# Cryptography Project: Modular $e$-th Root in RSA

Chenghao Guo, 2016012393, Xinzhi Zhang, 2016010253

January 20, 2019

## 1 Introduction

The modular $e$-th root problem is as follows: Given $e, c$ and $n$, find $x \in \mathbb{Z}_n$ such that

$$x^e = c \ mod \ n$$

In this report, we discuss three advanced problems relate with finding modular $e$-th root:

- Find modular $e$-th root when $e$ and $n$ are not necessarily prime.
- Comparison of difficulty between finding modular $e$-th root and factorization.
- Recent progress in cryptoanalysis of RSA.

The first problem were discussed in [7], but the method of testing $e$-th residue for general $N$ was not given. In the second section, we will solve this problem and give a full algorithm for calculating $e$-th root. However, the algorithm is not practical since it may take exponential time and relies on factorization.

In the third section, we will present two results [2] and [6]. Proof of the first will be discuss in detail while only the main results will be presented for the second.

In the fourth section, an overview of recent progresses on attacking RSA is firstly given. Then a specific type attack is introduced in detail, which works when a fraction of private key is exposed.

## 2 Advanced Problem 1: $E$-th root calculation modulo $N$ general with known factorization

Through this section, let the number field considered to be $N$, and we want to see if $a \in \mathbb{Z}/N\mathbb{Z}$ is an $E$-th power residue.

### 2.1 E can be prime without loss of generality

**Lemma 1.** *let $a_E$ be the set of E-th root of a modulo N, then $a_E = \cup_{x \in a_{E_1}} x_{E_2}$.*

*Proof.* For any $y \in a_E$, $(y^{E_1})^{E_2} = a$, so $y^{E_2} \in a_{E_1}$. And for any $y \in x_{E_2}$ where $x \in a_{E_1}$, $Y^E = (y^{E_1})^{E_2} = a$. □

Therefore, if $E$ is composite, we can factorize $E = E_1 E_2 \cdots$ and calculate $a_E$ by calculating the $a_{E_1}$ and $x_{E_2}$ for each $x \in a_{E_1}$ and so on.

In the following of this section, $E$ is always assumed to be prime.

Note that this procedure itself can take exponential time when $E$ is exponentially large, since there are at most $E$ different roots.

## 2.2 Testing $E$-th power residue

### 2.2.1 Circular Group Case

**Lemma 2.** *The group $(\mathbb{Z}/N\mathbb{Z}, *)$ is cyclic if and only if $N$ is $1, 2, 4, p^k$ or $2p^k$ where $p$ is an odd prime.*

Suppose the group size is $|G|$, the testing algorithm is as follows:

- if $E \big| |G|$, $a$ is an $E$-th power residue if and only if $a^{\frac{|G|}{E}} = 1$. (an extension of Euler's criteria)
- if $E \not| |G|$, any $a \in G$ is an $E$-th power residue.

Now we prove the correctness.

*Proof.* In the second case, the $E$-th root of $a$ is simply $a^{E^{-1}(\mod |G|)}$.

Now consider the first case, suppose the generator is $g$. On the one hand, if $a$ is a $E$-th power residue, and one of the root is $g^x$. Then $a = g^{xe}$, so $a^{\frac{|G|}{e}} = a^{x|G|} = 1$. On the other hand, if $a^{\frac{|G|}{e}} = 1$, we can assume $a = g^y$ and then $g^{\frac{y}{e}|G|} = 1$ which makes $\frac{y}{e}$ an integer and therefore $g^{\frac{y}{e}}$ is a $E$-th root of $a$. □

### 2.2.2 General $N$

The idea is to factorize $N$ and try to test whether $a$ is $E$-th power residue in each subgroup.

**Lemma 3.** *Suppose $N = n_1 n_2 \cdots n_m$, then $a$ is an $E$-th power residue modulo $N$ if and only if $a$ is an $E$-th power residue modulo $n_1$, $n_2 \cdots n_m$.*

*Proof.* Obvious through Chinese Remainder Theorem. □

Suppose $N = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$, we choose $n_i = p_i^{k_i} (1 \le i \le m)$. Therefore if $p_i$ is an odd prime, then test if $a$ is $E$-th power residue with method from previous section, and if $p_i = 2$, we could apply the following theorem:

**Theorem 1.** *$\mathbb{Z}/2^k\mathbb{Z} \cong C_2 \times C_{2^{k-2}}$ the generator are respectively -1 and 5.*

$a$ is an $E$-th power residue modulo $2^k$ if and only if it is an $E$-th power residue in both the group generated by -1 and the group generated by 5. Note that the method in previous section apply not just on the multiplicative modular group but on general circular group, the same technique can therefore determine whether $a$ is an $E$-th power residue modulo $2^k$.

## 2.3 Calculating $E$-th root

The algorithm of calculating $E$-th root modulo a prime $N$ is an extension of Cipolla's algorithm, as shown below:

- Randomly pick $b \in \mathbb{Z}/N\mathbb{Z}$ until $b^E - a$ is an $E$-th power nonresidue. Let $\omega = \sqrt{E} b^E - a$.
- Return $(a - \omega)^{p^{e-1} + p^{e-2} + \cdots + 1}/e$ as an $E$-th root of $a$.

For prime power, we could apply a power lifting technique to get $E$-th roots of higher power. And for a general composite $N$, we can first factorize $N$, find $E$-th roots in each prime power, then combine them using Chinese Remainder Theorem.

Proof of the correctness and the detail of power lifting technique could be found in [7].

