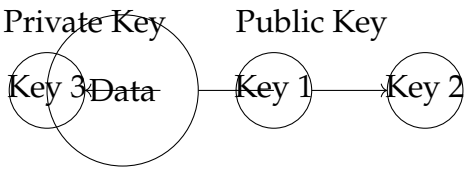
Public Key Cryptography

Ji Yong-Hyeon



Ji Yong-Hyeon
June 3, 2023

Contents

Chapter 1

Public Key Cryptography in a Nutshell: Classification and Security Notions

1.1 PKE

Signature, Key Establishment
    Onewayness Trapdoor, IND-CCA2, EUF-CMA

Chapter 2

IFP-based Primitives

2.1 Textbook RSA

RSA is a public-key cryptosystem that uses the mathematics of prime numbers to secure communication over the internet. It is widely used in various applications, such as digital signatures, secure email, and online banking.

The RSA cryptosystem is based on the following mathematical concepts:

- Modular arithmetic: - a mod b (remainder when a is divided by b) - Euler's totient function phi(n) (number of positive integers less than n that are coprime to n)
- Prime factorization: - finding the unique prime factors of a given integer
- The Chinese Remainder Theorem: - a theorem that provides a solution to a system of linear congruences with pairwise relatively prime moduli
- Fermat's Little Theorem: - a theorem that states that if p is a prime number and a is an integer not divisible by p, then $a^{(p-1)}$ is congruent to 1 modulo p.
- Euler's Theorem: - a generalization of Fermat's Little Theorem that states that if a and n are coprime, then $a^{(\phi(n))}$ is congruent to 1 modulo n.

2.1.1 The RSA Algorithm

Key Generation

To generate an RSA key pair, we follow these steps:

1. Choose two large prime numbers p and q.
2. Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
3. Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
4. Compute d such that $de \equiv 1 \pmod{\phi(n)}$.

The public key is (n, e), and the private key is d.

Encryption

To encrypt a message M, we use the public key (n, e) and compute:

$$C = M^e \pmod{n}$$

The ciphertext C is then sent to the recipient.

Decryption

To decrypt the ciphertext C, we use the private key d and compute:

$$M = C^d \pmod{n}$$

The plaintext M is then recovered.

2.1.2 Security

RSA is secure because it is based on the difficulty of factoring large integers. If an attacker can factor n into its prime factors p and q, then they can compute $\phi(n)$ and derive the private key. However, factoring large numbers is currently considered computationally infeasible, making RSA a secure choice for many applications.

2.1.3 Conclusion

RSA is a widely used public-key cryptosystem that uses prime numbers to secure communication over the internet. It is based on the difficulty of factoring large integers and is currently considered secure for many applications.

2.2 RSA-CRT

RSA-CRT is a modified version of the RSA public-key cryptosystem that leverages the Chinese Remainder Theorem to enhance the speed of the decryption process.

RSA-CRT works by first generating two large primes, p and q, and computing the modulus $n = pq$. Then, the public and private keys are generated using the usual RSA key generation algorithm.

When encrypting a message, the sender uses the recipient's public key to encrypt the message, as in standard RSA. However, when decrypting a message, the recipient first computes two intermediate values, m_1 and m_2, using the Chinese Remainder Theorem. These values are then combined to obtain the original message.

The Chinese Remainder Theorem states that given a system of linear congruences with pairwise relatively prime moduli, there exists a unique solution modulo the product of the moduli. In the case of RSA-CRT, the two moduli are p and q, which are both primes, and the system of congruences is:

$$m \equiv c^d \pmod{p} \; m \equiv c^d \pmod{q}$$

where c is the encrypted message, d is the recipient's private key, and m is the decrypted message.

The two intermediate values, m_1 and m_2, are computed as:

$$m_1 \equiv c^d \pmod{p} \; m_2 \equiv c^d \pmod{q}$$

These values can be computed efficiently using modular exponentiation, which is much faster than using the standard RSA decryption algorithm to compute m directly.

Finally, the original message can be recovered by combining m_1 and m_2 using the Chinese Remainder Theorem:

$$m \equiv m_1 + q((m_2 - m_1)q^{-1} \bmod p) \pmod{n}$$

where q^{-1} is the modular inverse of q modulo p. This formula allows the recipient to efficiently compute the original message without having to compute the expensive modular exponentiation step for the entire modulus n.

RSA-CRT Algorithm

1. Generate two large primes p and q, and compute $n = pq$.
2. Generate the public and private keys using the usual RSA key generation algorithm. That is,
   (a) Compute the totient of n, $\phi(n) = (p-1)(q-1)$.
   (b) Choose an integer e such that
   $$1 < e < \phi(n) \quad \text{and} \quad \gcd(e, \phi(n)) = 1.$$
   This is the public key exponent.
   (c) Compute the private key exponent d such that
   $$d \equiv e^{-1} \pmod{\phi(n)}.$$
   This can be done efficiently using the Extended Euclidean Algorithm.

   The public key is (n, e), and the private key is (n, d).
3. To encrypt a message $\mathcal{M}$, use the recipient's public key to compute
   $$C = \mathcal{M}^e \bmod n.$$
4. To decrypt a message C, compute the two intermediate values:
   (a) $m_1 = C^d \bmod p$.
   (b) $m_2 = C^d \bmod q$.
5. Combine the two intermediate values using the Chinese Remainder Theorem to obtain the original message $\mathcal{M}$:
   $$\mathcal{M} = m_1 + q\left((m_2 - m_1)\, q^{-1} \bmod p\right) \pmod{n}.$$

Chinese Remainder Theorem

Theorem 2.1. *Given a system of linear congruences with pairwise relatively prime moduli:*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_1}, \end{aligned}$$

there exists a unique solution x modulo $M = m_1 m_2 \cdots m_n$:

$$x \equiv a_1 b_1 M_1 + a_2 b_2 M_2 + \cdots a_n b_n M_n \pmod{M}, \text{i.e.,}$$
$$x \equiv \sum_{i=1}^{n} a_i b_i M_i \pmod{M}$$

where $M_i = M/m_i$ and b_i is the inverse of M_i modulo m_i.

2.3 Fermat's Little Theorem

Fermat's Little Theorem

Theorem 2.2. *Let p is a prime number and a is an integer not divisible by p, then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

In other words, if we take any non-zero integer a and raise it to the power $p-1$, then divide the result by a prime p, the remainder will always be 1. This theorem is widely used in number theory and cryptography.

Proof. We use mathematical induction and binomial theorem.

1. (Basic Step) Let $a = 1$, then $1^{p-1} = 1 \equiv \pmod{p}$, which is true.
2. (Inductive Step) Suppose that the theorem holds for some integer a. We need to show that it also holds for $a+1$. We can express $(a+1)^p$ as:

$$(a+1)^p = \sum_{i=0}^{p} \binom{p}{i} a^i 1^{p-i}$$

□

Using the binomial theorem. Since p is a prime number, we know that $\binom{p}{i}$ is divisible by p for $0 < i < p$, hence:

$$(a+1)^p \equiv a^p + 1^p \equiv a + 1 \pmod{p}$$

The last step follows from the fact that $a^{p-1} \equiv 1 \pmod{p}$. Therefore, we have shown that:

$$(a+1)^{p-1} \equiv 1 \pmod{p}$$

which completes the proof of Fermat's Little Theorem.

Proof. We use the group theory of the multiplicative group of integers modulo a prime p.

Let p be a prime number, and let a be an integer that is not divisible by p. Consider the set of integers modulo p, denoted by $\mathbb{Z}/p\mathbb{Z}$. This set forms a group under multiplication, denoted by $(\mathbb{Z}/p\mathbb{Z})^\times$. Since p is prime, the group $(\mathbb{Z}/p\mathbb{Z})^\times$ has order $p-1$, which means that it contains $p-1$ distinct elements.

Now consider the subset of $(\mathbb{Z}/p\mathbb{Z})^\times$ consisting of the multiples of a, denoted by $S_a = a, 2a, 3a, \ldots, (p-1)a$. Since a is not divisible by p, the set S_a consists of distinct elements. We claim that S_a is a permutation of the elements of $(\mathbb{Z}/p\mathbb{Z})^\times$.

To prove this claim, we need to show that every element of $(\mathbb{Z}/p\mathbb{Z})^\times$ can be expressed as a multiple of a modulo p. Suppose for the sake of contradiction that there exists an element $b \in (\mathbb{Z}/p\mathbb{Z})^\times$ that cannot be expressed as $b = ka \pmod{p}$ for any integer k. Then the integers $1a, 2a, \ldots, (p-1)a$ would not be distinct modulo p, which contradicts our assumption that S_a consists of distinct elements. Therefore, S_a is a permutation of the elements of $(\mathbb{Z}/p\mathbb{Z})^\times$.

Since S_a is a permutation of the elements of $(\mathbb{Z}/p\mathbb{Z})^\times$, we can multiply all the elements of S_a together to obtain:

$$a \cdot 2a \cdot \ldots \cdot (p-1)a \equiv 1 \cdot 2 \cdot \ldots \cdot (p-1) \pmod{p}$$

which simplifies to:

$$(p-1)!a^{p-1} \equiv (p-1)! \pmod{p}$$

Since p does not divide $(p-1)!$, we can cancel out $(p-1)!$ from both sides to obtain:

$$a^{p-1} \equiv 1 \pmod{p}$$

This completes the proof of Fermat's Little Theorem using the theory of abstract algebra. □

2.4 Euler's Theorem

Euler's Theorem is a generalization of Fermat's Little Theorem. It states that if a and n are two positive integers that are coprime, then:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

where $\varphi(n)$ is Euler's totient function, which gives the number of positive integers less than or equal to n that are coprime with n. Euler's Theorem is useful in cryptography for testing the primality of large numbers and for computing modular exponentiations efficiently.

Proof: We can prove Euler's Theorem using the fact that the totient function is multiplicative, which means that if m and n are two coprime positive integers, then:

$$\varphi(mn) = \varphi(m)\varphi(n)$$

We can use this property to reduce the theorem to the case where n is a prime power p^k, where p is a prime and k is a positive integer. If $k = 1$, then Euler's Theorem reduces to Fermat's Little Theorem. Otherwise, we can use the Chinese Remainder Theorem to combine the solutions of $a^{\varphi(p^k)} \equiv 1 \pmod{p^k}$ for all prime powers p^k that divide n. Since the exponents $\varphi(p^k)$ are powers of p, we can use repeated squaring to compute the modular exponentiations efficiently. This completes the proof of Euler's Theorem.

Proof. Euler's Theorem is a generalization of Fermat's Little Theorem and states that if a and n are two coprime positive integers, then

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

where $\varphi(n)$ is Euler's totient function, which gives the number of positive integers less than or equal to n that are coprime with n.

To prove Euler's Theorem, we first define a group G of integers modulo n that are coprime with n. This group is denoted as:

$$G = a \in \mathbb{Z}_n | gcd(a, n) = 1$$

Next, we define the function $f(a) = ra \pmod{n}$ where r is a fixed integer coprime with n. It can be shown that f is a permutation of G and that the order of the group G is $\varphi(n)$.

Now, we consider the product of all elements in G:

$$P = \prod_{a \in G} f(a) = \prod_{a \in G} ra \pmod{n}$$

We can rewrite this product as:

$$P = r^{\varphi(n)} \prod_{a \in G} a \pmod{n}$$

Since a is coprime with n, we know that a has a unique inverse b in the group G such that $ab \equiv 1 \pmod{n}$. Therefore, we can rewrite the product as:

$$P = r^{\varphi(n)} \prod_{a \in G} a \prod_{a \in G} b = r^{\varphi(n)} \prod_{a \in G} 1 = r^{\varphi(n)}$$

where we used the fact that $ab \equiv 1 \pmod{n}$ and that a and b are both in the group G.

On the other hand, we can rearrange the terms in the product as:

$$P = \prod_{a \in G} f(a) = \prod_{a \in G} (ar) \pmod{n} = r^{\varphi(n)} \prod_{a \in G} a \pmod{n}$$

where we used the fact that ar is also in the group G since r is coprime with n.

Equating the two expressions for P, we get:

$$r^{\varphi(n)} \prod_{a \in G} a \equiv r^{\varphi(n)} \prod_{a \in G} a \pmod{n}$$

Dividing both sides by $\prod_{a \in G} a$, we obtain:

$$r^{\varphi(n)} \equiv 1 \pmod{n}$$

which is Euler's Theorem. □

2.5 Primality Test

A primality test is a method used to determine if a given positive integer is a prime number or a composite number. There are various primality tests, and they differ in their speed, accuracy, and the range of numbers they can handle.

One of the simplest and most well-known primality tests is the trial division method. This method involves dividing the number by each integer from 2 up to the square root of the number, checking if any of the divisors evenly divide the number. If no divisor is found, then the number is prime. However, this method becomes impractical for very large numbers, as the number of potential divisors to check grows with the number being tested.

Another primality test is the Fermat primality test. This test is based on Fermat's Little Theorem, which states that if p is a prime number and a is any integer not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$. The Fermat primality test uses this theorem to check if a number is prime by randomly selecting values of a and checking if the equation holds for each value. If the equation fails for any a, then the number is composite. If the equation holds for many values of a, the number is likely prime, but there is still a small chance it could be composite.

The Miller-Rabin primality test is a more sophisticated primality test that is based on the same idea as the Fermat test but is more efficient and has a higher probability of correctly identifying composite numbers. The Miller-Rabin test involves selecting a random value a and then repeatedly squaring it and checking if the resulting values satisfy the equation $a^d \equiv 1 \pmod{n}$ or $a^{2^r d} \equiv -1 \pmod{n}$ for some values of r and d. If the equation holds for many values of r and d, the number is likely prime, but if the equation fails for any r or d, the number is composite.

There are also deterministic primality tests that can determine with certainty whether a number is prime or composite. One example is the AKS primality test, which is based on a polynomial-time algorithm and can handle very large numbers. However, deterministic tests are generally more complex and slower than probabilistic tests.

Chapter 3

Integer Factorization Problem

3.1 Pollard's $p-1$ Method

Pollard's $p-1$ method is an algorithm for factoring a composite number N when one of its prime factors, p, has a small prime factor in $p-1$. The method is based on Fermat's Little Theorem and relies on the properties of the greatest common divisor (GCD) function.

