Factoring

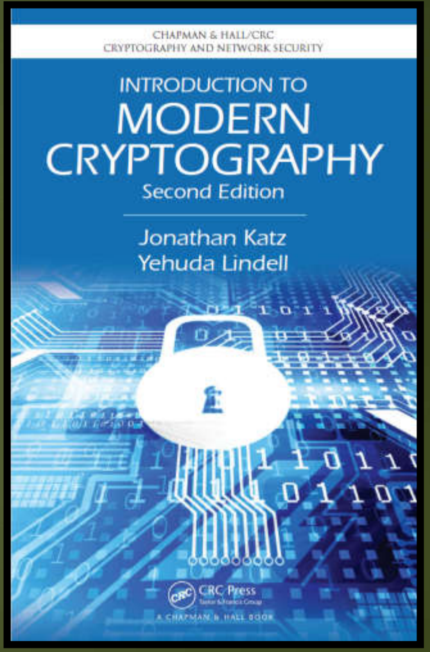
ALGORITHMS AND HARDNESS ASSUMPTIONS

Xinyu Mao

11/12/2021

Outline

- Algorithms for factoring ($N = pq$)
  - Pollard's $p-1$ algorithm: it is effective if $p-1$ has only small prime factors.
  - Pollard's rho algorithm: runs in $\tilde{O}\left(N^{\frac{1}{4}}\right)$.
  - Quadratic sieve algorithm(§ 15.3): runs in sub-exponential time, i.e., $2^{o(\log N)}$. [Based on *A Tale of Two Sieves* by Carl Pomerance]
  - Connection between factoring assumption and RSA Assumption
- The power of quantum computing [A famous book: *Quantum Computation and Quantum Information*]



Simple Algorithms for Factoring

Preparation and Warm-up

- Assume that $N = pq$, where p, q are distinct primes.
  - Usually, p and q each has the same length $n = \Theta(\log N)$.
- $x \overset{\$}{\leftarrow} S$ means *choose x uniformly at random from the set S.*
- $\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ by CRT, and we write $x \overset{\text{CRT}}{\longleftrightarrow} (x_p, x_q)$ with $x_p \coloneqq x \,\%\, p, x_q \coloneqq x \,\%\, q$, to denote the isomorphism.

The Main Idea behind *Pollard's $p-1$ Algorithm*

- We want to find $B \in \mathbb{N}$ such that $|\mathbb{Z}_p^*| = p-1$ divides B. Then for $x \xleftarrow{\$} \mathbb{Z}_N^*$, we have
$$(x^B - 1) \overset{\mathrm{CRT}}{\longleftrightarrow} (x_p, x_q)^B - (1,1) = (x_p^B - 1, x_q^B - 1) = (0, x_q^B - 1).$$
  - $y := (x^B - 1)\%N$ satisfies $p \mid y,\ q \nmid y$(with high probability).
  - Then $\gcd(y, N) = p$.
- How to find B?
  - If $p-1$ has no large factor, we set
$$B := \prod_{i=1}^{k} p_i^{\left\lfloor \frac{n}{\lfloor \log p_i \rfloor} \right\rfloor}.$$
  - k has to be small for efficiency.
  - If q has a factor greater that p_k, everything works.

Pollard's $p - 1$ Algorithm

Pollard's $p - 1$ Algorithm

Input: integer N
Output: A non-trivial factor of N

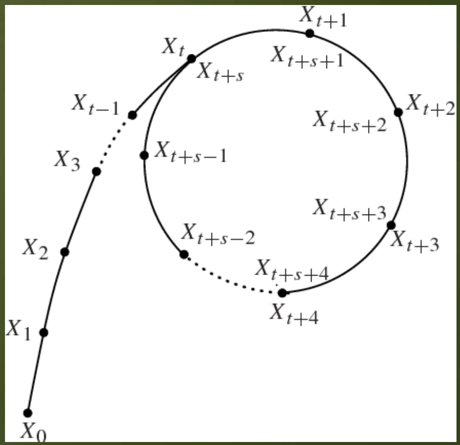
1. $x \xleftarrow{\$} \mathbb{Z}_N^*$
2. $y := (x^B - 1)\% N$
3. $p := \gcd(y, N)$
4. If $p \notin \{1, N\}$ return p

