]$ be a polynomial which is a sum of at most w monomials. Suppose that

- *$h(x_0, y_0) = 0 \mod e^m$ for some positive integer m where $x_0 < X$ and $y_0 < Y$, and*
- *$||h(xX, yY)|| < e^m/\sqrt{w}$.*

Then $h(x_0, y_0)$ holds over the integers.

Proof of fact 3.16 is omitted here and can be found in [BD98].

Fact 3.16 is crucial in solving small roots of a polynomial modulo e because, once we can construct a polynomial with small norm that has (x_0, y_0) as its root, we can instead solve this polynomial over $\mathbb{Z}$ (without modulo). And solving polynomial over $\mathbb{Z}$ is well studied and can be done in polynomial time[3].

To do so, construct monomials bases as,

$$g_{i,k} = x^i f^k(x,y)e^{m-k} \quad \text{and} \quad h_{j,k}(x,y) = y^j f^k(x,y)e^{m-k}.$$

Note that for (x_0, y_0) where $f(x_0, y_0) \equiv 0 \mod e$, we have $g_{i,k}(x_0, y_0) \equiv h_{j,k}(x_0, y_0) \equiv 0 \mod e^m$. The problem becomes finding a integer combination $t(x, y)$ of $g_{i,k}(x, y)$ and $f_{i,k}(x, y)$ such that the norm of $t(x, y)$ satisfies $||t(xX, yY)|| < e^m/\sqrt{w}$, where $X = e^\delta, Y = e^{0.5}$. Once such t is found, we can solve $t(x, y) = 0$ over $\mathbb{Z}$ and list all its solutions as our guess for (k, s) in equation 3.3. This is known as shortest vector problem in lattice[4]. LLL algorithm [LLL82] is used to obtain a constant factor approximation.

Boneh *et al.*constructed the corresponding lattice, and by some improvement, a bound $N^{0.292}$ is achieved.

[3]For example, by Hensel lifting in Chapter 15 of [JGG13].

[4]`https://en.wikipedia.org/wiki/Lattice_problem#Shortest_vector_problem_(SVP)`

Appendix A

Now we prove lemma 3.13.

On the one hand, suppose a is a q-th residue of p, i.e., there exists $b \in \mathbb{Z}_p^*$ such that $b^q \equiv a \mod p$. Let $x = ind(a)$ and $y = ind(b)$. Then we have,

$$g^{yq} \equiv g^x \mod p,$$

which by Fermat's theorem is equivalent to

$$yq \equiv x \mod p-1.$$

Note that $q \mid p-1$, therefore we have, $q \mid x$.

On the other hand, suppose $q \mid ind(a)$, let $b = g^{ind(a)/q} \mod p$. Then we have $b^q \equiv a \mod p$, which means that a is a q-th residue mod p.

[A52] Ankeny, N. C. "The Least Quadratic Non Residue." Annals of Mathematics 55.1(1952):65-72.

[B99] Boneh, Dan. "Twenty Years of Attacks on the RSA Cryptosystem." Notices of the American Mathematical Society 46.2(1999):203–213.

[BD98] Dan, Boneh, and G. Durfee. "New Results on the Cryptanalysis of Low Exponent RSA (Extend Abstract, Preprint)." (1997).

[G75] Gary L. Miller. "Riemann's hypothesis and tests for primality *." Journal of Computer & System Sciences 13.3(1975):300-317.

[JGG13] Joachim, Von Zur Gathen,, and J. Gerhard. Modern computer algebra, 2013:7-16.

[LLL82] Lenstra, A. K., H. W. L. Jr, and L. Lovsz. "Factoring polynomials with rational coefficients." Mathematische Annalen 261.4(1982):515-534.

[H97] N. Howgrave-Graham, "Finding small roots of univariate modular equations revisited", Proc of Cryptography and Coding, LNCS 1355, Springer-Verlag, 1997, pp. 131-142.

[RSA78] Rivest, R. L., A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems." Communications of the Acm 26.2(1978):96-99.

[W89] Wiener, Michael J. Cryptanalysis of Short RSA Secret Exponents. Advances in Cryptology EUROCRYPT 89. Springer Berlin Heidelberg, 1989:553-558.