Algorithm 1: Pollard's $p-1$ Method

Data: An integer $N = pq$ which is the product of two primes p and q.
Result: (p, q) or `Failure`.

1 $k_1, k_2 \leftarrow \mathbb{Z}^+$; // Choose two bounds
2 $i \leftarrow 0$;
3 **while** $i < k_1$ **do**
4 $\quad a \xleftarrow{\$} \left(1, \sqrt{N}\right)$; // Initialization
5 $\quad$ **if** $1 < \gcd(a, N) < N$ **then**
6 $\quad\quad$ **return** $\left(\gcd(a, N), \dfrac{N}{\gcd(a, N)}\right)$;
7 $\quad$ **end**
8 $\quad j \leftarrow 2$; // $\gcd(a, N = pq) = 1$
9 $\quad$ **while** $j < k_2$ **do**
10 $\quad\quad a \leftarrow a^j \bmod N$;
11 $\quad\quad d \leftarrow \gcd(a-1, N)$;
12 $\quad\quad$ **if** $1 < d < N$ **then**
13 $\quad\quad\quad$ **return** $(d, N/d)$;
14 $\quad\quad$ **else if** $d = N$ **then**
15 $\quad\quad\quad$ **Goto** line 19;
16 $\quad\quad$ **end**
17 $\quad\quad j \rightarrow j + 1$;
18 $\quad$ **end**
19 $\quad i \leftarrow i + 1$;
20 **end**
21 **return** *Failure*;

3.1.1 The multiple of $p-1$

Fermat's Little Theorem states that if p is a prime number and a is an integer not divisible by p, then

$$a^{p-1} \equiv 1 \pmod{p}.$$

Let $N = pq$, and let $a \xleftarrow{\$} [1, p-1]$. Consider an integer k such that $k = (p-1)t$ for some $t \in \mathbb{Z}$, that is, a multiple of $p-1$, then

$$a^k = \left(a^{p-1}\right)^t \equiv 1 \pmod{p}.$$

Then

$$\begin{aligned} a^k - 1 \equiv 0 \pmod{p} &\implies p \mid (a^k - 1) \\ &\implies \gcd\left(a^k - 1, N\right) \in \{p, N = pq\}. \end{aligned}$$

If $\gcd\left(a^k - 1, N\right) \neq N$ then

$$N = p \times q = \gcd\left(a^k - 1, N\right) \times \frac{N}{\gcd\left(a^k - 1, N\right)}.$$

Pollard's $p-1$ method aims to find such an integer k is a multiple of $p-1$.

3.1.2 How to find the multiple of $p-1$

Existence of the Multiple of $p-1$

Lemma 3.1. *Let p be a prime. Then*

$$\exists n_0 \in \mathbb{N} : [n \geq n_0 \implies p-1 \mid n!].$$

Proof. Define a set S by

$$S := \{k \in \mathbb{N} : k! \equiv 0 \pmod{p-1}\}.$$

Since $(p-1)! \equiv 0 \pmod{p-1}$, we know $p-1 \in S$, i.e., $S \neq \emptyset$. By well-ordering principle, $\exists \min S =: n_0$. Since $n_0 \in S$, $n_0! \equiv 0 \pmod{p-1}$ holds.

We want to show that $n \geq n_0 \implies p-1 \mid n!$. Let $n \geq n_0$ then $n = n_0 + k$ for some $k \in \mathbb{Z}_{\geq 0}$. Now, consider the factorial of n:

$$n! = (n_0 + k)! = (n_0 + k)(n_0 + k - 1) \cdots (n_0 + 1)\, n_0!.$$

Thus,

$$n_0! \equiv 0 \pmod{p-1} \implies p-1 \mid n_0! \implies p-1 \mid \underbrace{n_0!\,(n_0+1)\cdots(n_0+k)}_{=(n_0+k)!=n!}.$$

□

Example 3.1. Consider the prime $65537 = 2^{16} + 1$. Then

$$\begin{aligned}
15!(= 2^{11} \cdot 3^6 \cdot 5^3 \cdot 7^2 \cdot 11 \cdot 13) \bmod 2^{16} &= 22528 && \Rightarrow p - 1 \nmid 15!,\\
16!(= 2^{15} \cdot 3^6 \cdot 5^3 \cdot 7^2 \cdot 11 \cdot 13) \bmod 2^{16} &= 32768 && \Rightarrow p - 1 \nmid 16!,\\
17!(= 2^{15} \cdot 3^6 \cdot 5^3 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17) \bmod 2^{16} &= 32768 && \Rightarrow p - 1 \nmid 17!,\\
18!(= 2^{16} \cdot 3^8 \cdot 5^3 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17) \bmod 2^{16} &= 0 && \Rightarrow p - 1 \mid 18!,\\
19!(= 2^{16} \cdot 3^8 \cdot 5^3 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19) \bmod 2^{16} &= 0 && \Rightarrow p - 1 \mid 19!.
\end{aligned}$$

Thus, arbitrarily choose a value for a, then increment j from 1, and check if the value of $\gcd\left(a^{j!} - 1, N\right) \in (1, N)$:

$$\begin{aligned}
1 < \gcd(a^{1!} - 1, N) < N?\\
1 < \gcd(a^{2!} - 1, N) < N?\\
1 < \gcd(a^{3!} - 1, N) < N?\\
\vdots
\end{aligned}$$

If $gcd(a^{j!} - 1, N) = N$, then $gcd(a^{(j+1)!} - 1, N)$ also equals N, so the test is stopped, a new value for a is chosen, and the test proceeds.

Existence of the Multiple of $p - 1$

Proposition 3.2.

$$\gcd\left(a^{j!} - 1, N\right) = N \implies \gcd\left(a^{(j+1)!} - 1, N\right) = N.$$

Proof. Since

$$\begin{aligned}
\gcd\left(a^{j!} - 1, N\right) = N &\implies N \mid a^{j!} - 1\\
&\implies a^{j!} \equiv 1 \pmod{N}\\
&\implies \left(a^{j!}\right)^{j+1} \equiv 1 \pmod{N}\\
&\implies N \mid a^{(j+1)!} - 1\\
&\implies a^{(j+1)!} - 1 = Nk \quad \text{for some} \quad k \in \mathbb{Z},
\end{aligned}$$

we have

$$\gcd\left(a^{(j+1)!} - 1, N\right) = \gcd(Nk, N) = N.$$

□

3.1.3 How to compute $\gcd\left(a^{n!}-1, N\right)$ efficiently

Existence of the Multiple of $p-1$

Proposition 3.3.

$$\begin{aligned}\gcd\left(a^{(j+1)!}-1, N\right) &= \gcd\left(a^{(j+1)!} \bmod N - 1, N\right) \\ &= \gcd\left(\left(a^{j!} \bmod N\right)^{j+1} - 1, N\right).\end{aligned}$$

Remark 3.1. We can compute $\gcd\left(a^{n!}-1, N\right)$ as follows:

$$\begin{aligned}
&a_0 = a \\
&a_1 = a^{1!} \bmod N = a_0^1 \bmod N && \Rightarrow 1 < \gcd\left(a_1 - 1, N\right) < N? \\
&a_2 = a^{2!} \bmod N = a_1^2 \bmod N && \Rightarrow 1 < \gcd\left(a_2 - 1, N\right) < N? \\
&a_3 = a^{3!} \bmod N = a_2^3 \bmod N && \Rightarrow 1 < \gcd\left(a_3 - 1, N\right) < N? \\
&\vdots
\end{aligned}$$

Example 3.2. Factor $N = 5157437$ using Pollard $p-1$ method.

Sol. content... □

Exercise 3.1. Factor $N = 221$ using Pollard $p-1$ method.

Sol. content... □

3.1.4 Special Case I

Assume that $a^k \bmod p = 1$. Then

$$p \mid a^k - 1 \implies \gcd\left(a^t - 1, pq\right) \in \{p, pq\}.$$

3.1.5 Special Case II

Assume that $p-1 \mid n!$ and $q-1 \mid n!$.

3.2 Fermat Factorization

3.2.1 Fermat Factorization ($N = a^2 - b^2$)

Algorithm 2: Fermat Factorization: $N = a^2 - b^2$

Data: $N = pq$, $B \in \mathbb{Z}_{>0}$.
Result: (p, q) or `Failure`

```
for b = 1 to B do
    t ← N + b^2;
    if t is a square then
        a ← √t;
        return (a − b, a + b);
    end
end
return Failure;
```

Example 3.3. Factor 6077 using Fermat method.

Sol. Note that $\left\lceil \sqrt{6077} \right\rceil = 78$. Then

$$\begin{aligned} 78^2 - 6077 &= 7 \\ &\vdots \\ 81^2 - 6077 &= 484 = 22^2. \end{aligned}$$

Thus, $6077 = (81 + 22)(81 - 21) = 103 \cdot 59$. □

Lemma 3.4. *Let $n \in 2\mathbb{Z} + 1$. Define two sets:*

$$\begin{aligned} S_n &:= \left\{ (a, b) \in \mathbb{Z}^2 : n = ab \text{ with } 0 < b \leq a \right\}, \\ T_n &:= \left\{ \left(u^2, v^2\right) \in \mathbb{Z}^2 : n = u^2 - v^2 \right\}. \end{aligned}$$

Then $\exists \phi : S_n \leftrightarrow T_n$. In other words, there exists a one-to-one correspondence between factorizations of n into two positive integers and differences of two squares that equal n.

Proof. Define two functions:

$$\begin{aligned} f : S_n \to T_n : f((a, b)) &= \left(\left(\frac{a+b}{2} \right)^2, \left(\frac{a-b}{2} \right)^2 \right), \\ g : T_n \to S_n : g\left(\left(u^2, v^2\right) \right) &= \left(|u| + |v|, |u| - |v| \right). \end{aligned}$$

We claim that $g \circ f = \mathrm{Id}_{S_n}$ and $f \circ g = \mathrm{Id}_{T_n}$:

$$\begin{aligned}(g \circ f)(a,b) &= g\left(\left(\frac{a+b}{2}\right)^2, \left(\frac{a-b}{2}\right)^2\right) = \left(\left|\frac{a+b}{2}\right| + \left|\frac{a-b}{2}\right|, \left|\frac{a+b}{2}\right| - \left|\frac{a-b}{2}\right|\right) \\ &= \left(\frac{a+b}{2} + \frac{a-b}{2}, \frac{a+b}{2} - \frac{a-b}{2}\right) \\ &= (a,b)\,, \\ (f \circ g)\left(u^2, v^2\right) &= f\left(|u| + |v|\,, |u| - |v|\right) = \left(|u|^2\,, |v|^2\right) = \left(u^2, v^2\right).\end{aligned}$$

□

3.2.2 Improved Fermat Factorization ($rN = a^2 - b^2$)

Let $N = pq$ and $r < \min\{p, q\}$.

Algorithm 3: Improved Fermat Factorization: $rN = a^2 - b^2$

Data: $N = pq$, $R \in (0, \min\{p,q\}) \setminus \{2\}$ and $B \in \mathbb{Z}_{>0}$.
Result: (p, q) or `Failure`

```
for r = 1 to R do
    for b = 1 to B do
        t ← rN + b^2;
        if t is a square then
            a ← √t;
            d ← gcd(N, a − b);
            if 1 < d < N then
                return (d, N/d);
            end
        end
    end
end
return Failure;
```

Remark 3.2. Let $N \in 2\mathbb{Z} + 1$ and $b \in \mathbb{Z}^+$. Then $2N + b^2$ is not square.

Proof. Note that, for any square number α, $\alpha \bmod 4 \in \{0, 1\}$ since

$$(2k)^2 = 4k^2 \equiv 0 \pmod 4$$
$$(2k+1)^2 = 4k^2 + 4k + 1 \equiv 1 \pmod 4.$$

Let $N = 2m + 1$ then

(Case 1) ($b = 2n$)

$$2N + b^2 = (4m + 2) + 4n^2 \equiv 2 \pmod 4.$$

(Case 2) ($b = 2n + 1$)

$$2N + b^2 = (4m + 2) + (4n^2 + 4n + 1) \equiv 3 \pmod 4.$$

Thus, $2N + b^2 \bmod 4 \in \{2, 3\}$, and hence $2N + b^2$ is not square. □

3.3 Pollard's ρ Method for IFP

Pollard's Rho algorithm is a probabilistic factorization algorithm that is particularly effective in finding small factors of large composite numbers. Given a composite number N, the algorithm aims to find a non-trivial factor p of N.

The algorithm is not deterministic, meaning it may not always find a factor in a fixed number of steps. However, it has an expected running time of $O(\sqrt{p})$, where p is the smallest prime factor of N, making it an efficient algorithm for finding small factors.

Finding a and b that satisfy $rN = a^2 - b^2$ is equivalent to finding a and b satisfying

$$\text{(i) } a^2 \equiv b^2 \pmod{N}, \quad \text{(ii) } \gcd(N, a-b) \in (1, N).$$

Using Floyd's cycle-finding, we can find a and b satisfying above equation.

Define a function $f_{N,c} : \mathbb{Z}_N \to \mathbb{Z}_N$ as follows:

$$f_{N,c}(X) := X^2 + c \bmod N.$$

Proposition 3.5. *A sequence* $\left\{a_n^{(p)}\right\} := \{a_n \bmod p\}$ *satisfies*

$$a_i^{(p)} = a_j^{(p)} \implies a_{i+1}^{(p)} = a_{j+1}^{(p)}$$

Proof.

$$\begin{aligned} a_i^{(p)} = a_j^{(p)} \Leftrightarrow a_i \bmod p = a_j \bmod p &\Leftrightarrow a_i \equiv a_j \pmod{p} \\ &\Leftrightarrow a_i^2 + c \equiv a_j^2 + c \pmod{p} \\ &\Leftrightarrow \left(a_i^2 + c \bmod pq\right) \bmod p = \left(a_j^2 + c \bmod pq\right) \bmod p \\ &\Leftrightarrow a_{i+1} \bmod p = a_{j+1} \bmod p \\ &\Leftrightarrow a_{i+1}^{(p)} = a_{j+1}^{(p)}. \end{aligned}$$

□

Algorithm 4: Pollard's ρ Method for IFP

Data: $N = pq$ (p, q: distinct odd primes), $c \xleftarrow{\$} \mathbb{Z}_N$: constant.
Result: p and q.

```
(a, b) ← (2, 2);
while True do
    (a, b) ← (f(a), (f ∘ f)(b));          // f(x) := x^2 + c mod N
    d ← gcd(N, a − b);
    if 1 < d < N then
        return (d, N/d);
    end
end
```

Chapter 4

Birthday Attack

The birthday problem (also known as the birthday paradox) is a famous problem in probability theory that demonstrates the surprising fact that, in a group of just 23 people, there is a greater than 50% chance that at least two of them share the same birthday. It is based on the assumption that birthdays are uniformly distributed throughout the year (ignoring leap years).