$$B := \prod_{i=1}^{k} p_i^{\left\lfloor \frac{n}{\log p_i} \right\rfloor}$$

The Main Idea behind Pollard's rho algorithm

- If we have $x, x' \in \mathbb{Z}_N^*$ with $x \equiv x' \ (mod\ p)$, then $x'' := x - x'$ is a multiple of p.
- **Find a collision.** Randomly sample $x^{(1)}, x^{(2)}, \dots, x^{(k)}$. In the argument of birthday paradox, we know that if $k := O(\sqrt{p}) = O\left(N^{\frac{1}{4}}\right)$, collision occurs with high probability.
  - Check all pairs $(i, j) \in [k]^2$? This is as bad as trial division!.
  - Recall how we find a collision of a hash function.

Theorem. *Let $x_1, x_2, \dots, x_q$ be a sequence of values with $x_{k+1} = H(x_k)$. If $x_i = x_j$ for some $(i, j) \in [q]^2$, then $\exists\ k \in [j-1]$ s.t. $x_k = x_{2k}$.*



Pollard's rho algorithm

Pollard's rho Algorithm

Input: N *(product of two* n*-bit primes)*
Output: A non-trivial factor of N

1. $x \xleftarrow{\$} \mathbb{Z}_N^*, x' := x$
2. For $i = 1$ to $2^{n/2}$
   - $x := H(x)$
   - $x' := H(H(x'))$
   - $p := \gcd(x - x', N)$
   - If $p \notin \{1, N\}$ return p

- The 'hash function' H must satisfy
  $$x \equiv x' \ (mod\ p) \Rightarrow H(x) = H(x').$$
- A standard choice is $H(x) = (x^2 + 1)\%N$.
  - In fact, any polynomial will do.
- Each iteration takes only $O(\text{polylog}(n))$ time, and thus the total running time is $\tilde{O}\left(2^{\frac{n}{2}}\right) = \tilde{O}\left(N^{\frac{1}{4}}\right)$.
- It is space efficient.
- We can also use the same technique to solve discrete logarithm(see § 11.2.5).

Quadratic Sieve Algorithm

A SUB-EXPONENTIAL ALGORITHM FOR FACTORING

The First Idea

- Given $x, y \in \mathbb{Z}^+$ with $x^2 \equiv y^2 \ (mod\ N)$ and, we can factor N as follows.
  - $x^2 - y^2 = (x + y)(x - y) \equiv 0 \ (mod\ N)$. That is, $N|(x + y)(x - y)$.
  - If $x \not\equiv \pm y \ (mod\ N)$, we have $N \nmid (x + y)$ and $N \nmid (x - y)$. Then $\gcd(x - y, N)$ is a non-trivial factor of N.
- How to find such x, y ?
  - A naive way: $x \xleftarrow{\$} \mathbb{Z}_N^*$, check if $q = x^2 \ \%\ N$ is a square. If $q = y^2$ for some $y \in \mathbb{Z}^+$, we have $x^2 \equiv y^2 \ (mod\ N)$.
    - e.g. $N = 35,\ x = 12, q = 12^2 \ \%\ 35 = 4 \ \rightarrow y = 2$.
  - Improvement: $x_1, x_2 \dots, x_\ell \xleftarrow{\$} \mathbb{Z}_N^*$ and set $q_i := x_i^2 \ \%\ N$. Next, try to find a subset $S \subseteq [\ell]$ such that $\prod_{i \in S} q_i$ is a square.