A complete algorithm of solving the problem is as follows:

**Algorithm 1** Solving $E$-th root modulo $N$

**Require:** $N, E$ and $a$.
  Factorize $N$ and $E$, suppose $N = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$ and $E = E_1 E_2 \cdots E_n$
  Let $R_0 = \{a\}$
  **for** each $i \in 1..n$ **do**
    Let $R_i = \emptyset$
    **for** each element $r$ in $R_{i-1}$ **do**
      **for** each $j \in 1..m$ **do**
        **if** $E_i \not| \phi(p_j^{k_j})$ **then**
          Let $S_j = \{r^{E_i^{-1}(\mod \phi(p_j^{k_j}))} (\mod p_j^{k_j})\}$
        **else**
          **if** $r^{\frac{\phi(p_j^{k_j})}{E_i}} \neq 1 (\mod p_j^{k_j})$ **then**
            $S_j = \emptyset$.
          **else**
            Use extended Cipolla's algorithm to find all $E$-th roots of $r$ modulo $p_j$.
            Use power lifting to find all $E$-th roots of $r$ modulo $p_j^{k_j}$ as $S_j$
          **end if**
        **end if**
      **end for**
      $R_i \leftarrow R_i \cup \left( \times_{j=1}^{m} S_j \right)$.
    **end for**
  **end for**
  **if** $R_n \neq \emptyset$ **then**
    **return** $R_n$ as all $E$-th roots of $a$.
  **else**
    **return** $a$ has no $E$-th roots.
  **end if**

# 3 Advanced Problem 2: Relationship Between Breaking RSA and Factoring

In this section, we give brief introduction several results regarding the relationship between finding modular $E$-th root and factoring. Since this problem is equivalent to breaking RSA system $m = c^{1/E} \ mod \ N$, in the following discussion, we only consider the relationship between breaking RSA and factoring.

## 3.1 Results Overview

There are three assumptions regarding the difficulty of finding modular $E$-th root and factorization:

- *Factoring assumption:* There does not exist a polynomial time-probabilistic algorithm, that, given $N$, finds a non-trivial factor of $N$ with non-negligible probability.
- *RSA assumption:* There does not exist a polynomial time-probabilistic algorithm, that, given a pair $(N, E)$ and an element $a$ chosen uniformly random in $\mathbb{Z}_N^*$, computes $x \in \mathbb{Z}_N^*$ such that $x^E = a \ mod \ N$, with non-negligible probability.

- *Generic RSA assumption:* There does not exist a polynomial time-probabilistic *generic ring algorithm*, that, given a pair $(N, E)$ and an element $a$ chosen uniformly random in $\mathbb{Z}_N^*$, computes $x \in \mathbb{Z}_N^*$ such that $x^E = a \ mod \ N$, with non-negligible probability.

In 2009, Aggarwal and Maurer [2] proved the correlation between generic RSA assumption and factoring assumption:

**Theorem 2.** *[2] For any pair $(N, E)$, the factoring assumption holds for $N$ implies that the generic RSA assumption holds for $(N, E)$.*

Since generic ring algorithm is a subset of probabilistic algorithm, Theorem 2 is weaker than the proposition that Factoring assumption $\Rightarrow$ RSA assumption, which implies that factoring and breaking RSA has the same difficulty.

The contra-positive proposition of this theorem shows the existence of an efficient reduction from an efficient generic ring algorithm to an efficient factorizing algorithm. Theorem 2 also implies that any efficient algorithm that can break RSA system and does not need to factorize must be non-generic, assuming that factoring is hard.

In the following subsection, we will discribe the model of generic computation and the proof of Theorem 2 in detail.

## 3.2 Proof Sketch of Generic RSA assumption $\Rightarrow$ Factoring Assumption

### 3.2.1 Generic Ring Algorithm

First introduced in 2005, the generic model of computation [12] can be viewed as a blackbox **B** which can store variables in a certain ring $R$ as internal variables $V_0, V_1, \cdots, V_\ell$. The initial state (which can be regarded as the input of the algorith) consists of the values of $\{V_0, \cdots, V_\ell\}$ and follows some probability distribution.

The black box **B** consists two types of computations:

- *Computational operations* The set $\Pi - \{+, -, \times, /\}$ *here / means the inverse of* $\times$ consists of ordinary binary ring operation on $R$. A computational operation amounts to select three state variables $V_{i1}, V_{i2}, V_{i3}$ and $f \in \Pi$. The black box **B** computes $f(V_{i1}, V_{i2})$ and stores the result in $V_{i3}$.

- *Relation queries* The set $\Sigma = \{=\}$ consists of the ordinary equality relation. We pick $\rho \in \Sigma$ and two state variables $V_{i1}, V_{i2}$. The result of the query is $\rho(V_{i1}, V_{i2})$. It assumes the value 1 if the relation is satisfied, and 0 otherwise.

A generic ring algorithm (GRA) is defined as a (probabilistic) algorithm interacting the black box **B** above. Consider $R$ as the integral ring $\mathbb{Z}_n$, and suppose the black box **B** only has one input $x \in \mathbb{Z}_n$. Then every state variable $V_i$ can be represented as an element in the rational fraction field of the polynomial ring $\mathbb{Z}_n[x]$.

Note that some elements in $\mathbb{Z}_n$ does not have inversion thus the division is not well-defined, in the following discussion we actually use an equivalent definition which avoids the division problem: Consider $R = \mathbb{Z}_n \times \mathbb{Z}_n$, every state variable $V_i$ is a pair of elements in $\mathbb{Z}_n[x]$ $(P(x), Q(x))$ which can be regarded as the (nominator, denominator). The computational operations and relation query

