Number Theory and RSA Problem

Yu Zhang

HIT/CST/NIS

Cryptography, Spring, 2014

Outline

1. Arithmetic and Basic Group Theory
2. Primes and Factoring
3. RSA Assumption

Content

Primes and Divisibility

- The set of **integers** $\mathbb{Z}$, $a, b, c \in \mathbb{Z}$.
- a **divides** b: $a \mid b$ if $\exists c, ac = b$ (otherwise $a \nmid b$).
  b is a **multiple** of a. If $a \notin \{1, b\}$, then a is a **factor** of b.
- $p > 1$ is **prime** if it has no factors.
- An integer > 1 which is not prime is **composite**.
- $\forall a, b$, $\exists$ **quotient** q, **remainder** r: $a = qb + r$, and $0 \leq r < b$.
- **Greatest common divisor** $\gcd(a, b)$ is the largest integer c such that $c \mid a$ and $c \mid b$. $\gcd(0, b) = b$, $\gcd(0, 0)$ undefined.
- a and b are **relatively prime (coprime)** if $\gcd(a, b) = 1$.
- **Euclid's theorem**: there are infinitely many prime numbers.

Fundamental Theorem of Arithmetic

- **Bézout's lemma**: $\forall a, b,\ \exists\ X, Y:\ Xa + Yb = \gcd(a, b)$.
  $\gcd(a, b)$ is the smallest positive integer that can be expressed in this way.
- **Euclid's lemma**: If $c \mid ab$ and $\gcd(a, c) = 1$, then $c \mid b$.
  If p is prime and $p \mid ab$, then either $p \mid a$ or $p \mid b$.
- **Fundamental theorem of arithmetic**: $\forall N > 1,\ N = \prod_i p_i^{e_i}$,
  $\{p_i\}$ are distinct primes and $e_i \geq 1$. This expression is unique.

Modular Arithmetic

- Remainder $r = [a \bmod N] = a - b\lfloor a/b \rfloor$ and $r < N$. N is called **modulus**.
- **Reduction modulo** N: mapping a to $[a \bmod N]$.
- $\mathbb{Z}_N = \{0, 1, \ldots, N-1\} = \{a \bmod N | a \in \mathbb{Z}\}$.
- a and b are **congruent modulo** N: $a \equiv b \pmod N$ if $[a \bmod N] = [b \bmod N]$.
- a is **invertible modulo** $N \iff \gcd(a, N) = 1$. If $ab \equiv 1 \pmod N$, then $b = a^{-1}$ is **multiple inverse** of a **modulo** N.
- **Cancellation law**: If $\gcd(a, N) = 1$ and $ab \equiv ac \pmod N$, then $b \equiv c \pmod N$.
- **Euclidean algorithm**: $\gcd(a, b) = \gcd(b, [a \bmod b])$.
- **Extended Euclidean algorithm**: Given a, N, find X, Y with $Xa + YN = \gcd(a, N)$.

Examples of Modular Arithmetic

"Reduce and then add/multiply" instead of "add/multiply and then reduce".

Compute $193028 \cdot 190301 \bmod 100$

$193028 \cdot 190301 = [193028 \bmod 100] \cdot [190301 \bmod 100] \bmod 100$
$= 28 \cdot 1 \equiv 28 \bmod 100.$

$ab \equiv cb \pmod{N}$ does *not necessarily* imply $a \equiv c \pmod{N}$.

$a = 3, c = 15, b = 2, N = 24$

$3 \cdot 2 = 6 \equiv 15 \cdot 2 \pmod{24}$, but $3 \not\equiv 15 \pmod{24}$.

Use extended Euclidean algorithm to ...

Find the inverse of $11 \pmod{17}$

$(-3) \cdot 11 + 2 \cdot 17 = 1$, so 14 is the inverse of 11.

Groups

A **group** is a set $\mathbb{G}$ with a binary operation $\circ$:

- (**Closure**:) $\forall g, h \in \mathbb{G}$, $g \circ h \in \mathbb{G}$.
- (**Existence of an Identity**:) $\exists$ **identity** $e \in \mathbb{G}$ such that $\forall g \in \mathbb{G}, e \circ g = g = g \circ e$.
- (**Existence of Inverses**:) $\forall g \in G$, $\exists\ h \in \mathbb{G}$ such that $g \circ h = e = h \circ g$. h is an **inverse** of g.
- (**Associativity**:) $\forall g_1, g_2, g_3 \in \mathbb{G}$, $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$.

$\mathbb{G}$ with $\circ$ is **abelian** if

- (**Commutativity**:) $\forall g, h \in \mathbb{G}, g \circ h = h \circ g$.

Existence of inverses implies **cancellation law**.
When $\mathbb{G}$ is a **finite group** and $|\mathbb{G}|$ is the **order** of group.

Group Exponentiation

$$g^m \stackrel{\text{def}}{=} \underbrace{g \circ g \circ \cdots \circ g}_{m \text{ times}}.$$

Theorem 1

$\mathbb{G}$ is a finite group. Then $\forall g \in \mathbb{G}, g^{|\mathbb{G}|} = 1$.