4.1 Birthday Bound

Let n be the number of people in the group. Consider

$$a_1, \cdots, a_k \xleftarrow{\$} \{1, \cdots, n\}$$

with $k \leq n$. Let $P(n)$ be the probability that at least two people share a birthday:

$$P(n) := \Pr\left[\exists (i,j) \text{ with } i \neq j : a_i = a_j\right].$$

To calculate this probability, let $P'(n) := 1 - P(n)$ then

$$\begin{aligned} P'(n) &= 1 - \Pr\left[\exists (i,j) \text{ with } i \neq j : a_i = a_j\right] \\ &= \Pr\left[\forall (i,j) \text{ with } i \neq j : a_i \neq a_j\right]. \end{aligned}$$

That is, $P'(n)$ be the probability that all n birthdays are unique.

Assuming there are 365 possible birthdays, the probability that the first person has a unique birthday is:

$$P_1' = \frac{365}{365} = 1.$$

For the second person to have a unique birthday (different from the first person), there are 364 remaining possibilities:

$$P_2' = \frac{364}{365}.$$

For the third person to have a unique birthday (different from the previous two), there are 363 remaining possibilities:

$$P_3' = \frac{363}{365}.$$

Following this pattern, the probability that all n birthdays are unique is:

$$P'(n) = \frac{365}{365} \times \frac{364}{365} \times \cdots \times \frac{365-(n-1)}{365}.$$

Thus, the probability that at least two people share a birthday is:

$$P(n) = 1 - P'(n) = 1 - \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)\cdots\left(\frac{365-(n-1)}{365}\right).$$

Generalization of Birthday Problem

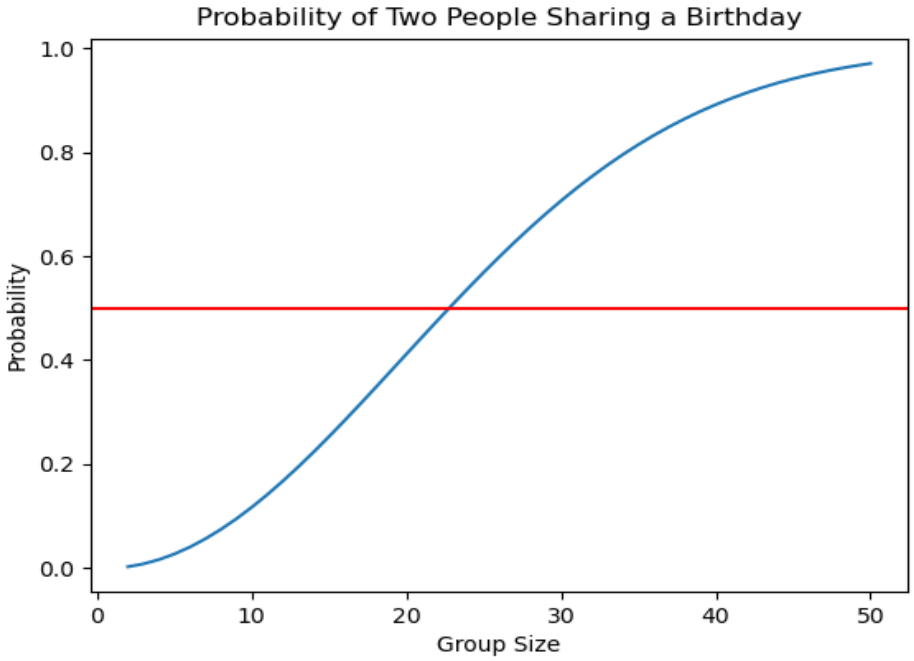
Proposition 4.1. *Assume all birthday are equally likely, and generalize the problem a little:* from n possible birthdays, sample k times with replacement.

$$\begin{aligned} P(\text{a shared birthday}) &= 1 - P(\text{no shared birthdays}) \\ &= 1 - P(\text{all birthdays are unique}) \\ &= 1 - \left(\frac{n}{n}\right)\left(\frac{n-1}{n}\right)\cdots\left(\frac{n-(k-1)}{n}\right) \\ &= 1 - 1 \cdot \prod_{i=1}^{k-1} \frac{n-i}{n} \\ &= 1 - \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right). \end{aligned}$$

Remark 4.1. When $n = 365$, the lowest k for which the above exceeds 0.5 is $k = 23$:

Code 4.1: Birthday Bound (Sage)

```
1 import matplotlib.pyplot as plt
2
```




```
# Define a function to compute the probability of two people
    sharing a birthday for a given group size
def birthday_prob(n):
return 1 - prod([(365-i)/365 for i in range(n)])

# Create a list of group sizes from 2 to 50
group_sizes = range(2, 51)

# Compute the probabilities for each group size
probabilities = [birthday_prob(n) for n in group_sizes]

# Plot the probabilities as a function of group size
plt.plot(group_sizes, probabilities)
plt.xlabel("Group Size")
plt.ylabel("Probability")
plt.title("Probability of Two People Sharing a Birthday")

# Add a horizontal line at the probability of 0.5
plt.axhline(y=0.5, color='r', linestyle='-')

plt.show()
```

Birthday Bound

Theorem 4.2. *From the set $I := \{1, \cdots, n\}$, sample k times with replacement:*

$$a_1, a_2, \cdots, a_k \xleftarrow{\$} \{1, 2, \cdots, n\}$$

where all element of I are equally likely. Then

$$P_{n,k} := \Pr\left[\exists (i,j) \in I^2, i \neq j : a_i = a_j\right] \approx 1 - e^{-\frac{k^2}{2n}}.$$

Proof. Note that

$$P_{n,k} = 1 - \Pr[a_i \neq a_j \text{ for all distinct } i, j] = 1 - \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right).$$

Recall that Taylor series for exponential:

$$e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots = \sum_{k=0}^{\infty}\frac{x^k}{k!}.$$

Note that $e^x \approx 1 + x$ for $x \in (-1, 1)$:

Since $i \in [1, k-1] \subseteq [1, n) \Rightarrow -\frac{i}{n} \in (-1, 1) \Rightarrow 1 + \left(-\frac{i}{n}\right) \approx e^{-i/n}$, we have

$$\begin{aligned} P_{n,k} = 1 - \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right) &\approx 1 - \prod_{i=1}^{k-1} e^{-\frac{i}{n}} \\ &= 1 - e^{-1/n} \cdot e^{-2/n} \cdots e^{-(k-1)/n} \\ &= 1 - e^{-\frac{1}{n}(1+2+\cdots+(k-1))} \\ &= 1 - e^{-\frac{1}{n}\cdot\frac{k(k-1)}{2}} \\ &\approx 1 - e^{-\frac{k^2}{2n}}. \end{aligned}$$

□

Remark 4.2. Find minimum value of k such that $P_{n,k} \geq 1/2$.

$$\begin{aligned} P_{n,k} = 1 - e^{-\frac{k^2}{2n}} \geq \frac{1}{2} &\implies e^{-k^2/2n} \leq \frac{1}{2} \\ &\implies -\frac{k^2}{2n} \leq \ln\frac{1}{2} \\ &\implies (2\ln 2)n \leq k^2 \\ &\implies \left\lceil\sqrt{(2\ln 2)\, n}\right\rceil \leq k. \end{aligned}$$

Thus, $k \geq c\sqrt{n}$, where c is a constant. That is, $O(\sqrt{n})$.

4.2 Set Intersection Bound

Set Intersection Bound

Theorem 4.3. *Consider a set S with $|S| = n$. Let*

$$a_1, a_2, \cdots, a_k \xleftarrow{\$} S, \quad b_1, b_2, \cdots, b_k \xleftarrow{\$} S$$

for $k < n$. For two multi-set $A = [a_1, a_2, \cdots, a_k]$ and $B = [b_1, b_2, \cdots, b_k]$, we have

$$\Pr[A \cap B \neq \emptyset] = 1 - e^{-\frac{k^2}{n}}$$

Proof. For all $i, j \in \{1, \cdots, k\}$, we have

$$\Pr[a_i = b_j] = \frac{1}{n} \implies \Pr[a_i \neq b_j] = 1 - \frac{1}{n}.$$

Thus

$$\begin{aligned}
\Pr[A \cap B \neq \emptyset] &= 1 - \prod_{i=1}^{k} \prod_{j=1}^{k} \Pr[a_i \neq b_j] \\
&= 1 - \prod_{i=1}^{k} \prod_{j=1}^{k} \left(1 - \frac{1}{n}\right) \\
&= 1 - \left(1 - \frac{1}{n}\right)^{k^2} \\
&\approx 1 - \left(e^{-\frac{1}{n}}\right)^{k^2} \\
&= 1 - e^{-\frac{k^2}{n}}.
\end{aligned}$$

□

Remark 4.3. Find minimum value of k such that $P_{n,k} \geq 1/2$.

$$\begin{aligned}
P_{n,k} = 1 - e^{-\frac{k^2}{n}} \geq \frac{1}{2} &\implies e^{-k^2/n} \leq \frac{1}{2} \\
&\implies -\frac{k^2}{n} \leq \ln \frac{1}{2} \\
&\implies (\ln 2) n \leq k^2 \\
&\implies \left\lceil \sqrt{(\ln 2)\, n} \right\rceil \leq k.
\end{aligned}$$

Thus, $k \geq c\sqrt{n}$, where c is a constant. That is, $O(\sqrt{n})$.

4.3 Finding Collision of a (One-way) Function

Finding a collision for a one-way function is a computationally hard problem. We can define a one-way function as:

$$f : \{0,1\}^* \rightarrow \{0,1\}^n$$

By the pigeon-hole principle, there are collision. A collision in a one-way function is a pair of inputs:

$$x_1, x_2 \in \{0,1\}^* \text{ such that } f(x_1) = f(x_2) \text{ but } x_1 \neq x_2.$$

One common method used to find a collision is the *birthday attack*, which is based on the birthday paradox. The birthday paradox states that in a group of 23 people, there is a 50% chance that two people share the same birthday, assuming uniform distribution of birthdays over 365 days.

Algorithm 5: Birthday Attack to Find a Collision in a One-way Function

Data: One-way function f.
Result: Collision pair (x_1, x_2).

Initialize an empty set $S = \emptyset$;
while *True* **do**
  $x \xleftarrow{\$} \{0,1\}^*$;
  $y \leftarrow f(x)$;
  if $y \in S$ **then**
    return (x_1, x_2) s.t. $f(x_1) = f(x_2) = y$;
  else
    $S \leftarrow S \cup \{y\}$;
  end
end

Initialize an empty dictionary D;
while *True* **do**
  $x \xleftarrow{\$} \{0,1\}^*$;
  $y \leftarrow f(x)$;
  if y *is a key in* D **then**
    $x_1 \leftarrow x$;
    $x_2 \leftarrow D[y]$;
    return (x_1, x_2);
  else
    $D \leftarrow D \cup \{(y, x)\}$;
  end
end

The dictionary stores D key-value pairs where the key is the output value y and the value is the corresponding input value x. This way, we can keep track of the input values corresponding to the computed output values.

4.3.1 Native Approach

From 2^n possible outputs of f, sample k times with replacement:

$$P\left(\text{no collision for } k \text{ inputs}\right) = \prod_{i=1}^{k-1}\left(1 - \frac{i}{2^n}\right).$$

Then

$$\prod_{i=1}^{k-1}\left(1 - \frac{i}{2^n}\right) \approx e^{-\frac{k^2}{2\cdot 2^n}} \geq \frac{1}{2} \implies (2\ln 2)2^n \leq k^2$$
$$\implies \sqrt{2\ln 2}\sqrt{2^n} \leq k.$$

For $N = O(\sqrt{2^n})$, we generate the set of input/output pairs

$$S = \{(x_i, f(x_i))\}_{i=1}^{N}.$$

By the birthday bound,

$$\Pr[\exists (x_i, f(x_i)), (x_j, f(x_j)) \in S : f(x_i) = f(x_j)] \approx \frac{1}{2}.$$

Remark 4.4 (Memory Space)**.** To store k inputs, each of which is represented by n bits, we need $k \times n$ bits. If we consider that the probability of a collision is greater than 50% when $k > \sqrt{2^n}$, the memory space required to store these inputs is approximately:

$$\sqrt{2^n} \times n.$$

Thus the memory space needed to store n-bit i/o is approximately $\sqrt{2^n} \times n \times 2$.

Example 4.1. Consider 160-bit output function. Then

$$\begin{aligned}
\text{Memory space} &= \sqrt{2^n} \times n \times 2 \\
&= 2^{80} \times 160 \times 2 \\
&= 2^{80} \times 2^5 \times 5 \times 2 \\
&= 10 \times 2^{85} \\
&= 10 \times 2^{82} \times 2^3 (= 10 \times 2^{82} \text{ bytes}) \\
&= 10 \times 2^{42} \text{ TB}.
\end{aligned}$$

4.3.2 Small-Space Attack using Floyd's Cycle-Finding

Floyd's cycle-finding algorithm is a method to detect cycles in a sequence of values, using a small amount of memory. It is particularly useful for finding cycles in functions, like hash functions or pseudorandom number generators. In the context of a small-space attack, an adversary tries to find a collision (two inputs producing the same output) with limited memory resources.

Algorithm 6: Small-Space Attack; Floyd's Cycle-Finding Algorithm

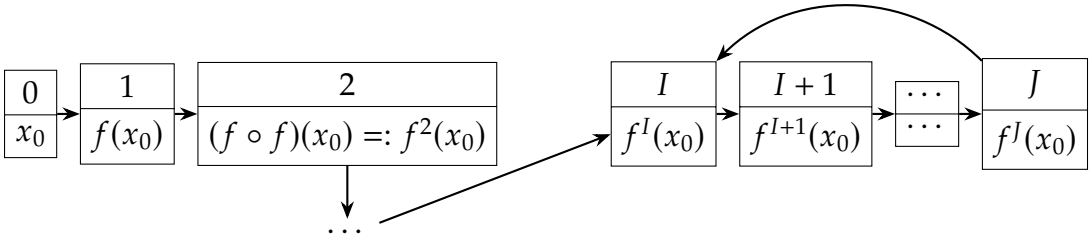
Data: One-way function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$.
Result: Collision pair (x, x') such that $x \neq x'$ but $\mathsf{H}(x) = \mathsf{H}(x')$.