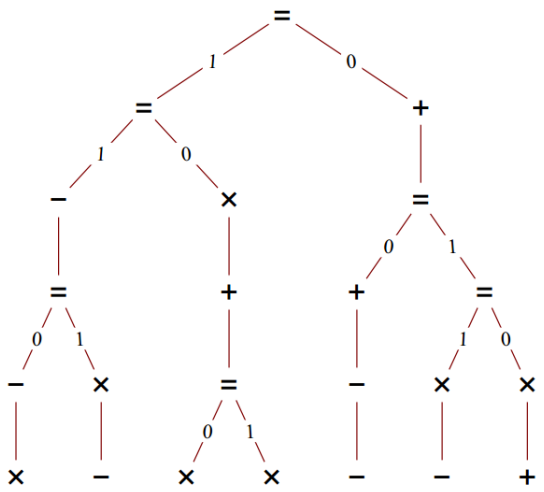

Figure 1: An example of $T^G$

are defined as:

$$((P_i, Q_i), (P_i', Q_i'), \circ_i) = \begin{cases} (P_iQ_i' + P_i'Q_i,\ Q_iQ_i'), & \text{if } \circ_i \text{ is } + \\ (P_iQ_i' - P_i'Q_i,\ Q_iQ_i'), & \text{if } \circ_i \text{ is } - \\ (P_iP_i',\ Q_iQ_i'), & \text{if } \circ_i \text{ is } \times \\ (P_iQ_I',\ P_i'Q_i), & \text{if } \circ_i \text{ is } \div \\ \mathbb{I}((P_i, Q_i) = (P_i', Q_i')), & \text{if } \circ_i \text{ is } = \end{cases} \tag{1}$$

Note that bit manipulation (such as XOR) cannot be represented by a GRA, thus the set of GRAis smaller than the set of probabilistic algorithms.

For all GRA, it can decompose into probabilitstic combination of deterministic GRAs (Deterministic GRA). For a Deterministic GRA$G$, we can construct a binary tree $T^G$ from $G$ as follows: Each step performed by $G$ involves an operation from the set $\Pi \cup \Sigma$. For all operations in $G$, if the operation is a computational step, we use a node with one child representing the computation; if the operation is a equality query, we use a node with two children, with their edges labeled 0 and 1 respectively, to represent the operation. The first step in $G$ is the root in the corresponding $T^G$, and for each operation in $G$ we append a node in $T^G$ from its previous operation. The input $x \in \mathbb{Z}_n$ first assigns value to the variables of the corresponding operation in the root, and then transmit values to the variables of its children, and so on. Note that each fixed input has a unique information flow in the tree, which is a path from the root to a leaf, the output of $G$ should be a probabilistic combination (over the distribution of input $x$) of all the results of the leaf nodes.

An example of the tree representation of a GRAis Figure 1.

Each path in $T^G$ from the root to a leaf can be regarded as a straight-line program (SLP). Here we give a formal definition of SLP:

**Definition 1.** *An $L$-step straight line program (*SLP*) is a sequence of triples $(i_1, j_1, \circ_1),\ (i_2, j_2, \circ_2), \cdots,\ (i_L, j_L, \circ_L)$ such that $i_k, j_k \in \mathbb{Z}_n[x] \times \mathbb{Z}_n[x]$ and $\circ_k \in \{+, -, \times, /\}$.*

An property of SLPis that its output degree is uppper-bounded:

**Lemma 4.** *Suppose $(P(x), Q(x))$ is the output of an $L$-step* SLP*. Then $P$ and $Q$ are polynomialas of degree at most $2^k$.*

*Proof.* By induction on $L$. The claim is trivially true for $L = 0$. Suppose $P_r(x)$ and $Q_r(x)$ are polynomials of degree at most $2^r$ for all $r < L$. Then since $i_k, j_k \leq L-1$, $deg(P_{ik}), deg(P_{jk}), deg(Q_{ik}), deg(Q_{jk})$ are all at most $2^{L-1}$, then by definition of $P_L$, $Q_L$, the degree of $P_L$ and $Q_L$ are at most $2^{L-1}+2^{L-1} = 2^L$. □

### 3.2.2 Mathematical Preliminaries

In the following discussion, we assume $n = pq$ where $p$ and $q$ are prime numbers. Denote by $a =_c b$ that $a$ is congruent to $b$ modulo $c$.

**Definition 2.** *$\forall P \in \mathbb{Z}_n[x]$, denote $\nu_n(P)$ as the fraction of roots of $P$, i.e.*

$$\nu_n(P) = |\{x \in \mathbb{Z}_n \mid P(x) =_n 0\}|/n$$

**Definition 3.** *$\forall P \in \mathbb{Z}_n[x]$, denote $\eta_n(P)$ as the fraction of elements of $P$ which has nontrivial gcd with $n$, i.e.*

$$\eta_n(P) = |\{x \in \mathbb{Z}_n \mid gcd(P(x), n) \notin \{1, n\}\}|/n$$

We can interpret it $\eta_n(P)$ as the probability to yield $n$'s factorization from $gcd(P(x), n)$ when $x$ is uniformly sampled from $\mathbb{Z}_n$.

**Lemma 5.** *For any $P(x) \in \mathbb{Z}_n[x]$, if $\nu_n(P) \in [\delta, 1-\delta]$, then $\eta_n(P) \geq \delta^{3/2}$.*

*Proof.* By Chinese Remainder Theorem, $\nu_n(P) = \nu_p \cdot \nu_q$ and $\eta_n(P) = \nu_p(1-\nu_q) + \nu_q(1-\nu_p)$. Since $\delta \leq \nu_p \cdot \nu_q \leq 1-\delta$,

$$\begin{aligned}
\eta_n(P) &= \nu_p(1-\nu_q) + \nu_q(1-\nu_p) \\
&= \nu_p + \nu_q - 2\nu_p \cdot \nu_q \\
&= (\sqrt{\nu_p} - \sqrt{\nu_q})^2 + 2\sqrt{\nu_p \cdot \nu_q} - 2\nu_p \cdot \nu_q \\
&\geq 2\sqrt{\nu_p \cdot \nu_q} - 2\nu_p \cdot \nu_q \\
&= 2\sqrt{\nu_p \cdot \nu_q}(1 - \sqrt{\nu_p \cdot \nu_q}) \\
&\geq 2\sqrt{\delta}(1 - \sqrt{1-\delta}) \\
&\geq 2\delta(1 - (1 - \frac{\delta}{2})) = \delta^{3/2}
\end{aligned}$$

□

**Lemma 6.** *Let $p$ be a prime. A random monic polynomial $f(x) \in \mathbb{Z}p[x]$ of degree $d$ is irreducible in $\mathbb{Z}_p[x]$ with probability at least $\frac{1}{2d}$ and has a root in $\mathbb{Z}_p$ with probability at least $\frac{1}{2}$.*

*Proof.* From the distribution theorem of monic polynomials [11], the number of irreducible monic polynomials of degree $d$ over $F_p[x]$ (where $F_p$ is a finite field of degree $p$) is at least $\frac{p^d}{2d}$. Therefore $f(x)$ is an irreducible polynomial over $\mathbb{Z}_p$ with probability at least $\frac{1}{2d}$.