Corollary 2

$\forall g \in \mathbb{G}$ and i, $g^i = g^{[i \bmod |\mathbb{G}|]}$.

Corollary 3

Define function $f_e : \mathbb{G} \to \mathbb{G}$ by $f_e(g) = g^e$.
If $\gcd(e, |\mathbb{G}|) = 1$, then f_e is a permutation.
Let $d = [e^{-1} \bmod |\mathbb{G}|]$, then f_d is the inverse of f_e. ($f_d(f_e(g)) = g$)
*e**'th root of** c: $g^e = c$, $g = c^{1/e} = c^d$.*

The Group $\mathbb{Z}_N^*$

$$\mathbb{Z}_N^* \overset{\text{def}}{=} \{a \in \{1, \ldots, N-1\} \mid \gcd(a, N) = 1\}$$

Euler's phi function: $\phi(N) \overset{\text{def}}{=} |\mathbb{Z}_N^*|$.

Theorem 4

$N = \prod_i p_i^{e_i}$, $\{p_i\}$ are distinct primes, $\phi(N) = \prod_i p_i^{e_i - 1}(p_i - 1)$.

Corollary 5 (Euler's theorem & Fermat's little theorem)

$a \in \mathbb{Z}_N^*$. $a^{\phi(N)} \equiv 1 \pmod{N}$.
If p is prime and $a \in \{1, \ldots, p-1\}$, then $a^{p-1} \equiv 1 \pmod{p}$.

Corollary 6

Define function $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ by $f_e(x) = [x^e \bmod N]$.
If $\gcd(e, \phi(N)) = 1$, then f_e is a permutation.
Let $d = [e^{-1} \bmod \phi(N)]$, then f_d is the inverse of f_e.
*e**'th root of** c: $g^e = c$, $g = c^{1/e} = c^d$.*

Examples on Groups

- $\mathbb{Z}$ is an abelian group under '$+$', not a group under '$\cdot$'.
- The set of real numbers $\mathbb{R}$ is not a group under '$\cdot$'.
- $\mathbb{R} \setminus \{0\}$ is an abelian group under '$\cdot$'.
- $\mathbb{Z}_N$ is an abelian group under '$+$' modulo N.
- If p is prime, then $\mathbb{Z}_p^*$ is an abelian group under '$\cdot$' modulo p.
- $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$, $|\mathbb{Z}_{15}^*| = 8$.
- $\mathbb{Z}_3^*$ is a subgroup of $\mathbb{Z}_{15}^*$, but $\mathbb{Z}_5^*$ is not.
- $2^{1/3} \bmod 5 = 2^3 \bmod 5 = 3$. $(3^{-1} = 3 \pmod 4)$
- g^3 is a permutation on $\mathbb{Z}_{15}^*$, but g^2 is not (e.g., $8^2 \equiv 2^2 \equiv 4$).

$N = pq$ where p, q are distinct primes. $\phi(N) = ?$

$\phi(N) = (N-1) - (q-1) - (p-1) = (p-1)(q-1)$.

Arithmetic algorithms

- **Addition/subtraction**: linear time $O(n)$.
- **Mulplication**: naively $O(n^2)$. Karatsuba (1960): $O(n^{\log_2 3})$
  Basic idea: $(2^b x_1 + x_0) \times (2^b y_1 + y_0)$ with 3 mults.
  Best (asymptotic) algorithm: about $O(n \log n)$.
- **Division with remainder**: $O(n^2)$.
- **Exponentiation**: $O(n^3)$.

Algorithm 1: Exponentiating by Squaring

input : $g \in G$; exponent $x = [x_n x_{n-1} \ldots x_2 x_1 x_0]_2$

output: g^x

```
y ← g; z ← 1
for i = 0 to n do
    if x_i == 1 then z ← z × y
    y ← y^2
return z
```

Content

Integer Factorization/Factoring

"The problem of distinguishing prime numbers from composite numbers and of resolving the later into their prime factors is known to be one of the most important and useful in arithmetic." – Gauss (1805)

The "hardest" numbers to factor seem to be those having only large prime factors.

- The best-known algorithm is the **general number field sieve** [Pollard] with time $\mathcal{O}(\exp(n^{1/3} \cdot (\log n)^{2/3}))$.
- RSA Factoring Challenge: RSA-768 (232 digits)
  - Two years on hundreds of machines (2.2GHz/2GB, 1500 years)
  - Factoring a 1024-bit integer: about 1000 times harder.

Generating Random Primes

Algorithm 2: Generating a random prime

input : Length n; parameter t

output: A random n-bit prime

for $i = 1$ **to** t **do**
    $p' \leftarrow \{0,1\}^{n-1}$
    $p := 1 \| p'$
    if p *is prime* **then return** p
return fail

To show its efficiency, we need understand two issues:

- the probability that a randomly-selected n-bit integer is prime.
- how to efficiently test whether a given integer p is prime.