/* Step1: We discover k such that $x_k = x_{2k}$ */

$x_0 \xleftarrow{\$} \{0,1\}^{n+1}$; // $x_0 \notin \operatorname{Img}\mathsf{H}$ is the start node
$(x, x') \leftarrow (x_0, x_0)$;
while *True* **do**
  $(x, x') \leftarrow (H(x), (H \circ H)(x))$;
  if $x = x'$ **then**
    break;
  end
end

/* Step2: Using (x_0, x_k) obtained in Step1, we find the collision pair */

$(x, x') \leftarrow (x_0, x')$;
while *True* **do**
  if $\mathsf{H}(x) = \mathsf{H}(x')$ **then**
    return (x, x')
  else
    $(x, x') \leftarrow (\mathsf{H}(x), \mathsf{H}(x'))$;
  end
end



We define a sequence $\{x_i\}_{i>0}$ as follows:

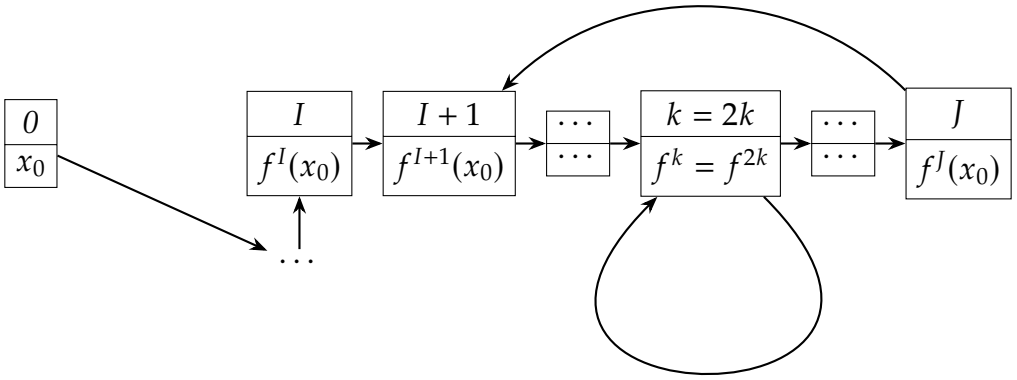
$$x_i := f(x_{i-1}).$$

Here, $x_0 \in \{x \mid f(x) \neq x\}$. Note that a sequence $\{x_i\}$ has a collision pair because $|X| < \infty$ (feat. pigeon-hole principle).

Theorem 4.4. *Define a sequence* $\{x_i\}_{i>0}$ *as follows:*

$$x_i := f(x_{i-1})$$

with initial node $x_0 \notin \mathrm{Img}(f)$. *Then* $\exists k > I$ *such that* $x_k = x_{2k}$*:*



Proof. Let (x_I, x_J) be a collision pair with $0 \le I < J \le N \left(= O\left(\sqrt{2^n}\right)\right)$; that is, $x_I \neq x_J$ but $f(x_I) = f(x_J)$. For $J > I$, we have a circular sequence $\{x_j\}_{j>I}$:

$$x_J, x_{I+1}, x_{I+2}, \cdots, x_{I+(J-I)}(= x_J), \cdots$$

with the period of $\lambda = J - I > 0$. That is,

$$x_j = x_{j+t\cdot\lambda} \quad \text{for some} \quad t \in \mathbb{Z}_{\ge 0}.$$

Define a number

$$k := \min\{t\lambda \in \mathbb{Z}_{>I} : t \in \mathbb{N}\} > I.$$

In other words, k is the smallest multiple greater than I of the period of λ. Clearly, $\lambda \mid k \implies \lambda \mid 2k$, that is, $2k$ is also multiple of the period of λ. Hence the equality $x_k = x_{2k}$ holds. □

Remark 4.5. We know that $\exists k > I$ such that $x_k = x_{2k}$. Since

$$x_k = f(x_{k-1}), \quad x_{2k} = f(x_{2k-1}),$$

by the definition of $\{x_i\}_{i>0}$, then, we obtain $f(x_{k-1}) = f(x_{2k-1})$. But (x_{k-1}, x_{2k-1}) may not be a collision pair. Let x_α is the initial term of a circular sequence $\{x_j\}_{j>I}$. Since k is a multiple of the period of λ of $\{x_j\}_{j>I}$, Then, we see that

$$\begin{aligned} x_\alpha &= x_{\alpha+k} \quad \because k \text{ is the multiple of the period of } \lambda \\ f(x_{\alpha-1}) &= f(x_{\alpha+k-1}) \quad \text{by the def. of } \{x_i\}_{i>0}, \end{aligned}$$

and so $(x_{\alpha-1}, x_{\alpha+k-1})$ is a clear collision pair.

Chapter 5

Discrete Logarithm Problem

In cryptography, the discrete logarithm problem (DLP) is the problem of computing x given g, h, and n, where

(i) $G = \langle g \rangle$ with $|G| = n$;

(ii) $h = g^x \in G$.

In other words,

"Given $h \in \langle g \rangle$, find $x \in [0, n)$ such that $g^x = h$".

5.1 DLP Solutions

Order	Algorithm	Complexity	Cyclic Group
n	Exhaustive Search	$O(n)$	General
n	Baby-Step/Giant-Step	$O(\sqrt{n})$	General
p (prime)	Pollard-ρ	$O(\sqrt{p})$	General

5.2 Exhaustive Search

- Time Complexity: $O(n)$
- Memory Complexity: $O(1)$

Algorithm 7: DLP Solution: Brute Force Search

Data: Cyclic group $G = \langle g \rangle$ with order n, and $h \in G$.
Result: $x \in \{0, 1, \cdots, n-1\}$ such that $g^x = h$.

```
1 t ← 1;
2 for j ← 0 to n − 1 do
3 |   if t = h then
4 |   |   return j;
5 |   else
6 |   |   t ← tg;
7 |   end
8 end
```

5.3 Shanks' Algorithm: Baby-Step/Giant-Step (1971)

Algorithm 8: Shanks' Baby-step/Giant-step Algorithm

Data: A group G of order n with generator g, and an element h in G.
Result: The discrete logarithm $x \in \mathbb{Z}_n$ such that $g^x = h$.

1 $m \leftarrow \lfloor \sqrt{n} \rfloor + 1$; // $m > \sqrt{n}$
2 $SB \leftarrow \left\{(i, g^i) : i = 0, 1, \cdots, m-1\right\}$; // Baby-step
3 $SB \leftarrow \left\{\left(j, h(g^{-m})^j\right) : i = 0, 1, \cdots, m-1\right\}$; // Giant-step
4 Find i, j such that $g^i = h\,(g^{-m})^j$;
5 **return** $x \leftarrow mj + i$;

Consider cyclic group $G = \langle g \rangle$ of order n. We want to find k such that $g^k = h$ for some $h \in G$. Clearly, $k \in \{0, 1, \cdots, n-1\} = \mathbb{Z}_n$. Define a number $m := \left\lfloor \sqrt{n} \right\rfloor + 1$. By division theorem, $\exists! q, r \in \mathbb{Z}_{\geq 0}$ such that

$$k = mq + r, \quad 0 \leq r < m.$$

Note that

$$\sqrt{n} < m = \frac{k-r}{q} \leq \frac{k}{q} < \frac{n}{q} \implies q < \frac{n}{\sqrt{n}} = \sqrt{n} \implies 0 \leq q < \sqrt{n} < m.$$

That is, $r, q \in \mathbb{Z}_m$. Then

$$h = g^k = g^{mq+r} \implies g^r = h\,(g^{-m})^q\,.$$

Define two sets

$$BS := \left\{g^i : i = 0, 1, \cdots, m-1\right\},$$
$$GS := \left\{h\,(g^{-m})^j : j = 0, 1, \cdots, m-1\right\}.$$

Clearly $BS \cap GS \neq \emptyset$. Thus we can find $i, j \in \mathbb{Z}_m$ such that $g^i = h\,(g^{-m})^j$. Hence we obtain the discrete logarithm

$$k = mj + i.$$

5.4 Pollard ρ Method for DLP (1978)

The Pollard's rho method is based on the idea of finding a collision in a sequence of group elements and their corresponding exponents.

Algorithm 9: Pollard's ρ Method for DLP

Data: Cyclic group $G = \langle g \rangle = \dot{\bigcup}_{i=1}^{3} G_i$ with prime order p, and $h \in G$.
Result: The discrete logarithm $x \in \mathbb{Z}_p$ such that $g^x = h$ or `Failure`

/* $\mathrm{Proj}_0, \mathrm{Proj}_1, \mathrm{Proj}_2 : G^3 \to G : \mathrm{Proj}_j(x_0, x_1, x_2) = x_j, \quad j = 0, 1, 2.$ */

$x_0 \leftarrow (g^0 h^0, 0, 0)$; // x_0 is the start node
$(X, X') \leftarrow (x_0, x_0)$;
while *True* **do**
  $(X, X') \leftarrow (f(X), f^2(X'))$; // $f(x_j, a_j, b_j) = (x_{j+1}, a_{j+1}, b_{j+1})$
  if $\mathrm{Proj}_0(X) = \mathrm{Proj}_0(X')$ **then**
    break ; // $x_j = x_{2j}$
  end
end
/* $b_{2j} - b_j \neq 0$ */
if $\mathrm{Proj}_2(X) \neq \mathrm{Proj}_2(X')$ **then**
  $t_1 \leftarrow (\mathrm{Proj}_1(X) - \mathrm{Proj}_1(X))$;
  $t_2 \leftarrow (\mathrm{Proj}_2(X') - \mathrm{Proj}_2(X'))^{-1}$; // EEA
  return $t_1 t_2 \bmod p$
else
  return `Failure`;
end

Consider a group G of *prime* order p with generator g. We want to find k such that $g^k = h \in G = \langle g \rangle$.

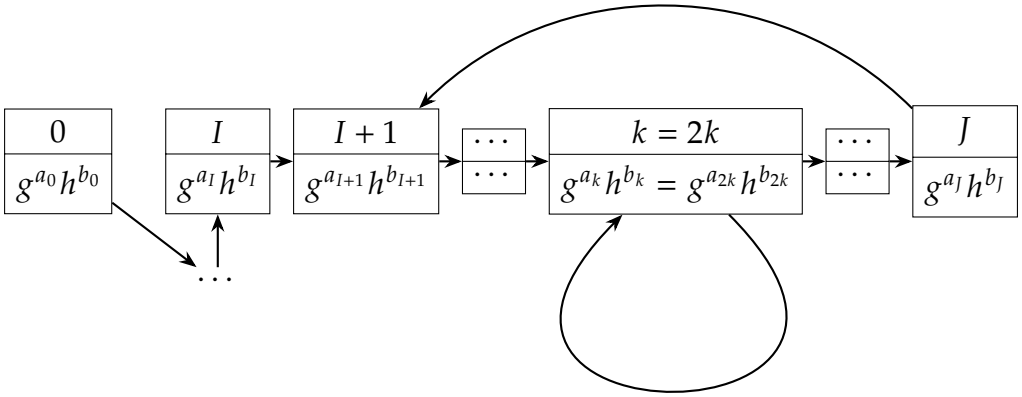
(Step I)

$$G = G_1 \dot{\cup} G_2 \dot{\cup} G_3, \quad |G_1| \approx |G_2| \approx |G_3|.$$

(Step II) Define a sequence $\{x_j, a_j, b_j\}_{j=0}^{\infty} = \{g^{a_j} h^{b_j}, a_j, b_j\}_{j=0}^{\infty}$ as follows:

$$(x_j, a_j, b_j) := \begin{cases} (g^0 h^0, 0, 0) & : j = 0, \\ (x_{j-1} h, a_{j-1}, b_{j-1} + 1 \bmod p) & : j \geq 1 \text{ and } x_{j-1} \in G_1, \\ \left(x_{j-1}^2, 2a_{j-1} \bmod p, 2b_{j-1} \bmod p\right) & : j \geq 1 \text{ and } x_{j-1} \in G_2, \\ (g x_{j-1}, a_{j-1} + 1 \bmod p, b_{j-1}) & : j \geq 1 \text{ and } x_{j-1} \in G_3. \end{cases}$$

(Step III) Using Floyd's cycle-finding, we find (x_j, a_j, b_j) and $(x_2 j, a_{2j}, b_{2j})$ such that $x_j = x_{2j}$:



(Step IV) Find a collision pair!

$$
\begin{aligned}
g^{a_j} h^{b_j} = g^{a_{2j}} h^{b_{2j}} &\implies g^{a_j - a_{2j}} = h^{b_{2j} - b_j} = (g^x)^{b_{2j} - b_j} \\
&\implies a_j - a_{2j} \equiv x \left(b_{2j} - b_j\right) \pmod{p} \\
&\implies x = \left(a_j - a_{2j}\right) \left(b_{2j} - b_j\right)^{-1} \bmod p \quad \because p \text{ is a prime.}
\end{aligned}
$$

Chapter 6

Algorithms

6.1 Exponentiation

In this chapter, we introduce three algorithms to compute the exponentiation $x^n = \underbrace{x \times \cdots \times x}_{n}$ (or the constant multiple $nx = \underbrace{x + \cdots + x}_{n}$) for an l-bit positive integer $n = \sum_{j=0}^{l-1} n_j 2^j$ with $n_j \in \{0, 1\}$ and $n_{l-1} = 1$[1].

n_{l-1}	n_{l-2}	$\cdots$	$\cdots$	$\cdots$	n_1	n_0
2^{l-1}	2^{l-2}	$\cdots$	$\cdots$	$\cdots$	2^1	2^0

$\longrightarrow n = \sum_{j=0}^{l-1} n_j 2^j$

(1) Left-to-Right Binary Method

(2) Right-to-Left Binary Method

(3) Multiply-and-Square Method (Montgomery Ladder)

Note. The operation x^n is used in algorithms such as

(i) Diffie-Hellman (DH),

(ii) Digital Signature Algorithm (DSA),

(iii) and RSA,

while the operation nx is used in

(i) Elliptic Curve Diffie-Hellman (ECDH)

(ii) and Elliptic Curve Digital Signature Algorithm (ECDSA).