From the principle of inclusion of exclusion and by enumerating the number of roots, the number of monic polynomials which has at least one root is:

$$\sum_{i-1}^{d}(-1)^{i-1}\binom{p}{i}p^{d-i} \geq p \cdot p^{d-1} - \binom{p}{2}p^{d-2} \geq \frac{p^d}{2}$$

Therefore, the probability that $f(x)$ has a root in $\mathbb{Z}_p$ is at least $\frac{1}{2}$ □

**Algorithm 1.** Factoring Algorithm

**Input**: $n$, SLP $S_1$
**Output**: A factor of $n$

1 Choose a monic polynomial $h(x)$ uniformly at random from all monic polynomials of degree $d$ in $\mathbb{Z}_n[x]$;
2 Compute $h'(x)$, the derivative of $h(x)$ in $\mathbb{Z}_n[x]$;
3 Choose a random element $r(x) \in \mathbb{Z}_n[x]/h(x)$;
4 Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$ using SLP $S_1$;
5 Run Euclid's algorithm in $\mathbb{Z}_n[x]$ on $h(x)$ and $z(x)$. If this fails return $\gcd(n, H(h(x), z(x)))$;
6 Run Euclid's algorithm in $\mathbb{Z}_n[x]$ on $h(x)$ and $h'(x)$. If this fails return $\gcd(n, H(h(x), h'(x)))$;

Figure 2: Algorithm for Lemma 7

For polynomials $b(x), c(x) \in \mathbb{Z}_n[x]$, let $gcd_p(b(x), c(x))$ and $gcd_q(b(x), c(x))$ denote the greatest common divisor of $b(x)$ and $c(x)$ in $\mathbb{Z}_p[x]$ and $\mathbb{Z}_q[x]$. When $n = pq$ is not a prime, the Euclid's algorihm running on $b(x)$ and $c(x)$ may fail since $\mathbb{Z}_n[x]$ is not a Euclid's domain. The following proposition shows failing condition and implication:

**Proposition 1.** *Let $b(x), c(x) \in \mathbb{Z}_n[x]$. Then*

- *If Euclid's algorithm running on $b(x)$ and $c(x)$ fails, then the last step of the algorithm yields a non-trivial non-invertible element of $\mathbb{Z}_n$. We denote this element as $H(b(x), c(x))$.*
- *If $deg(g_p(b(x), c(x))) \neq deg(g_q(b(x), c(x)))$, then Euclid's algorithm, when running on $b(x)$ and $c(x)$, fails.*

### 3.2.3 Proof

**1st step: From SLPto factorization** The first step of the proof gives an efficient algorithm that, with non-negligible probability, computes $n$'s non-trivial factor, given access to an $L-$step SLP, which computes the $e-$th root of an element chosen uniformly from $\mathbb{Z}_n$, with non-negligible probability. The main result is as follows:

**Lemma 7.** *For all $\epsilon > 0, \mu > 0, L \in \mathbb{N}$, there exists an algorithm of time complexity $O(L^e + log^e(\epsilon))$ that, for every* SLP *$S$ such that $\nu_n(P(x)^e - xQ(x)^e) \geq \mu$ and $Q(x)$ is not the zero polynomial, returns a factor of $n$ with probability $\frac{\mu}{8(L+log(e))}$.*

*Proof.* Let $f(x) = P(x)^e - xQ(x)^e$, then $\nu_n(f) \geq \mu$. From Lemma 4, $\deg(f) \leq 2^L e + 1$. Let $d = L + \lceil log(e) \rceil$. The factoring algorithm is as follows:

From Proposition 1, if the Euclid's algorithm in line 5 or line 6 fails, the algorithm will yield a prime factor of $n$. Now we lower-bound the success probability.

From Lemma 6, the probability that $h(x)$ has a root (say, $s$) in $\mathbb{Z}_p[x]$ and is irreducible in $\mathbb{Z}_q[x]$ is at least $\frac{1}{2} \cdot \frac{1}{2d} = \frac{1}{4d}$. We only consider $h(x)$ conditional to cases above.

Since $s$ is $h(x)$'s root in $\mathbb{Z}_p[x]$, $(x-s)|h(x)$ in $\mathbb{Z}_p[x]$. There are two cases regarding $s$'s multiple:

- $(x-s)^2 \mid h(x)$ *in* $\mathbb{Z}_p[x]$: In this case, $(x-s) \mid gcd_p(h(x), h'(x))$. Therefore $(x-s) \mid gcd_p(h(x), h'(x))$ and $deg(gcd_p(h(x), h'(x))) \geq 1$.

But on the other hand, since $h(x)$ is irreducible in $\mathbb{Z}_q[x]$ and $h'(x)$ has smallaer degree than $h(x)$, $deg(gcd_q(h(x), h'(x))) = 0$.

Therefore, Euclid's algorithm fail on line 6 and yields a prime factor of $n$.

- $(x-s)^2 \not| \; h(x)$ *in* $\mathbb{Z}_p[x]$: In this case, let $h(x) = (x-s)h_1(x)$. From elementary algebra:

$$\begin{aligned}\mathbb{Z}_n[x]/h(x) &\cong \mathbb{Z}_p[x]/h(x) \times \mathbb{Z}_q[x]/h(x) \\ &\cong \mathbb{Z}_p[x]/(x-s) \times \mathbb{Z}_p[x]/h1(x) \times \mathbb{F}_{q^d}\end{aligned}$$

where $\mathbb{F}_{q^d}$ is an arbitrary finite field containing $q^d$ elements, which is isomorphic to $\mathbb{Z}_q[x]/h(x)$ since $h(x)$ is irreducible.