Smooth Numbers

- How to check whether $\prod_{i \in S} q_i$ is a square efficiently?
  - How nice would It be if we can figure out the factoring of each q_i !
- ***Definition.*** *Let $y \in \mathbb{R}^+$ and $m \in \mathbb{Z}^+$. We say that m is y-smooth if all prime divisors of m are at most y.*
- *Fix some bound B, if we make sure every q_i is B-smooth, we can factor q_i easily.*
  - Can we efficiently sample B-smooth numbers? This depends on the density

$$\rho(X, B) := \frac{1}{X} |\{i \in \mathbb{Z}^+ : i < X \text{ and } i \text{ is } B \text{ smooth}\}|.$$

A General Plan for factoring

A General Plan for factoring
Input: N, B, ℓ
Output: A non-trivial factor of N

1. $\delta \coloneqq 0$
2. For $i = 1$ to ℓ
   - Find $x_i \in \mathbb{Z}_N^*$ such that $q_i \coloneqq x_i^2 \; \% \; N$ is B-smooth
   - $(e_{i1}, \dots, e_{ik}) \leftarrow \text{Factor}(q_i)$
3. Find a subset $S \subseteq [\ell]$ such that $\sum_{i \in S} \boldsymbol{e_i} \in 2\mathbb{Z}^k$
4. $x \coloneqq \prod_{i \in S} x_i$
5. $\beta_i \coloneqq \sum_{j \in S} e_{j,i}, \quad y \coloneqq \prod_{i \in [k]} p_i^{\beta_i/2}$
6. $p \coloneqq \gcd(x - y, N)$
7. If $p \notin \{1, N\}$ return p

- Let $\{p_1, \dots, p_k\}$ be the primes $\leq B$.
- $q_i \coloneqq x^2 \; \% \; N = \prod_{j \in [k]} p_j^{e_{ij}}$.
- To ensure step 2 can succeed by Gaussian elimination, we must set $\ell \geq k + 1$.
- Next we need to:
  - Specify how to find x_i.
  - Choosing the optimal B.

Kraitchik's method

How to find x such that $q \coloneqq x^2 \% N$ is B-smooth?
Intuition: small numbers are more likely to be B-smooth.

- Write $s \coloneqq \lfloor\sqrt{N}\rfloor$. Consider the sequence $x_i \coloneqq s + i,\ i \in [D]$.
- Let $q_i \coloneqq x_i^2 \% N$. Note that q_i's are not at all random, since $q_i \in \mathcal{QR}_N$.
- Heuristically, the number of good q_i's is roughly equal to $\rho(X, B) \cdot D$.
- Note that we require $\ell \geq k + 1 = \pi(B) + 1$, and hence we shall set $D \approx \frac{\pi(B)}{\rho(X,B)}$.
- If we factor each q_i by trial division, the running time is bounded by $O\big(D \cdot \pi(B)\big) = O\left(\frac{\pi(B)^2}{\rho(X,B)}\right)$.
- When $B \approx \exp(\frac{1}{2}\sqrt{\log X \log\log X})$, and the minimum value of $\frac{\pi(B)^2}{\rho(X,B)}$ is $\exp(2\sqrt{\log X \log\log X})$.
- X is about $n^{\frac{1}{2}+o(1)}$ here, and the running time is bounded by $\exp\left(\sqrt{2 \log N \log\log N}\right)$.

The Quadratic Sieve: a practical improvement

We study the polynomial $F(X) \coloneqq (X+s)^2 - N$.
How to identify B-smooth numbers in the sequence $F(1), F(2), \dots, F(D)$ more efficiently?

Set $v[j] \coloneqq F(j)$ for all $j \in [D]$
// do the following for each p
For $i = 1,2,\dots,d$:

1. $j \coloneqq r_i$
2. While $j \le D$ do
   1. While $p | v[j]$ do
      $v[j] \coloneqq v[j]/p$
   2. $j \coloneqq j + p$

- The idea of sieve: e.g. Sieve of Eratosthenes
- Assume that F has d distinct roots modulo p lying in the interval $[1, p]$, call them $r_1, \dots, r_d$.
- For each $j \in [D]$, if $p|F(j)$, divide $v[j]$ by p if possible. At the end, $F(j)$ is B-smooth iff $v[j] = 1$.
- Note that $p|F(j) \Leftrightarrow j$ is a root of $F(X)$ modulo $p \Leftrightarrow j \equiv r_k \pmod p$ for some $k \in [d]$.
- $r_1, \dots, r_d$ can be calculated with an algorithm for computing modular square roots. (§ 12.5)
- Running time is improved to $\exp\left(\sqrt{\log N \log\log N}\right)$.