[1]MSB:Most Significant Bit

6.1.1 Left-to-Right Binary Method

Algorithm 10: Exponentiation: Left-to-Right Binary Method

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: x^n

```
1 t ⟶ 1;
2 for i ⟵ l − 1 downto 0 do
3 |   t ⟵ t^2 ;            // Squaring
4 |   t ⟵ t x^{n_i} ;       // Fixed
  |    multiplication
5 end
6 return t;
```

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: nx

```
1 t ⟶ 0;
2 for i ⟵ l − 1 downto 0 do
3 |   t ⟵ 2t ;             // Doubling
4 |   t ⟵ t + n_i x ;       // Fixed
  |    addition
5 end
6 return t;
```

Algorithm 11: **Modular** Exponentiation: Left-to-Right Binary Method

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: x^n over $\mathbb{F}_p^*$

```
1 t ⟶ 1 ;
2 for i ⟵ l − 1 downto 0 do
3 |   t ⟵ t^2 mod p ;
4 |   t ⟵ t x^{n_i} mod p ;
5 end
6 return t ;
```

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: nx over $\mathbb{Z}_p$

```
1 t ⟶ 0 ;
2 for i ⟵ l − 1 downto 0 do
3 |   t ⟵ 2t mod p ;
4 |   t ⟵ (t + n_i x) mod p ;
5 end
6 return t ;
```

Left-to-Right Binary Method utilizes the following equation:

$$\begin{aligned}(n)_l = n_{l-1} \parallel n_{l-2} \parallel \cdots \parallel n_1 \parallel n_0 &= \sum_{j=0}^{l-1} n_j 2^j \\ &= n_{l-1}2^{l-1} + n_{l-2}2^{l-2} + \cdots + n_2 2^2 + n_1 2 + n_0 \\ &= 2\left(n_{l-1}2^{l-2} + n_{l-2}2^{l-3} + \cdots + n_2 2 + n_1\right) + n_0 \\ &= 2\left(2\left(n_{l-1}2^{l-3} + n_{l-2}2^{l-4} + \cdots + n_2\right) + n_1\right) + n_0 \\ &= 2\left(\cdots 2\left(2\left(2(0) + n_{l-1}\right) + n_{l-2}\right)\cdots\right) + n_0.\end{aligned}$$

Then

$$nx = \left(\sum_{j=0}^{l-1} n_j 2^j\right) x = 2\left(\sum_{j=1}^{l-2} n_{j+1} 2^j\right) x + n_0 x,$$

$$x^n = x^{\sum_{j=0}^{l-1} n_j 2^j} = x^{2\sum_{j=1}^{l-2} n_j 2^j + n_0} = \left(x^{\sum_{j=1}^{l-2} n_j 2^j}\right)^2 \times x^{n_0}.$$

Correctness of algorithm (Left-to-Right Binary Method). Let

$$\begin{aligned}
m_l &:= 0, \\
m_{l-1} &:= n_{l-1}, \\
m_{l-2} &:= n_{l-2} + 2n_{l-1}, \quad (n_{l-1} \parallel n_{l-2}) \\
m_{l-3} &:= n_{l-3} + 2n_{l-2} + 2^2 n_{l-1}, \quad (n_{l-1} \parallel n_{l-2} \parallel n_{l-3}) \\
&\vdots \\
m_0 &:= n_0 + 2n_1 + 2^2 n_2 + \cdots + 2^{l-1} n_{l-1}, \quad (n_{l-1} \parallel n_{l-2} \parallel n_{l-3} \parallel \cdots \parallel n_0 = n).
\end{aligned}$$

That is, the following recurrence relation is generated:

$$m_j := \begin{cases} 0 & : j = l \\ 2m_{j+1} + n_j & : j = l-1, l-2, \ldots, 1, 0. \end{cases}$$

Since $m_0 = n$, we obtain x^n or nx through the following process:

$$x^{m_l} = x^0 = 1 \longrightarrow x^{m_{l-1}} \longrightarrow x^{m_{l-2}} \longrightarrow \cdots \longrightarrow x^{m_1} \longrightarrow x^{m_0} = x^n,$$

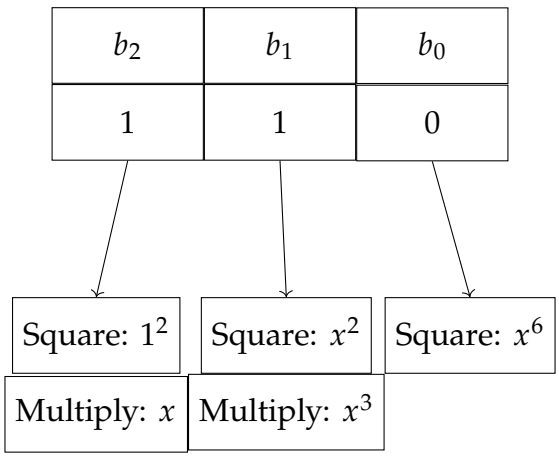
$$x^{m_j} \longleftarrow x^{2m_{j+1}+n_j} = x^{2m_{j+1}} x^{n_j} = \left(x^{m_{j+1}}\right)^2 x^{n_j} = \begin{cases} \left(x^{m_{j+1}}\right)^2 & : n_j = 0 \\ \left(x^{m_{j+1}}\right)^2 x & : n_j = 1. \end{cases}$$

$$m_l x = 0 \cdot x = 0 \longrightarrow m_{l-1}x \longrightarrow m_{l-2}x \longrightarrow \cdots \longrightarrow m_1 x \longrightarrow m_0 x = nx,$$

$$m_j x \longleftarrow \left(2m_{j+1} + n_j\right) x = 2\left(m_{j+1}x\right) + n_j x = \begin{cases} 2\left(m_{j+1}x\right) & : n_j = 0 \\ 2\left(m_{j+1}x\right) + x & : n_j = 1. \end{cases}$$

□

Example 6.1. $x^6 = x^{\texttt{0b110}} = \left(\left((1)^2 x\right)^2 x\right)^2$ and $6x = (\texttt{0b110})x = 2\left(2\left(2\left(0\right) + x\right) + x\right)$.



6.1.2 Right-to-Left Binary Method

Algorithm 12: Exponentiation: Right-to-Left Binary Method

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: x^n

```
1 t_0, t_1 ⟶ 1, x;
2 for i ⟵ 0 to l − 1 do
3 |   t_0 ⟵ t_0 t_1^{n_i} ;
  |    // Multiplication
4 |   t_1 ⟵ t_1^2 ;          // Squaring
5 end
6 return t_0;
```

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: nx

```
1 t_0, t_1 ⟶ 0, x;
2 for i ⟵ 0 to l − 1 do
3 |   t_0 ⟵ t_0 + n_i t_1 ;   // Addition
4 |   t_1 ⟵ 2t_1 ;            // Doubling
5 end
6 return t_0;
```

Algorithm 13: **Modular** Exponentiation: Right-to-Left Binary Method

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: x^n over $\mathbb{F}_p^*$

```
1 t_0, t_1 ⟶ 1, x;
2 for i ⟵ 0 to l − 1 do
3 |   t_0 ⟵ t_0 t_1^{n_i} mod p ;
4 |   t_1 ⟵ t_1^2 mod p ;
5 end
6 return t_0 ;
```

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: nx over $\mathbb{Z}_p$

```
1 t_0, t_1 ⟶ 0, x;
2 for i ⟵ 0 to l − 1 do
3 |   t_0 ⟵ t_0 + n_i t_1 mod p ;
4 |   t_1 ⟵ 2t_1 mod p ;
5 end
6 return t_0 ;
```

Right-to-Left Binary Method utilizes the following equation:

$$x^n = x^{\sum_{j=0}^{l-1} n_j 2^j} = x^{n_0 2^0 + n_1 2^1 + n_2 2^2 + \cdots + n_{l-1} 2^{l-1}} = \left(x^{2^0}\right)^{n_0} \times \left(x^{2^1}\right)^{n_1} \times \left(x^{2^2}\right)^{n_2} \times \cdots$$
$$= \left(x^{2^0}\right)^{n_0} \times \left(\left(x^{2^0}\right)^2\right)^{n_1} \times \left(\left(\left(x^{2^0}\right)^2\right)^2\right)^{n_2} \times \cdots$$

Then

$$x^{2^0} \xrightarrow{\text{squaring}} \left(x^{2^0}\right)^2 \xrightarrow{\text{squaring}} \left(\left(x^{2^0}\right)^2\right)^2 \xrightarrow{\text{squaring}} \cdots$$

Correctness of algorithm (Right-to-Left Binary Method). Let

$$\begin{aligned}
m_0 &:= 0, \\
m_1 &:= n_0, \\
m_2 &:= 2n_1 + n_0 \\
m_3 &:= 2^2 n_2 + 2n_1 + n_0 \\
&\vdots \\
m_l &:= 2^{l-1} n_{l-1} + 2^{l-2} n_{l-2} + \cdots + 2^2 n_2 + 2n_1 + n_0.
\end{aligned}$$

That is, the following recurrence relation is generated:

$$m_j := \begin{cases} 0 & : j = l \\ 2m_{j+1} + n_j & : j = l-1, l-2, \ldots, 1, 0. \end{cases}$$

Since $m_0 = n$, we obtain x^n or nx through the following process:

$$x^{m_l} = x^0 = 1 \longrightarrow x^{m_{l-1}} \longrightarrow x^{m_{l-2}} \longrightarrow \cdots \longrightarrow x^{m_1} \longrightarrow x^{m_0} = x^n,$$

$$x^{m_j} \longleftarrow x^{2m_{j+1}+n_j} = x^{2m_{j+1}} x^{n_j} = \left(x^{m_{j+1}}\right)^2 x^{n_j} = \begin{cases} \left(x^{m_{j+1}}\right)^2 & : n_j = 0 \\ \left(x^{m_{j+1}}\right)^2 x & : n_j = 1. \end{cases}$$

$$m_l x = 0 \cdot x = 0 \longrightarrow m_{l-1} x \longrightarrow m_{l-2} x \longrightarrow \cdots \longrightarrow m_1 x \longrightarrow m_0 x = nx,$$

$$m_j x \longleftarrow \left(2m_{j+1} + n_j\right) x = 2\left(m_{j+1} x\right) + n_j x = \begin{cases} 2\left(m_{j+1} x\right) & : n_j = 0 \\ 2\left(m_{j+1} x\right) + x & : n_j = 1. \end{cases}$$

□

6.1.3 Multiply-and-Square Method (Montgomery Ladder)

Algorithm 14: Exponentiation: Multiply-and-Square Method (a.k.a. Montgomery Ladder)

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: x^n

```
t_0, t_1 → 1, x;
for i ← l − 1 downto 0 do
    t_{1−n_i} ← t_0 t_1;
    t_{n_i} ← t_{n_i}^2;
end
return t_0;
```

Data: x and $n = \sum_{j=0}^{l-1} n_j 2^j$
Result: nx

```
t_0, t_1 → 0, x;
for i ← l − 1 downto 0 do
    t_{1−n_i} ← t_0 + t_1;
    t_{n_i} ← 2t_{n_i};
end
return t_0;
```

Correctness of algorithm (Multiply-and-Square Method (a.k.a. Montgomery Ladder)). For l-bit integer $n = \sum_{j=0}^{l-1} n_j 2^j = n_{l-1} \| n_{l-2} \| \cdots \| n_1 \| n_0$, the followings are defined:

$$L_j := \left\lfloor \frac{n}{2^j} \right\rfloor = (n \gg j) = n_{l-1} \| \cdots \| n_j,$$
$$R_j := L_j + 1$$

for $j = 0, 1, \cdots, l$. Note that

$$\begin{cases} L_0 = \left\lfloor \frac{n}{2^0} \right\rfloor = \lfloor n \rfloor = n \\ R_0 = L_0 + 1 = n + 1 \end{cases}, \quad \begin{cases} L_l = \left\lfloor \frac{n}{2^l} \right\rfloor = 0 \\ R_l = L_l + 1 = 1 \end{cases}.$$

Since

$$\begin{aligned} L_j = (n \gg j) = n_{l-1} \| \cdots \| n_{j+1} \| n_j &= \left(n_{l-1} \| \cdots \| n_{j+1} \| 0\right) + n_j \\ &= \left(L_{j+1} \ll 1\right) + n_j \\ &= 2\left(L_{j+1}\right) + n_j, \end{aligned}$$

we obtain

$$\begin{aligned} L_j &= 2L_{j+1} + n_j \\ &= L_{j+1} + \left(R_{j+1} - 1\right) + n_j \quad \because R_{j+1} = L_{j+1} + 1 \\ &= 2R_{j+1} + n_j - 2 \quad \because R_{j+1} = L_{j+1} + 1. \end{aligned}$$

Then

L_j	$n_j = 0$	$n_j = 1$
$2L_{j+1} + n_j$	$2L_{j+1}$	$2L_{j+1} + 1$
$L_{j+1} + \left(R_{j+1} - 1\right) + n_j$	$L_{j+1} + R_{j+1} - 1$	$L_{j+1} + R_{j+1}$
$2R_{j+1} + n_j - 2$	$2R_{j+1} - 2$	$2R_{j+1} - 1$

From the aforementioned equation, the following recurrence relation is derived:

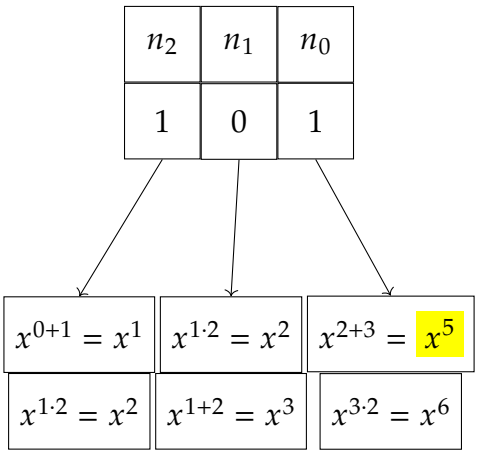
$$\begin{cases} (L_l, R_l) = (L_l, L_l + 1) \leftarrow (0, 1), \\ (L_j, R_j) = (L_j, L_j + 1) \leftarrow \begin{cases} (2L_{j+1}, L_{j+1} + R_{j+1}) & : n_j = 0 \\ (L_{j+1} + R_{j+1}, 2R_{j+1}) & : n_j = 1. \end{cases} \end{cases}$$

Thus,

$$(x^{L_l}, x^{R_l}) = (1, x) \xrightarrow{n_{l-1}} (x^{L_{l-1}}, x^{R_{l-1}}) \xrightarrow{n_{l-2}} \cdots \xrightarrow{n_1} (x^{L_1}, x^{R_1}) \xrightarrow{n_0} (x^{L_0} = x^n, x^{R_0}).$$

□

Example 6.2. $x^5 = x^{\texttt{0b101}}$ and $5x = (\texttt{0b101})x$. Note that $5 = 1 \parallel 0 \parallel 1$.