Suppose the isomorphism above maps $r(x) \in \mathbb{Z}_n[x]$ to a triple $r(s) \; mod \; p, u(x), r_q(x)$ and maps $z(x) \in \mathbb{Z}_n[x]$ to a triple $z(s) \; mod \; p, u(x), z_q(x)$ Since $r(x)$ is uniformly random in $\mathbb{Z}_n[x]/h(x)$, $r(s)$ is uniformly random in $\mathbb{Z}_p$, we have

$$P(z(s) =_p 0) = P(f(r(s)) =_p 0) \geq P(f(r(s)) =_n 0) \geq \mu$$

Therefore, with probability at least $\mu$, $(x-s) \mid z(x)$ and $deg(gcd_p(h(x), z(x))) \geq 1$.

On the other hand, since $r(x)$ is uniformly random in $\mathbb{Z}_n[x]/h(x)$, $r_q(x)$ is uniformly random in $\mathbb{Z}_q[x]/h(x) \cong \mathbb{F}_{q^d}$. From the fact that $f(x)$ has degree at most $2^L e + 1$, for $x$ uniformly random in $\mathbb{Z}_q$

$$P(z_q(x) = 0) = P(f(r_q(x)) = 0) \leq \frac{deg(f)}{q^d} \leq \frac{2^L e + 1}{q^d} \leq \frac{1}{2}$$

when $d = L + \lceil log(e) \rceil$. Therefore, the probability that $z(x)$ is a nonzero polynomial is greater than $\frac{1}{2}$, which implies that $Pr(deg(gcd_q(z(x), h(x))) = 0) \geq \frac{1}{2}$.

To summarize, the probability that Euclid's algorithm fail on line 5 or 6 is at least $\frac{1}{4d} \cdot \frac{\mu}{2} = \frac{\mu}{8(L+\lceil log(e) \rceil)}$.

Now compute the time complexity of the algorithm. Generating fandomg $h(x)$ and $r(x)$ whose degree is at most $d$ takes $O(d)$ time. Since $P^e(x), Q^e(x)$ can be computed in $2\lceil loge \rceil$ computational steps each, $f(x)$ can be computed in $L + 4\lceil log(e) \rceil + 2$ steps. Also since each operation in $\mathbb{Z}_n[x]/h(x)$ can be implemented by at most $d^2$ operations in $\mathbb{Z}_n$, we have that line 4 of the algorithm takes $O(d^2 L + d^2 log(e)) = O(d^3)$ time. Line 5 and 6 of the algorithm, which is Eculid's algorithm on $(z(x), h(x))$ and $(h(x), h'(x))$ takes at most $O(d^2)$ time. Therefore, the total running time is $O(d^3)$. □

**Step 2: From GRA to SLP or Factorization:** The second step of the proof aims to give an efficient algorithm that, given access to a deterministic RSA-GRAthat computes $e-$th root in $\mathbb{Z}_n$, outputs either the factorization of $n$ or an SLPwith high probability. The main result is as follows:

**Definition 4.** *For a* Deterministic GRA *G, let* $\lambda_n(G)$ *denote the probability that G, when running on an input x chosen uniformly at random from* $\mathbb{Z}_n$*, is successful in computing the e-th root of x.*

**Algorithm 2.**

**Input**: GRA $G$, $n$
**Output**: A factor of $n$ or an SLP $S$

Initialize $S$ to be the empty sequence;
**for** $k \leftarrow 2$ **to** $L$ **do**
  Get the operation $\{i_k, j_k, \circ_k\}$ from $G$;
  **if** $\circ_k \in \{+, -, \cdot, /\}$ **then**
    Append $\{i_k, j_k, \circ_k\}$ to $S$;
  **else** /* Here, $\circ_k$ is $eq$ */
    Append $\{k-1, 0, \cdot\}$ to $S$;
    **for** $i \leftarrow 1$ **to** $M$ **do**
      Generate a random element $x \in_R \mathbb{Z}_n$;
      Compute $g = \gcd\left(P^S_{i_k}(x) \cdot Q^S_{j_k}(x) - P^S_{j_k}(x) \cdot Q^S_{i_k}(x), n\right)$;
      **if** $g \notin \{1, n\}$ **then** return $g$;
    **end**
    Generate a random element $x' \in_R \mathbb{Z}_n$;
    **if** $P^S_{i_k}(x') \cdot Q^S_{j_k}(x') - P^S_{j_k}(x') \cdot Q^S_{i_k}(x') = 0$ **then** return the bit 0 to $G$
    **else** return the bit 1 to $G$;
  **end**
**end**
Return $S$;

Figure 3: Algorithm for Lemma 8

**Lemma 8.** *For all $\epsilon > 0, L \in \mathbb{N}$, there exists an algorithm of time complexity $O((\frac{L}{\epsilon})^{5/2})$ that, given access to an $L$-step* DETERMINISTIC GRA$G$*, with prpobability $1-\epsilon$, either outputs a factor of $n$ or an $L$-step* SLP$S$ *such that $\nu_n(P^e(x) - xQ^e(x)) \geq \lambda_n - \frac{\epsilon}{2}$.*

*Proof.* The algorithm is as Figure shows:

Suppose $T^G$ is DETERMINISTIC GRA$G$'s tree representation. Let $\delta = \frac{\epsilon}{2L}$, we first classify the vertices corresponding to equality queries in $T^G$ into two kinds:

**Definition 5.** *Let $v$be a vertex corresponding to a equality query $(i_k, j_k, eq)$ in $T^G$. If*

$$\nu_n(P^e(x) - xQ^e(x)) \in [\delta, 1-\delta],$$

*then we call $v$ a non-extreme vertex. Otherwise we call $v$ as an extreme vertex.*