The Distribution of Prime

Theorem 7 (Prime number theorem)

$\exists$ *a constant* c *such that,* $\forall n > 1$*, a randomly selected* n*-bit number is prime with probability at least* c/n*.*

The probability that a prime is *not* chosen in $t = n^2/c$ iterations is

$$\left(1 - \frac{c}{n}\right)^t = \left(\left(1 - \frac{c}{n}\right)^{n/c}\right)^n \leq \left(e^{-1}\right)^n = e^{-n}.$$

The algorithm will fail with a negligible probability.

Testing Primality

- **Trial division**: Divide N by $a = 2, 3, \ldots, \sqrt{N}$.
- **Probabilistic algorithm for approximately computing**:
  - Atlantic City algorithm with two-sided error.
  - Monte Carlo algorithm with one-sided error.
  - Las Vegas algorithm with zero-sided error.
- **Fermat primality test**: $a^{N-1} \equiv 1 \pmod{N}$.
- a is a **witness** that N is composite if $a^{N-1} \not\equiv 1 \pmod{N}$.
- a is a **liar** if N is composite and $a^{N-1} \equiv 1 \pmod{N}$.
- **Carmichael numbers**: composite numbers without witnesses.

Theorem 8

If $\exists$ a witness, then at least half the elements of $\mathbb{Z}_N^*$ are witnesses.

Examples of Primality Tests

Liars in Fermat primality test

$2^{340} \equiv 1 \pmod{341}$, but $341 = 11 \cdot 31$.
$5^{560} \equiv 1 \pmod{561}$, but $561 = 3 \cdot 11 \cdot 17$.
Carmichael numbers < 10000:
561, 1105, 1729, 2465, 2821, 6601, 8911.

The Factoring Assumption

Let $\mathsf{GenModulus}(1^n)$ be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$, and p, q are n-bit primes except with probability negligible in n.

The factoring experiment $\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n)$:

1. Run $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q).
2. $\mathcal{A}$ is given N, and outputs $p', q' > 1$.
3. $\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1$ if $p' \cdot q' = N$, and 0 otherwise.

Definition 9

Factoring is hard relative to GenModulus if $\forall$ PPT algorithms $\mathcal{A}$, $\exists$ negl such that

$$\Pr[\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] \leq \mathsf{negl}(n).$$

Algorithms for Factoring

- **Factoring** $N = pq$. p, q are of the same length n.
- **Trial division**: $\mathcal{O}(\sqrt{N} \cdot \text{polylog}(N))$.
- **Pollard's** $p-1$ method: effective when $p-1$ has "small" prime factors.
- **Pollard's rho** method: $\mathcal{O}(N^{1/4} \cdot \text{polylog}(N))$.
- **Quadratic sieve** algorithm [Carl Pomerance]: sub-exponential time $\mathcal{O}(\exp(\sqrt{n \cdot \log n}))$.
- The best-known algorithm is the **general number field sieve** [Pollard] with time $\mathcal{O}(\exp(n^{1/3} \cdot (\log n)^{2/3}))$.

Content

The RSA Problem

Recall group exponentiation on $\mathbb{Z}_N^*$

Define function $f_e\ : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ by $f_e(x) = [x^e \bmod N]$.
If $\gcd(e, \phi(N)) = 1$, then f_e is a permutation.
If $d = [e^{-1} \bmod \phi(N)]$, then f_d is the inverse of f_e.
e**'th root of** c: $g^e = c$, $g = c^{1/e} = c^d$.

Idea: factoring is hard

$\implies$ for $N = pq$, finding p, q is hard

$\implies$ computing $\phi(N) = (p-1)(q-1)$ is hard

$\implies$ computations modulo $\phi(N)$ is not available

There is a gap.

$\implies$ **RSA problem** [Rivest, Shamir, and Adleman] is hard:
Given $y \in \mathbb{Z}_N^*$, compute y^{-e}, e^{th}-root of y modulo N.

Open problem

RSA problem is easier than factoring?

Generating RSA Problem

Algorithm 3: GenRSA

input : Security parameter 1^n

output: N, e, d

$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$

$\phi(N) := (p-1)(q-1)$

find e such that $\gcd(e, \phi(N)) = 1$

compute $d := [e^{-1} \bmod \phi(N)]$

return N, e, d

The RSA Assumption

The RSA experiment $\mathsf{RSAinv}_{\mathcal{A},\mathsf{GenRSA}}(n)$:

1. Run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d).
2. Choose $y \leftarrow \mathbb{Z}_N^*$.
3. $\mathcal{A}$ is given N, e, y, and outputs $x \in \mathbb{Z}_N^*$.
4. $\mathsf{RSAinv}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1$ if $x^e \equiv y \pmod N$, and 0 otherwise.

Definition 10

RSA problem is hard relative to GenRSA if $\forall$ PPT algorithms $\mathcal{A}$, $\exists$ negl such that

$$\Pr[\mathsf{RSAinv}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] \leq \mathsf{negl}(n).$$

Summary

- Primes, modular arithmetic.
- e^{th}-root modulo N, RSA.

Textbook

"*A Computational Introduction to Number Theory and Algebra*" (Version 2) by Victor Shoup