Chapter 7

Classification

7.1 Encryption

Symmetric Key Encryption

(1) Key Generation: $\mathsf{GenKey} : \mathbb{N} \to \mathcal{K}$; $k \leftarrow \mathsf{GenKey}(1^n)$ for a security parameter n

(2) Encryption: $\mathsf{Encrypt} : \mathcal{K} \times \mathcal{P} \to \mathcal{C}$; $c \leftarrow \mathsf{Encrypt}(k; p)$

(3) Decryption: $\mathsf{Decrypt} : \mathcal{K} \times \mathcal{C} \to \mathcal{P}$; $p \leftarrow \mathsf{Encrypt}(k; c)$

Public Key Encryption

(1) Key Generation: $\mathsf{GenKeyPair} : \mathbb{N} \to \mathcal{K}_p \times \mathcal{K}_s$;

$$pk, sk \leftarrow \mathsf{GenKeyPair}(1^n)$$

for a security parameter n

(2) Encryption: $\mathsf{Encrypt} : \mathcal{K}_p \times \mathcal{P} \to \mathcal{C}$;

$$c \leftarrow \mathsf{Encrypt}(pk; p)$$

(3) Decryption: $\mathsf{Decrypt} : \mathcal{K}_s \times \mathcal{C} \to \mathcal{P}$;

$$p \leftarrow \mathsf{Encrypt}(sk; c)$$

We call pk public key and sk secret key (or private key).

7.2 Objective of Attack

7.2.1 OW: Onewayness

Challenger	OW security game (symmetric and public)	Adversary $\mathcal{A}$
$k \leftarrow \mathsf{GenKey}(1^n)$ $pk, sk \leftarrow \mathsf{GenKeyPair}(1^n)$		
$p \xleftarrow{\$} \mathcal{P}$ $c \leftarrow \mathsf{Encrypt}(k; p)$ $c \leftarrow \mathsf{Encrypt}(pk; p)$	$\xrightarrow{c}$	
if $p = p'$, then $\mathcal{A}$ wins, else $\mathcal{A}$ loses.	$\xleftarrow{p'}$	$\mathcal{A}$ thinks p' s.t. $\mathsf{Decrypt}(k; c) = p'$ $\mathsf{Decrypt}(sk; c) = p'$

Table 7.1: OW security game.

In the OW attack, the advantage of the adversary $\mathcal{A}$, denoted as $\mathrm{Adv}^{\mathsf{OW}}_{\Pi}(\mathcal{A})$, is expressed as follows:

$$\begin{aligned}\mathrm{Adv}^{\mathsf{OW}}_{\Pi}(\mathcal{A}) &:= \Pr\left[p \leftarrow \mathcal{A}\left(\mathsf{Encrypt}(k; p)\right)\right] \\ &:= \Pr\left[p \leftarrow \mathcal{A}\left(\mathsf{Encrypt}(pk; p)\right)\right]\end{aligned}$$

where $p \xleftarrow{\$} \mathcal{P}$,

$$k \leftarrow \mathsf{GenKey}(1^n) \quad \text{and} \quad pk, sk \leftarrow \mathsf{GenKeyPair}(1^n).$$

7.2.2 IND: Indistinguishability

Step	Challenger	IND security game ⟷ symmetric and public	Adversary $\mathcal{A}$
Phase I	$k \leftarrow \mathsf{GenKey}(1^n)$ $pk, sk \leftarrow \mathsf{GenKeyPair}(1^n)$ $b \xleftarrow{\$} \{0,1\}$	$\xleftarrow{p_0, p_1}$	$\mathcal{A}$ choose $p_0, p_1 \in \mathcal{P}$ s.t. $p_0 \neq p_1$ and $\lvert p_0 \rvert = \lvert p_1 \rvert$
Phase II	$c \leftarrow \mathsf{Encrypt}(k; p_b)$ $c \leftarrow \mathsf{Encrypt}(pk; p_b)$ if $b = b'$, then $\mathcal{A}$ wins, else $\mathcal{A}$ loses.	$\xrightarrow{c}$ $\xleftarrow{b' \leftarrow j}$	$\mathcal{A}$ thinks p_j s.t. $\mathsf{Decrypt}(k; c) = p_j$ $\mathsf{Decrypt}(sk; c) = p_j$

Table 7.2: IND security game.

$\mathrm{Adv}^{\mathsf{IND}}_{\Pi}(\mathcal{A})$ is expressed as follows:

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND}}_{\Pi}(\mathcal{A}) &:= 2\left|b \leftarrow \mathcal{A}\left(\mathsf{Encrypt}(k; p_b)\right) - \frac{1}{2}\right| \\ &:= 2\left|b \leftarrow \mathcal{A}\left(\mathsf{Encrypt}(pk; p_b)\right) - \frac{1}{2}\right|\end{aligned}$$

where $b \xleftarrow{\$} \{0,1\}$,

$$k \leftarrow \mathsf{GenKey}(1^n) \quad \text{and} \quad pk, sk \leftarrow \mathsf{GenKeyPair}(1^n).$$

Theorem 7.1. IND-*secure* $\implies$ SEM-*secure.*

Proof. Note that the following game: □

Challenger	$\overset{\text{IND}}{\longleftrightarrow}$ game	Adversary $\mathcal{A}$	$\overset{\text{SEM}}{\longleftrightarrow}$ game	Adversary $\mathcal{B}$
$k \leftarrow \mathsf{GenKey}(1^n)$ $pk, sk \leftarrow \mathsf{GenKeyPair}(1^n)$				
	$\xleftarrow{p_0, p_1}$	$\mathcal{A}$ choose p_0, p_1 such that $g(p_0) \neq g(p_1)$ and $\lvert p_0 \rvert = \lvert p_1 \rvert$		
$b \xleftarrow{\$} \{0,1\}$				
$c \leftarrow \mathsf{Encrypt}(k; p_b)$ $c \leftarrow \mathsf{Encrypt}(pk; p_b)$	$\xrightarrow{c}$		$\xrightarrow{c}$	$t \leftarrow g\,(\mathsf{Decrypt}(k; c))$ $t \leftarrow g\,(\mathsf{Decrypt}(sk; c))$
if $b = b'$, then $\mathcal{A}$ wins, else $\mathcal{A}$ loses.	$\xleftarrow{b'}$	if $g(p_0) = t$, then, $b' \leftarrow 0$, else $b' \leftarrow 1$	$\xleftarrow{\hat{t}}$	

Table 7.3: $\mathcal{A}$'s IND game using $\mathcal{B}$.

7.3 Classification of Ability of Adversary

(1) Passive Attack (PASS), a.k.a Ciphertext-Only Attack (COA)

(2) Chosen-Plaintext Attack (CPA)

(3) Chosen-Ciphertext Attack (CCA)

(4) Adaptive Chosen-Ciphertext Attack (CCA2)

Theorem 7.2.

$$\text{CCA2-}secure \Rightarrow \text{CCA-}secure \Rightarrow \text{CPA-}secure \Rightarrow \text{PASS-}secure.$$

7.4 Security Relation

Theorem 7.3. IND-*secure scheme is* OW-*secure.*

Proof. Note that the following game:

Challenger	$\overset{\text{IND}}{\longleftrightarrow}$ game	Adversary $\mathcal{A}$	$\overset{\text{OW}}{\longleftrightarrow}$ game	Adversary $\mathcal{B}$
$k \xleftarrow{\$} \mathsf{GenKey}(1^n)$				
	$\xleftarrow{p_0, p_1}$	$p_0, p_1 \leftarrow \mathcal{A}$		
$b \xleftarrow{\$} \{0,1\}$				
$c \leftarrow \mathsf{Encrypt}(k; p_b)$	$\xrightarrow{c}$		$\xrightarrow{c}$	
if $b = b'$, then $\mathcal{A}$ wins, else $\mathcal{A}$ loses.	$\xleftarrow{b'}$	if $p_0 = \hat{p}$, then, $b' \leftarrow 0$, else $b' \leftarrow 1$	$\xleftarrow{\hat{p}}$	$\mathcal{A}$ thinks $\hat{p}$ s.t. $\mathsf{Decrypt}(k; c) = \hat{p}$

Table 7.4: $\mathcal{A}$'s IND game using $\mathcal{B}$.

□

Theorem 7.4. *Deterministic scheme is* ***NOT*** IND-CPA-*secure.*

Proof. content... □

Example 7.1. Textbook RSA is deterministic, and so it is not IND-CPA-secure.

Theorem 7.5. *Homomorphic scheme is* ***NOT*** OW-CCA2-*secure.*

Proof. content... □

7.5 Authentication and Signature

Message Authentication Code (MAC)

(1) Key Generation: $\mathsf{GenKey} : \mathbb{N} \to \mathcal{K}$; $k \leftarrow \mathsf{GenKey}(1^n)$ for a security parameter n

(2) Value gen: $\mathsf{GenMacTag} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$; $tag \leftarrow \mathsf{GenMacTag}(k; m)$

(3) Gen. Verify: $\mathsf{VerifyMacTag} : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{\texttt{Valid}, \texttt{Invalid}\}$; $ans \leftarrow \mathsf{VerifyMacTag}(k; m, tag)$

Signature

(1) Key Generation: $\mathsf{GenKeyPair} : \mathbb{N} \to \mathcal{K}_p \times \mathcal{K}_s$;

$$pk, sk \leftarrow \mathsf{GenKeyPair}(1^n)$$

for a security parameter n

(2) Gen. sig.: $\mathsf{Sign} : \mathcal{K}_p \times \mathcal{M} \to \mathcal{S}$;

$$sig \leftarrow \mathsf{Sign}(pk; m)$$

(3) Gen. veri.: $\mathsf{Verify} : \mathcal{K}_s \times \mathcal{M} \times \mathcal{S} \to \{\texttt{Valid}, \texttt{Invalid}\}$;

$$ans \leftarrow \mathsf{Verify}(sk; m, sig)$$

We call pk public key and sk secret key (or private key).

Chapter 8

IFP-based Ciphers

8.1 RSA-OAEP

Since both Textbook RSA and RSA-CRT are deterministic algorithms, they are not IND-CPA-secure. In 1994, M. Bellare and P. Rogaway proposed a method of transforming a deterministic algorithm into a specific protocol unrelated to the key before encrypting the message using a random oracle, to ensure IND-CPA-secure. This is referred to as OAEP (Optimal Asymmetric Encryption Padding) transformation.

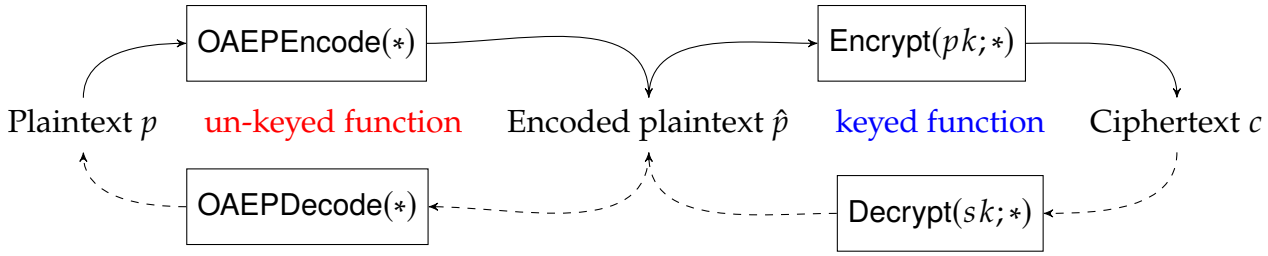


Figure 8.1: OAEP Transformation

Algorithm 15: RSA-OAEP: Encrypt, Decrypt

Data: A public key pk, and a plaintext p
Result: A ciphertext c or `error`

```
/* pk = (e, N)                                   */
procedure Encrypt(pk; p):
    p̂ ← OAEPEncode(p);
    if p̂ = error then
        return error;
    end
    c ← (p̂)^e mod N;
    return c;
end
```

Data: A secret key sk, and a ciphertext c
Result: A ciphertext p or `error`

```
/* sk = (d, N)                                   */
procedure Decrypt(sk; c):
    p̂ ← c^d mod N;
    p ← OAEPDecode(p̂);
    if p = error then
        return error;
    end
    return p;
end
```