Let $M = \lceil \frac{L^{3/2}}{(\epsilon/2)^{5/2}} \rceil$ We now compute the success probability of the algorithm. There are two possible cases depending on whether the equality vertex $v$ we meet. Denote the two inputs of the equality query as $i_v = (P(x), Q(x)), j_v = (P'(x), Q'(x))$

- $v$ is a non-extreme vertex: Let $g(x) = P(x)Q'(x) - P'(x)Q(x)$. In this case $\nu_n(g) \in [\delta, 1-\delta]$. According to Lemma 5, $\eta_n(g) \geq \delta^{3/2}$. Therefore, if we sample $M$ inputs $x$ and compute $gcd(n, g(x))$ respectively, the probability that a nontrivial factor is returned is at least

$$1 - (1-\delta^{3/2})^M \geq 1 - exp(-(\delta^{3/2})M) = 1 - exp(\frac{-2}{\delta}) \geq 1 - \frac{\epsilon}{2}$$

- $v$ is an extreme vertex: In this case, $\nu_n(g) < \delta$ or $\nu_n(g) > 1-\delta$. Therefore, if we sample $x$ uniformly from $\mathbb{Z}_p$ and compute $\mathbb{I}(g(x) = 0)$, then with probability less than $\delta$, we will obtain a branch that at most $\delta$ fraction of inputs will go on.

If we do note meet non-extreme vertex, in the end we will obtain a path in $T^G$, which is a SLPwith at most $L$ steps. From the above analysis, with probability at least $1 - \delta L = 1 - \frac{\delta}{2}$, at least $1 - \delta L$ fraction of inputs run on this SLPto obtain (possibly) its modular $e$-th root.

Therefore, if the factoring algorithm is not successful, then with probability $1 - \frac{\epsilon}{2}$, we can obtain a SLPwhose success probability is at least $\lambda_n - \frac{\epsilon}{2}$.

Now we analyse the time complexity of the algorithm. Since the loop in steps 8-12 and steps 5, 13 and 14 are executed at most $L$ steps, the time complexity of the algorithm is $O(LM) = O((\frac{L}{\epsilon})^{5/2})$. □

**Combining step 1 and step 2:** Suppose there exists a randomized RSA-GRA$G$ that succeed in breaking RSA with probability $\mu_n$ on $\mathbb{Z}_n$ for sme $e > 1$. Then with probability $\mu_n - \frac{\epsilon}{2}$, Algorithm 2 either returns a factor of $n$ or an SLP$S$ that succeeds in breaking RSA with probability at least $\frac{\mu_n}{2}$, which can be converted into an algorithm that factors $N$ with probability $\frac{\mu_n}{16(L+log(e))}$.

Let $\epsilon = \frac{\mu_n}{2}$, we can guarantee that, one execution of Algorithm 2 followed by Algorithm 1 (if needed), returns a factor of $n$ with probability at least $\frac{\mu_n^2}{32(L+log(e))}$. Therefore, the expected number of times the two algorithms needed to be repeated to get a factor of $n$ is $\frac{32(L+log(e))}{\mu_n^2}$, and the expected time complexity of the factoring algorithm is $O((L^3 + log^3(e) + (\frac{L}{\mu_n})^{5/2}) \cdot (\frac{L+log(e)}{\mu_n^2}))$ which is polynomial in $L, log(e), \frac{1}{\mu_n}$. Therefore, the factoring algorithm is a polynomial time probabilistic algorithm that succeeds with non-negligible probability. This completes the proof of factoring assumption ⇒ generic RSA assumption.

### 3.3 Other Results

Another important result regarding the equivalence of breaking RSA and factoring is given by Boneh and Vankateson in 1998 [6]. It gives indirect evidence that breaking RSA when the exponent is small (say, $e = 3$) may be easier than factoring. It's main result is shown as follows:

**Theorem 3.** *Suppose there exists an algebraic factoring algorithm $\mathcal{A}$ whose running time is $T(n)$. Further suppose that each of the* RSA-SLP*s generated by $\mathcal{A}$ on input $N \in \mathbb{Z}_2(n)$ contains at most $O(logT(n))$ radical steps. Then there is a real factoring algorithm $\mathcal{B}$ whose running time is $T(n)^{O(1)}$ and factors all $N \in \mathbb{Z}_2(n)$ that $\mathcal{A}$ does.*

Here RSA-SLPis defined as a SLPthat contains not only ring operation, but also visits of an modular $E-$th root oracle, and radical steps is defined as the steps visiting the oracle:

**Definition 6.** *A straight line RSA program (*RSA-SLP*) $P$ is a sequence of algebraic expressions $1, c_1, c_2 \cdots, c_L$ such that for all $i \in [L]$, the expression $c_i$ is either $c_i = c_k \circ c_l$ for some $k, l < i$ and $\circ \in \{+, -, \times, /\}$ or $c_i = \sqrt[e]{c_k}$ for some $k < i$ and $e < \omega$.*

and radical step is defined as a computational step in RSA-SLPthat visits the modular $e$-th root oracle.

An implication of this result is that unless factoring is easy, any efficient algegbraic reduction from factoring to breaking LE-RSA can be converted into efficient factoring algorithm. Thus, breaking LE-RSA cannot be equivalent to factoring under algebraiac reductions, unless factoring is easy.

# 4 Survey on current RSA attacks

## 4.1 Attack Overview

### 4.1.1 Correlation Attacks

Correlation attack refer to the kind attack that explore the lack of randomness in the key generation process.

For instance, a large prime generator that is not random enough may generate shared primes for different public key $N$, making them easy to be attacked by simply calculating the GCDs. A worse scenario is when the attacker is able to guess the structure of key generation process and use the information to assist hacking. In [4], authors use the procedure above to attack a real RSA system and succeeded.

### 4.1.2 Coppersmith Attacks

Coppersmith's attack describes a class of attacks on RSA based on the Coppersmith method, which is a method to find small integer roots of univariate or bivariate polynomials modulo a given integer. This kind of attack could be applied when public exponent $E$ is small and cipher texts of several correlated messages is known.