Relating Factoring Assumption to RSA Assumption

Factoring Assumption

GenModulus
Input: 1^n
Output: (N, p, q) *where* $N = pq$ *and* p, q *are* n*-bit primes.*

$\mathcal{C}$		$\mathcal{A}$
$(N, p, q) \leftarrow \mathrm{GenModulus}(1^n)$	N $\longrightarrow$	Compute
	p', q' $\longleftarrow$	p', q'
If $p'q' = N$ output 1; Otherwise output 0.		

The experiment $\mathrm{Factor}_{\mathcal{A}}^{\mathrm{GenModulus}}(n)$

- ***Definition.*** *Factoring is hard relative to* GenModulus *if for all PPT* $\mathcal{A}$*,*
$$\Pr\left[\mathrm{Factor}_{\mathcal{A}}^{\mathrm{GenModulus}}(n) = 1\right] \leq negl(n).$$
- **Factoring assumption**: there exists a GenModulus relative to which factoring is hard.

RSA Assumption

GenRSA
Input: 1^n
Output: (N, e, d) *such that*
- $N = pq$ *and* p, q *are* $n-$*bit primes.*
- $e > 1$ and $ed \equiv 1 \left(mod\ \varphi(N)\right)$.

$\mathcal{C}$		$\mathcal{A}$
$(N, e, d) \leftarrow \mathrm{GenRSA}(1^n)$ $y \overset{\$}{\leftarrow} \mathbb{Z}_N^*$	$\xrightarrow{(N, e, y)}$	Compute x
	$\xleftarrow{x}$	
If $x^e = y$ output 1; Otherwise output 0.		

The experiment $\mathrm{RSA}_{\mathcal{A}}^{\mathrm{GenRSA}}(n)$

- ***Definition.*** *RSA problem is hard relative to* GenRSA *if for all PPT* $\mathcal{A}$,
$$\Pr\left[\mathrm{RSA}_{\mathcal{A}}^{\mathrm{GenRSA}}(n) = 1\right] \le negl(n).$$
- **RSA assumption**: there exists a GenRSA relative to which RSA problem is hard.

Relations between RSA Assumption and Factoring Assumption

- RSA assumption holds → Factoring assumption holds
  - The other direction is still open.
- ***Theorem.*** *Assuming that Factoring Assumption holds, then for all PPT algorithm $\mathcal{A}$,*

$$\Pr[\mathcal{A}(1^n, N, e) = d] = negl(n).$$

  - Note that $\varphi(N) | (ed - 1)$.
  - (§ 10.4) Given any nonzero multiple of $|\mathbb{Z}_N^*|$, one can efficiently factor N.

The Power of Quantum Computing

Shor's Algorithm: Reducing Factoring to Order Finding

FindOrder
Input: $N, x \in \mathbb{Z}_N^*$
Output: $\mathrm{ord}(x)$ *in* $\mathbb{Z}_N^*$

- If we can find a non-trivial square root of 1, then we can factor N easily.
- FindOrder can be implemented efficiently by quantum computing devices.

Theorem. *Let* $N = \prod_{i=1}^{m} p_i^{e_i}$ be the factorization of a odd composite number N. Then

$$\Pr_{x \xleftarrow{\$} \mathbb{Z}_N^*}\left[\mathrm{ord}(x) \text{ is even} \wedge x^{\frac{ord(x)}{2}} \not\equiv -1 \ (mod\ N)\right] \geq 1 - \frac{1}{2^m}$$

Thanks for listening.☺