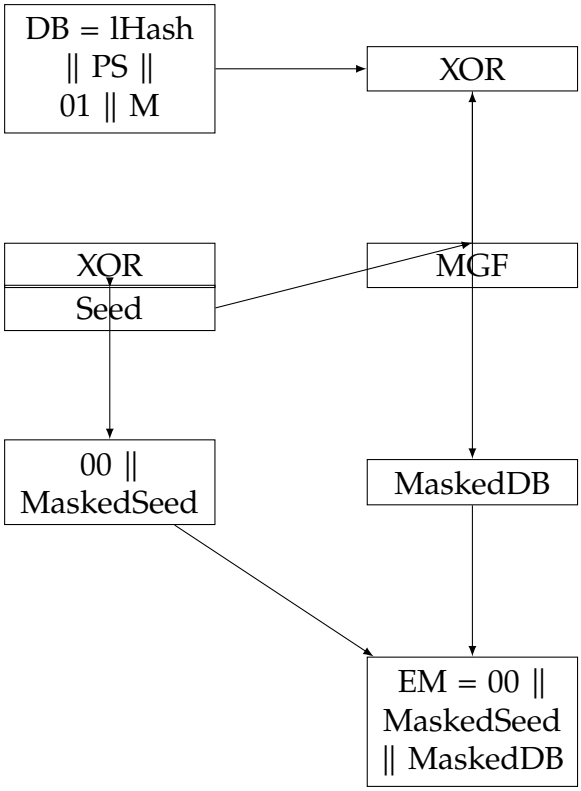


Figure 8.2: RSA-OAEP Encoding Operation

Algorithm 16: RSA-OAEP: OAEPEncode

Data: An plaintext p, an optional label L (default = `NULL`)
Result: An encoded plaintext $\hat{p}$ or `error`

procedure OAEPEncode(p, L)**:**
  $k, m, h \leftarrow$; // $N = pq$
  if $m > k - 2h - 2$ **then**
    return `error`;
  end
  /* $\underbrace{\texttt{DB}}_{k-h-1} = \underbrace{\texttt{lHash}}_{h} \| \underbrace{\texttt{PS}}_{k-2h-2-m} \| \underbrace{\texttt{0x01}}_{1} \| \underbrace{p}_{m}$ */
  Construct *DB***:**
    $\texttt{lHash} \leftarrow (L) \in \{0,1\}^{8h}$;
    $\texttt{PS} \leftarrow \texttt{0x00}^{(k-2h-2-m)}$;
    $\texttt{DB} \leftarrow (\texttt{lHash}, \texttt{PS}, \texttt{0x01}, p) \in \{0,1\}^{8h} \times \{0,1\}^{8(k-2h-2-m)} \times \{0,1\}^{8} \times \{0,1\}^{8m}$
  end
  $\texttt{seed} \xleftarrow{\$} \{0,1\}^{8h}$;
  $\texttt{maskedDB} \leftarrow$;
end

if $\text{Proj}_2(X) \neq \text{Proj}_2(X')$ **then**
  $t_1 \leftarrow \left(\text{Proj}_1(X) - \text{Proj}_1(X)\right)$;
  $t_2 \leftarrow \left(\text{Proj}_2(X') - \text{Proj}_2(X')\right)^{-1}$; // EEA
  return $t_1 t_2 \bmod p$
else
  return `Failure`;
end

8.2 RSA-FDH

8.3 RSA-PSS

Chapter 9

Primality Tests

9.1 Distribution of Primes

Prime Number Theorem

Theorem 9.1. *The prime-counting function $\pi(n)$, that is, the number of primes less than n, satisfies:*

$$\lim_{n\to\infty} \pi(n)\frac{\ln n}{n} = 1 \quad \left(\text{i.e., } \pi(n) \sim \frac{n}{\ln n}\right).$$

Remark 9.1.

n	512	1024	1536	2048	3840	7680
$\pi(2^n) - \pi(2^{n-1})$	$2^{502.53}$	$2^{1013.53}$	$2^{1524.94}$	$2^{2036.53}$	$2^{3827.62}$	$2^{7666.62}$

Code 9.1: Prime-Counting Function (Sage)

```
def pi (n) : return n / ln(n)
N = [512, 1024, 1536, 2048, 3840, 7680]
for n in N :
    U, L = 2^n, 2^(n-1)
    T = pi(U) - pi(L)
    print (n, log(T,2).n())

'''
512 502.525940328866
1024 1013.52735542414
1536 1524.94286369947
2048 2036.52806141567
3840 3827.62149992868
7680 7666.62168788972
'''
```

9.2 Classification of Primality Tests

Primality tests is classified into the following two types:

1. **Deterministic Primality Tests** (return provable prime):

   These are algorithms that determine with certainty whether a number is prime or not. For every input, they will always return the same output. An example of a deterministic primality test is the AKS primality test, which operates in polynomial time.

   A deterministic primality test, like the AKS, proceeds by eliminating every possible way that the number could be composite (not prime). This means that if the number passes the test, it is definitely a prime number.

   Pros:

   They always provide the correct answer. They are useful when absolute certainty is needed.

   Cons:

   They are usually slower than probabilistic tests, especially for large numbers

2. **Probabilistic Primality Tests** (return probable prime):

   These are algorithms that determine whether a number is probably prime, with a certain level of confidence. For large numbers, these tests are often much faster than deterministic tests. Examples of probabilistic primality tests include the Miller-Rabin and the Solovay-Strassen tests.

   A probabilistic test, like Miller-Rabin, works by randomly selecting values and performing a series of computations. If the number fails any of these computations, it is definitely composite. But if it passes, it is only "probably" prime. However, by performing the test multiple times with different randomly selected values, the probability that a composite number will be mistakenly identified as prime (called a "false positive") can be made arbitrarily small.

   Pros:

   They are often much faster than deterministic tests, especially for large numbers. They can provide a "probably prime" answer that is good enough for most practical purposes.

   Cons:

   They may provide false positives, although the probability can be made very small.

In short, the choice between deterministic and probabilistic primality tests often comes down to a trade-off between speed and certainty.

9.3 Deterministic Primality Test

9.3.1 Naive Test

Let $\mathbb{P}$ represent the set of all prime numbers.

Lemma 9.2. *Every integer greater than* 1 *has a prime divisor. That is,*

$$\forall n \in \mathbb{Z}_{>1} : \exists p \in \mathbb{P} : p \mid n.$$

Proof. Define a set S by

$$S := \{n \in \mathbb{Z}_{>1} : [\forall p \in \mathbb{P} : p \nmid n]\} \subseteq \mathbb{N}_{>1}.$$

Then

$$\begin{aligned}
S \neq \emptyset &\implies \exists \min S =: m \quad \text{by well-ordering principle} \\
&\implies \exists a, b \in (1, m) : m = ab \quad \because \forall p \in \mathbb{P} : p \notin S, \text{ i.e., } m \text{ is composite} \\
&\implies a \notin S \quad \because a < m = \min S \\
&\implies \exists p \in \mathbb{P} : p \mid a \\
&\implies p \mid m \in S \quad \lightning.
\end{aligned}$$

Thus S is empty and so every integer $n \in \mathbb{Z}_{>1}$ has a prime divisor. □

Proposition 9.3. *A number $n \in \mathbb{Z}_{>1}$ is composite if and only if there exists a prime divisor p such that $p \leq \sqrt{n}$. That is,*

$$\exists k \in \mathbb{N} \setminus \{1, n\} : k \mid n \iff \exists p \in \mathbb{P}_{\leq \sqrt{n}} : p \mid n.$$

Proof. Let $n \in \mathbb{Z}_{>1}$.

($\Leftarrow$) Let $p \in \mathbb{P}_{\leq \sqrt{n}}$ such that $p \mid n$. We show that $p \in \mathbb{N} \setminus \{1, n\}$:

$$p \in \mathbb{P}_{\leq \sqrt{n}} \implies 1 < p \leq \sqrt{n} < n \implies p \in \mathbb{N} \setminus \{1, n\}.$$

($\Rightarrow$) Let $\exists a \in \mathbb{N} \setminus \{1, n\} : a \mid n$ then $\exists b \in \mathbb{Z}$ such that $n = ab$ with $1 < a \leq b < n$. We claim that $a \leq \sqrt{n}$: suppose that $a > \sqrt{n}$ then

$$n = ab > \sqrt{n}\sqrt{n} = n \quad \lightning.$$

Thus $a \leq \sqrt{n}$. Since $a \in \mathbb{Z}_{>1}$, a has a prime divisor, that is, $\exists p \in \mathbb{P} : p \mid a$. Thus

$$p \mid a \Rightarrow p \leq a \Rightarrow p \leq a \leq \sqrt{n} \quad \text{and} \quad p \mid a \Rightarrow p \mid ab = n.$$

□

Trivial Division

Corollary 9.3.1. *The integer $n \in \mathbb{Z}_{>1}$ is prime iff $p \nmid n$ for any prime $p \leq \sqrt{n}$. That is,*

$$[\forall k \in \mathbb{Z}_{>0} : (k \mid n) \Rightarrow (k \in \{1, n\})] \iff \left[p \in \mathbb{P}_{\leq \sqrt{n}} \Rightarrow p \nmid n\right].$$

Algorithm 17: Naive Primality Test

Data: $n \in \mathbb{Z}_{>1}$
Result: Composite or Prime

```
1  if n = 2 then
2  |  return Prime;
3  end
4  if 2 | n then
5  |  return Compostie;
6  end
7  for j ← 1 to ⌊√n/2⌋ do
   |  /* 2⌊√n/2⌋ + 1 ≤ √n + 1 */
8  |  if (2j + 1) | n then
9  |  |  return Composite;
10 |  end
11 end
12 return Prime;
```

Data: Odd $n \in \mathbb{Z}_{>1}$
Result: Composite or Prime

```
1  for j ← 1 to ⌊√n/2⌋ do
   |  /* gcd(n, 2j + 1) ≠ 1 */
2  |  if n mod 2j + 1 = 0 then
3  |  |  return Composite;
4  |  end
5  end
6  return Prime;
```

9.3.2 Wilson Theorem

Wilson Theorem

Theorem 9.4. *Let $n \in \mathbb{Z}_{>1}$. Then*

$$[\forall k \in \mathbb{Z}_{>0} : (k \mid n) \Rightarrow (k \in \{1, n\})] \iff (n-1)! \equiv -1 \pmod{n}.$$

Proof. Let $n \in \mathbb{Z}_{>1}$.

($\Leftarrow$) Let $(n-1)! \equiv -1 \pmod{n}$. Suppose that n is a composite, that is, $n = ab$ where $1 < a \leq b < n$. Note that

$$a \in (1, n) \implies a \in \{2, 3, \cdots, n-1\} \implies a \mid (n-1)!.$$

Then

$$\begin{aligned}
(n-1)! \equiv -1 \pmod{n} &\implies n \mid (n-1)! + 1 \\
&\implies a \mid (n-1)! + 1 \quad \because n = ab \\
&\implies a \mid 1 \quad \because a \mid (n-1)! \\
&\implies a \leq 1 \quad \lightning.
\end{aligned}$$

Thus, n is a prime.

($\Rightarrow$) Let n is a prime. Consider a finite field $\mathbb{F}_n = \{0, 1, \cdots, n-1\}$. We want to find $a \in \mathbb{F}_n$ such that $a = a^{-1}$, that is, solve the equation $a^2 - 1 = 0$ over $\mathbb{F}_n$:

$$a = \pm 1 \implies \begin{cases} a = 1 \equiv 1 \pmod{n}, \\ a = -1 \equiv n-1 \pmod{n}. \end{cases}$$

Thus

$$\begin{aligned} (n-1)! = \prod_{a \in \mathbb{F}_n^*} a &= \left(\prod_{\substack{a \in \mathbb{F}_n^* \\ a = a^{-1}}} a \right) \left(\prod_{\substack{a \in \mathbb{F}_n^* \\ a \neq a^{-1}}} a \right) \\ &= (1 \times (n-1)) \underbrace{(1 \times \cdots \times 1)}_{\frac{n-3}{2} \text{ times}} \\ &= n - 1 \\ &\equiv -1 \pmod{n}. \end{aligned}$$

□

9.4 Probabilistic Primality Test

$$\Pr\left[\mathbb{P}^C \leftarrow T \mid n \in \mathbb{P}\right] = 0$$
$$\Pr\left[\mathbb{P} \leftarrow T \mid n \in \mathbb{P}^C\right] \geq 0$$

9.4.1 Fermat Primality Test: Application of FLT

Fermat's Little Theorem

Theorem 9.5. *Let $p \in \mathbb{P}$, and let $a \in \mathbb{Z}_{>0}$ such that $\gcd(p, a) = 1$. Then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Remark 9.2.

$$n \in \mathbb{P} \implies \forall a \in [1, n-1] : a^{n-1} \equiv 1 \pmod{n}$$
$$\exists a \in [1, n-1] : a^{n-1} \not\equiv 1 \pmod{n} \implies n \in \mathbb{P}^C.$$

Algorithm 18: Fermat Primality Test

Data: odd n and witness k
Result: Composite or Probable Prime

```
while k > 0 do
    k ← k − 1;
    a ←$ [2, n − 2];
    if gcd(a, n) ≠ 1 then
        return Composite;
    end
    if a^{n−1} ≢ 1 (mod n) then
        return Composite;
    end
end
return Probable Prime;
```

Remark 9.3. $1^{n-1} \equiv 1 \pmod{n}$ and

$$\begin{aligned}(n-1)^{n-1} &= \underbrace{(n-1)(n-1)\cdots(n-1)}_{n-1 \text{ times}} \\ &\equiv \underbrace{(-1)(-1)\cdots(-1)}_{n-1 \text{ times}} \pmod{n} \\ &= (-1)^{n-1} \\ &= -1 \quad \because n-1 \in 2\mathbb{Z}\end{aligned}$$

Remark 9.4 (Carmichael Number)**.**

$$\forall a \in \mathbb{Z} : a^n \equiv a \pmod{n}$$

9.4.2 Miller-Rabin Primality Test

Proposition 9.6. *Let $p \in \mathbb{P} \setminus \{2\}$ be an odd prime and $a \in \{2, \cdots, p-1\}$. Then*

(1) $a^q \equiv 1 \pmod{p}$, *or*

(2) $\bigvee_{i=0}^{l-1} \left[a^{2^i q} \equiv -1 \pmod{p} \right]$,

where $l, q \in \mathbb{Z}$ such that $p = 2^l q + 1 = q \ll l + 1$ and $2 \nmid q$.

Proof. By Fermat's Little Theorem, we have

$$0 \equiv a^{p-1} - 1 = a^{2^l q} - 1 \pmod{p}.$$

Then $a^{2^l q} - 1 = (a^{2^{l-1} q})^2 - 1^2$ is factorized as follows:

$$\begin{aligned} a^{2^l q} - 1 &= \left(a^{2^{l-1} q} + 1\right)\left(a^{2^{l-1} q} - 1\right) \\ &= \left(a^{2^{l-1} q} + 1\right)\left(a^{2^{l-2} q} + 1\right)\left(a^{2^{l-2} q} - 1\right) \\ &\vdots \\ &= \left(a^{2^{l-1} q} + 1\right)\left(a^{2^{l-2} q} + 1\right)\left(a^{2^{l-3} q} + 1\right) \cdots \left(a^{2q} + 1\right)\left(a^q + 1\right)\left(a^q - 1\right). \end{aligned}$$

Since $a^{2^l q} - 1 \equiv 0 \pmod{p}$, we obtain (1) and (2). □

Miller-Rabin's Test

Corollary 9.6.1. *Let an odd integer $n = 2^l q + 1 \in \mathbb{Z}_{\geq 3}$ satisfies the followings:*

1. $a^q \not\equiv 1 \pmod{p}$, *and*
2. $\bigwedge_{i=0}^{l-1} \left[a^{2^i q} \not\equiv -1 \pmod{p} \right]$.