Univariate Coppersmith method is presented as follows.

**Theorem 4.** *Let $N$ be an integer and $f \in \mathbb{Z}[x]$ be a monic polynomial of degree $d$ over the integers. Set $X = N^{1/d-\epsilon}$ for $\frac{1}{d} > \epsilon > 0$. Then given $N, f$ attacker, Eve, can efficiently find all integers $x_0 < X$ satisfying $f(x_0) \equiv 0(\mod N)$. The running time is polynomial in $\frac{1}{\epsilon}$ and $\log_2 N$.*

Some examples of attacks using univariate Copper smith method can be found on Wiki.

We will later discuss some attacks using bivariate Coppersmith method in detail.

### 4.1.3 Side Channel Attacks

Over the years, researches on attacking RSA public-key system through side channel has never stopped.

For instance, in [1], researchers applied a branch prediction technique. In [13], the authors performed the attack by varying the CPU power voltage limits.

Many of the contents of this overview could be found in [3].

## 4.2 Attack on RSA Given a Fraction of Private Key Bits

The problem of attacking RSA with a fraction of private key bits is investigated in many literatures [5, 9, 10]. These results are mostly based on Coppersmith method. We will present one of the earliest results [6] in this field which made use of bivariate Coppersmith method.

### 4.2.1 Results

A total of three attacks will be presented, their formal description is written as the following three theorems. The proofs are given in the next part of this subsection.

Throughout we assume $N = pq$ is an RSA modulus with $\sqrt{N}/2 < q < p < 2\sqrt{N}$. Further, define $k$ and $s$ as follows:

$$ed = k\phi(N) = ed - k(N - s + 1) = 1 \tag{2}$$

**Theorem 5.** ***(LSB)*** *Let $N = pq$ be an $n$-bit RSA modulus. Let $1 \leq d, d \leq \phi(N)$ satisfy $ed = 1 \mod \phi(N)$. There is an algorithm that given the $\frac{n}{4}$ least significant bits of $d$ computes all of $d$ in polynomial time in $n$ and $e$.*

**Theorem 6.** ***(MSB, $e$ prime)*** *Let $N = pq$ be an $n$-bit RSA modulus. Let $1 \leq d, d \leq \phi(N)$ satisfy $ed = 1 \mod \phi(N)$. Suppose $e$ is prime and $2^t \leq e < 2^{t+1}$ with $\frac{n}{4} \leq t \leq \frac{n}{2}$. Then given the $t$ most significant bits of $d$ there is an algorithm to compute all of $d$ in polynomial time in $n$.*

**Theorem 7.** ***(MSB, $e$ general)*** *Let $N = pq$ be an $n$-bit RSA modulus. Let $1 \leq d, d \leq \phi(N)$ satisfy $ed = 1 \mod \phi(N)$. Suppose $2^t \leq e < 2^{t+1}$ with $t \leq \frac{n}{2}$. Further, suppose $d > \epsilon N$ for some $\epsilon > 0$. Then given the $n - t$ most significant bits of $d$ there is an algorithm to compute all of $d$ in polynomial time in $n$ and $\frac{1}{\epsilon}$.*

### 4.2.2 Attack When Part of $p$ is Known

Before showing proof of the results, we first present an intermediate result, which made use of Copersmith's result on bivariate polynomial.

**Theorem 8.** ***(Coppersmith)[8]*** *Let $f(x, y)$ be a polynomial in two variables over $\mathbb{Z}$ with maximum degree $\delta$ in each variable seperately, and assume the coefficients of $f$ are relatively prime. Let $X, Y$ be bounds on the solutions $x_0$, $y_0$. Define $\tilde{f}(x, y) = f(Xx, Yy)$ and let $D$ be the absolute value of the largest coefficient. If $XY < D^{2/(3\delta)}$, then all integer pairs $(x_0, y_0)$ with $f(x_0, y_0) = 0$, $|x_0| < X$, $y_0 < Y$ can be solved in time polynomial in $\log D$ and $2^\delta$.*

The following lemma is an immediate consequence of this theorem.

**Lemma 9.** *Let $N = pq$ be an $n$-bit RSA modulus. Let $r \geq 2^{n/4}$ be given and suppose $p_0 := p \mod r$ is known. Then it is possible to factor $N$ in time polynomial in $n$.*

*Proof.* Define $q_0 = q \equiv N/p_0 \mod r$. We want to solve the following polynomial:

$$f(x, y) = \frac{(rx + p_0)(ry + q_0) - N}{r} = rxy + p_0 y + q_0 x + \frac{p_0 q_0 - N}{r}$$

where $0 \leq x_0 \leq 2^{n/2+1}/r = X$ and $0 \leq y_0 \leq 2^{n/2+1}/r = Y$. Note that $x_0 = \frac{p - p_0}{r}$ and $y_0 = \frac{q - q_0}{r}$ is a solution for the equation, we can factorize $N$ is this polynomial is solved.

To use Theorem. 8, let $\tilde{f}(x, y) = f(Xx, Yy) = rXYxy + Yp_0 y + Xq_0 x + \frac{p_0 q_0 - N}{r}$, then the absolute value of the largest coefficient is at least $rXY = 2^{n+2}/r$. So to meet the requirement of $XY < D^{2/(3\delta)}$, we have $r > 2^{(n+2)/4}$. By doing exhaustive search on the first two bits of $x_0$ and $y_0$, this can be reduced to $r \geq 2^{n/4}$. □

### 4.2.3 Proof of Theorem. 5

Statement of the theorem is:Let $N = pq$ be an $n$-bit RSA modulus. Let $1 \leq d, d \leq \phi(N)$ satisfy $ed = 1 \mod \phi(N)$. There is an algorithm that given the $\frac{n}{4}$ least significant bits of $d$ computes all of $d$ in polynomial time in n and $d$.

*Proof.* We can get $d_0 = d \mod 2^{n/4}$ from the least significant bits of $d$. By Equation 2,

$$ed_0 \equiv 1 + k(N - s + 1)( \mod 2^{n/4}).$$

Because $k \leq e$, we enumerate all possible $k$ from 0 to $e$, and for each candidate get $s \mod 2^{n/4}$ from the above equation. Then solve

$$p^2 - sp + N \equiv 0( \mod 2^{n/4}).$$

As a direct consequence of Lemma. 9, $N$ can be factorized through the solution of $p \mod 2^{n/4}$. □

### 4.2.4 Proof of Theorem. 6

Statement of the theorem is:Let $N = pq$ be an $n$-bit RSA modulus. Let $1 \leq d, d \leq \phi(N)$ satisfy $ed = 1 \mod \phi(N)$. Suppose $e$ is prime and $2^t \leq e < 2^{t+1}$ with $\frac{n}{4} \leq t \leq \frac{n}{2}$. Then given the $t$ most significant bits of $d$ there is an algorithm to compute all of $d$ in polynomial time in $n$.

*Proof.* Given the $t$ most significant bits of $d$, we could construct an estimate $d$ of $d_0$ such that $|d - d_0| < 2^{n-t}$. Because $k = \frac{ed-1}{\phi(N)}$, it is possible to estimate $k$ in Equation 2 through $d_0$. Let $k_0 = \frac{ed_0-1}{N}$. Then

$$|k_0 - k| = \left|(ed_0 - 1)(\frac{1}{\phi(N)} - \frac{1}{N} + \frac{e(d - d_0)}{\phi(N)})\right| < 2N^{3/2}\frac{N - \phi(N)}{\phi(N)N} + \frac{2N}{\phi(N)} < 40$$

The first inequality holds because $|e(d - d_0)| < 2^n \leq 2N$ and $ed_0 < 2^{2/n}N < 2N^{3/2}$. The second inequality holds because $N - \phi(N) < 4\sqrt{N}$ and $\phi(N) > N/2$.

Therefore we can estimate $k$ up to a constant error and enumerate all possible $k$. For each possible $k$, we do the following:

1. Compute $s \equiv N + 1 - k^{-1}( \mod e)$.
2. Find $p \mod e$ by solving the quadratic equation

   $$p^2 - sp + N = 0( \mod e).$$

   This is possible because $e$ is prime. So it could be solved in probabilistic polynomial time.
3. Because $e \geq 2^{n/4}$, we can apply Lemma. 9 to recover $p$.

□

### 4.2.5 Proof of Theorem. 7

Statement of the theorem is: Let $N = pq$ be an $n$-bit RSA modulus. Let $1 \leq d, d \leq \phi(N)$ satisfy $ed = 1 \mod \phi(N)$. Suppose $2^t \leq e < 2^{t+1}$ with $t \leq \frac{n}{2}$. Further, suppose $d > \epsilon N$ for some $\epsilon > 0$. Then given the $n-t$ most significant bits of $d$ there is an algorithm to compute all of $d$ in polynomial time in $n$ and $\frac{1}{\epsilon}$.

*Proof.* Compare to the previous theorem, more bits of $d$ are given, which adequately helps us to estimate $k$ up to a constant error. So we will still enumerate $k$, and for each candidate $k$, we do the following:

1. Compute $d_1 = e^{-1} \mod k$. This is possible since e and k are relatively prime. Since $ed - k\phi(N) = 1$ we know that $d_1 = d \mod k$.
2. By assumption $k > \epsilon 2^t$. Note that at this point we know $dmodk$ as well as the $nt$ most significant bits of $d$. We determine the rest of the bits by an exhaustive search. More precisely, write

   $$d = kd_2 + d_1$$

   Then $d_2 = d_0/k + (d - d_0)/k - d_1/k$. The only unknown term in this sum is $v = (d - d_0)/k$. Since $k > \epsilon 2^t$ we know that $v = (d - d_0)/k < 1/\epsilon$. To find $v$ we try all possible candidates in the range $0 \cdots \lceil \frac{1}{\epsilon} \rceil$. For each candidate we compute the candidate value of d and test it out.

Once the correct $k$ and $v$ are found, $d$ is exposed. □

## References


[1] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. On the power of simple branch prediction analysis. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 312–320. ACM, 2007.

[2] Divesh Aggarwal and Ueli Maurer. *Breaking RSA Generically Is Equivalent to Factoring*. 2009.

[3] K Berlin and SS Dhenakaran. An overview of cryptanalysis of rsa public key system.

[4] Daniel J Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko Van Someren. Factoring rsa keys from certified smart cards: Coppersmith in the wild. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 341–360. Springer, 2013.

[5] Johannes Blömer and Alexander May. New partial key exposure attacks on rsa. In *Annual International Cryptology Conference*, pages 27–43. Springer, 2003.

[6] Dan Boneh and Ramarathnam Venkatesan. Breaking rsa may not be equivalent to factoring (extended abstract. *Lecture Notes in Computer Science*, 1403:59–71, 1998.

[7] Kefan Dong Ce Jin. Cryptography project report: Modular $e$-th root in rsa. 2018.

[8] Don Coppersmith. Finding a small root of a univariate modular equation. In *International Conference on Theory  Application of Cryptographic Techniques*, 1996.

[9] Matthias Ernst, Ellen Jochemsz, Alexander May, and Benne De Weger. Partial key exposure attacks on rsa up to full size exponents. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 371–386. Springer, 2005.

[10] Nadia Heninger and Hovav Shacham. Reconstructing rsa private keys from random key bits. In *Advances in Cryptology-CRYPTO 2009*, pages 1–17. Springer, 2009.

[11] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. 1986.

[12] Ueli Maurer. Abstract models of computation in cryptography. In *International Conference on Cryptography  Coding*, 2005.

[13] Andrea Pellegrini, Valeria Bertacco, and Todd Austin. Fault-based attack of rsa authentication. In *Proceedings of the conference on Design, automation and test in Europe*, pages 855–860. European Design and Automation Association, 2010.