Then n is composite.

Algorithm 19: Miller-Rabin Primality Test

Data: odd n and witness k
Result: Composite or Probable Prime

Compute q and l such that $n = 2^l q + 1$, where q is odd;
while $k > 0$ **do**
    $k \leftarrow k - 1$;
    $a \xleftarrow{\$} [2, n-2]$;
    if $\gcd(a, n) \neq 1$ **then**
        return Composite;
    end
    $a \leftarrow a^q \bmod n$;
    if $a = 1$ **then**
        continue to next iteration of loop;
    end
    for $j \leftarrow 0$ **to** $l - 1$ **do**
        if $a = n - 1$ **then**
            continue to next iteration of outer loop;
        end
        $a \leftarrow a^2 \bmod n$;
    end
    return composite;
end
return Probable Prime;

Algorithm 20: Miller-Rabin Primality Test

Input: a positive integer $n > 3$ and accuracy parameter k
Output: composite if n is composite, probably prime otherwise

for $i \leftarrow 1$ **to** k **do**
    Choose a random integer a in the range $[2, n-2]$;
    $x \leftarrow a^d \bmod n$;
    if $x = 1$ ***or*** $x = n - 1$ **then**
        continue to next iteration of loop;
    end
    for $r \leftarrow 1$ **to** $s - 1$ **do**
        $x \leftarrow x^2 \bmod n$;
        if $x = 1$ **then**
            return composite;
        end
        if $x = n - 1$ **then**
            continue to next iteration of outer loop;
        end
    end
    return composite;
end
return probably prime;

Code 9.2: Miller-Rabin Primality Test (Sage)

```
def get_l(a):
    cnt = 0
    while true:
        if (a&1) == 0:
            a, cnt = a >> 1 , cnt + 1
        else:
            return cnt

def is_composite(n, q, l, a):
    a = power_mod(a, q, n)
    print("a^(q) = {}".format(a))
    if(a%n) == 1:
        return "NOT Composite"
    for j in [0.. l -1]:
        print ("a^(2^{}*q) = {}".format(j,a))
        if(a%n) == n-1:
            return "NOT Composite"
        a = (a^2 % n)
    return "Composite"

def is_prime_by_miller_rabin(n, k):
    l = get_l (n -1)
    q = (n-1) >> l
    print("l = {}, q = {}".format(l,q))
    while k > 0:
        k = k - 1
        a = ZZ.random_element(2, n-2)
        print ("a = {}".format(a))
        ret = is_composite(n, q, l, a)
        if ret == "Composite":
            return "Composite"
        return "Probable Prime"

n = ZZ.random_element(2^40)
# n = random_prime(10000)
print ("n = {}".format(n))
print (is_prime_by_miller_rabin(n,10))
print (n.is_prime())

'''
n = 1080059945557
l = 2, q = 270014986389
a = 1033463656440
a^(q) = 1080059945556
a^(2^0*q) = 1080059945556
Probable Prime
True
'''
```

Code 9.3: Miller-Rabin Primality Test (C)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to compute (base^exponent) % modulus
unsigned long long modPow(unsigned long long base, unsigned long long exponent, unsigned long long modulus) {
    base %= modulus;
    unsigned long long result = 1;
    while (exponent > 0) {
        if (exponent & 1) result = (result * base) % modulus;
        base = (base * base) % modulus;
        exponent >>= 1;
    }
    return result;
}

// Function to perform the Miller-Rabin test for a given number (n) and a random witness (a)
int millerTest(unsigned long long d, unsigned long long n) {
    // Generate a random number [2, n-2].. rand() % (n-4) + 2 effectively does this
    srand(time(0));
    unsigned long long a = rand() % (n - 4) + 2;
    unsigned long long x = modPow(a, d, n);

    if (x == 1 || x == n - 1) return 1;

    // Repeat the squaring step until one of the following occurs:
    // d becomes n-1,
    // (x^2) % n becomes 1
    // (x^2) % n becomes n-1
    while (d != n - 1) {
        x = (x * x) % n;
        d *= 2;

        if (x == 1) return 0;
        if (x == n - 1) return 1;
    }

    // Return composite
    return 0;
}

// Function to perform the Miller-Rabin primality test for accuracy of k
int isPrime(unsigned long long n, int k) {
    // Corner cases
    if (n <= 1 || n == 4) return 0;
    if (n <= 3) return 1;

    // Reduce n-1 to (2^r)*d
    unsigned long long d = n - 1;
    while (d % 2 == 0) d /= 2;

    // Perform k tests
```

```
53      while (k--) if (!millerTest(d, n)) return 0;
54
55      // Return probably prime
56      return 1;
57  }
58
59  int main() {
60      unsigned long long n = 561;  // Carmichael number
61      int k = 5;  // Number of iterations
62
63      if (isPrime(n, k)) printf("%llu is probably prime.\n", n);
64      else printf("%llu is composite.\n", n);
65
66      return 0;
67  }
```

Proposition 9.7. *The Miller-Rabin primality test T, which repeats k trials, has the following properties:*

(1) $\Pr[\mathbb{P} \leftarrow T \mid n \in \mathbb{P}] = 1.$

(2) $\Pr[\mathbb{P}^C \leftarrow T \mid n \in \mathbb{P}] = 0,$

(3) $\Pr[\mathbb{P} \leftarrow T \mid n \in \mathbb{P}^C] \leq \frac{1}{4^k},$

(4) $\Pr[\mathbb{P}^C \leftarrow T \mid n \in \mathbb{P}^C] \geq 1 - \frac{1}{4^k}.$

Proof. content... □

Chapter 10

Elliptic Curves

10.1 Projective Variety

Note. (**Affine Space**; Associativity Axiom) Let K be a field, and let $(V, +_V, \circ)$ be a vector space over K. Let $\mathbb{A}$ be a set on which two mappings are defined

$$+ : \mathbb{A} \times V \to \mathbb{A}$$
$$- : \mathbb{A} \times \mathbb{A} \to V$$

satisfying the following associativity conditions:

(A1) $\forall a, b \in \mathbb{A} : a + (b - a) = a$

(A2) $\forall a \in \mathbb{A} : \forall u, v \in V : (a + u) + v = p + (u +_V v)$

(A3) $\forall a, b \in \mathbb{A} : \forall u \in V : (a - b) +_V u = (a + u) - b$. Then the ordered triple $(\mathbb{A}, +, -)$ is an **affine space**.

The affine space of demension n over field K is given by

$$\mathbb{A}^n := \mathbb{A}^n(K) = \{(x_1, \cdots, x_n) : x_i \in K \text{ for } i \in \{1, \cdots, n\}\}$$

10.1.1 Projective Line

Projective Line

Definition 10.1. Let K be a field.

(1) In the 2-dimensional vector space $V = K^2$ defined over the field K, a 1-dimensional subspace is referred to as a **projective line**.

(2) The set $\mathbb{P}^1(K)$ of all projective lines is called a 1-*dimensional projective space*.

$$\mathbb{P}^1(K) = \left\{S \subseteq V = K^2 : \dim S = 1\right\}$$

(3) An element of a projective space is referred to as a *point*.

Remark 10.1 (Notation)**.** The 1-dimensional subspace S of V can be defined as follows:

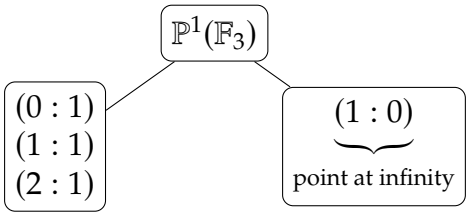
$$S = \{(\lambda a, \lambda b) : \lambda \in K\} \subseteq V$$

for some $(a, b) \setminus \{(0, 0)\}$. Then S is denoted as $S = (a : b)$. That is,

$$\mathbb{P}^1(K) = \{(a : b) : a, b \in K \setminus \{0\}\}$$

Example 10.1. Consider $\mathbb{P}^1(\mathbb{F}_3) = \mathbb{P}^1(\{0, 1, 2\})$. Then

$$\begin{cases} (0:1) \\ (0:2) \\ (1:0) \\ (1:1) \\ (1:2) \\ (2:0) \\ (2:1) \\ (2:2) \end{cases} \Longrightarrow \begin{cases} (0:1) \\ (1:0) \\ (1:1) \\ (2:1) \end{cases}$$



Remark 10.2. The point $S = (a : b)$ of $\mathbb{P}^1(K)$ can be denoted as follows

$$S = (a : b) = \begin{cases} b^{-1}(a : b) = (b^{-1}a : 1), & : b \neq 0 \\ (1 : 0) & : b = 0. \end{cases}$$

Proposition 10.1.

(1) $\mathbb{P}^1(K)$ *can be defined as follows:*

$$\mathbb{P}^1(K) := \{(c : 1) : c \in K\} \cup \{(1 : 0)\}\,.$$

where $c = b^{-1}a \in K$*. In this context,* $(1 : 0)$*, also denoted as* ∞*, is called the* ***point at infinity****.*

(2) When $|K| = p$*, the number of points in* $\mathbb{P}^1(K)$ *is* $\left|\mathbb{P}^1(K)\right| = p + 1$*, among which there is one point at infinity.*

(3) The points $(c : 1)$ *and* $(1 : 0)$ *can each correspond to the first-degree homogeneous polynomials* $X - cY = 0$ *and* $Y = 0$ *defined on* $\mathbb{P}^1(K)$*, respectively. That is, every element of* $\mathbb{P}^1(K)$ *can be expressed as* $aX + bY = 0$ *with* $a, b \in K \setminus \{0\}$

10.1.2 Projective Plane

Projective Plane

Definition 10.2. Let K be a field.

(1) In the 3-dimensional vector space $V = K^3$ defined over the field K, a 1-dimensional subspace is referred to as a **projective plane**.

(2) The set $\mathbb{P}^2(K)$ of all projective planes is called a 2-*dimensional projective space*.

$$\mathbb{P}^2(K) = \left\{S \subseteq V = K^3 : \dim S = 1\right\}$$

(3) An element of a projective space is referred to as a *point*.

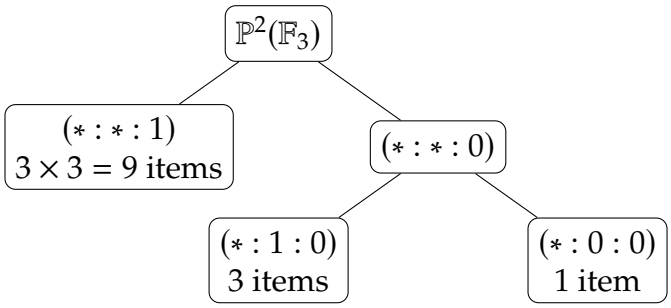
Remark 10.3 (Notation)**.** The 1-dimensional subspace S of V can be defined as follows:

$$S = \{(\lambda a, \lambda b, \lambda c) : \lambda \in K\} \subseteq V$$

for some $(a, b, c) \setminus \{(0, 0, 0)\}$. Then S is denoted as $S = (a : b : c)$. That is,

$$\mathbb{P}^2(K) = \{(a : b : c) : a, b, c \in K \setminus \{0\}\}.$$

Example 10.2. Consider $\mathbb{P}^2(\mathbb{F}_3) = \mathbb{P}^2(\{0, 1, 2\})$. Then



Remark 10.4. The point $S = (a : b : c)$ of $\mathbb{P}^2(K)$ can be denoted as follows

$$S = (a : b : c) = \begin{cases} (c^{-1}a : c^{-1}b : 1), & : c \neq 0 \\ (a : b : 0) & : c = 0. \end{cases}$$

Proposition 10.2.

(1) $\mathbb{P}^2(K)$ *can be defined as follows:*

$$\mathbb{P}^2(K) := \{(u : v : 1) : u, v \in K\} \cup \{(c : 1 : 0) : c \in K\} \cup \{(1 : 0 : 0)\}$$

where $u = c^{-1}a \in K$ *and* $v = c^{-1}b \in K$.

(2) When $|K| = p$, *the number of points in* $\mathbb{P}^2(K)$ *is* $\left|\mathbb{P}^2(K)\right| = p^2 + p + 1$, *among which there are* $p + 1$ *points at infinity.*

(3) (Generalization of #points) When $|K| = p$, *the number of points in* $\mathbb{P}^n(K)$ *is*

$$|\mathbb{P}^n(K)| = p^n + p^{n-1} + \cdots + p^2 + p + 1 = \sum_{i=0}^{n} p^i,$$

among which there are $\sum_{i=0}^{n-1} p^i$ *points at infinity.*

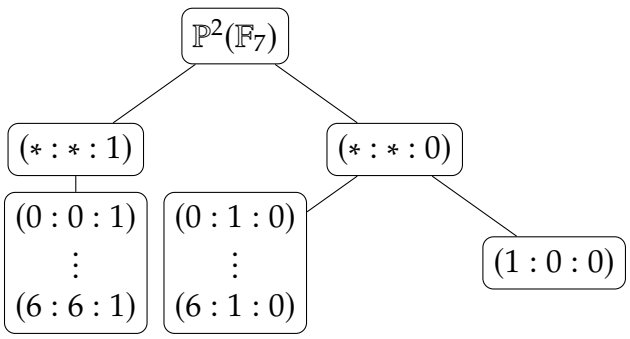
Remark 10.5. In $\mathbb{P}^2(K)$, points of type $(a : b : 0)$ are referred to as **points at infinity**, and the set $\{(c : 1 : 0) : c \in K\} \cup \{(1 : 0 : 0)\}$ of points at infinity can be defined by the linear equation $Z = 0$, therefore it is called the **line at infinity**.

Proof. (1) Let $u = c^{-1}a \in K$ and $v = c^{-1}b \in K$ then

$$\begin{aligned}
&\mathbb{P}^2(K) \\
&:= \{(u : v : 1) : u, v \in K\} \cup \{(a : b : 0) : a, b \in K\} \\
&= \{(u : v : 1) : u, v \in K\} \cup \left(\bigcup_{b \in K \setminus \{0\}} \{(a : b : 0) : a \in K\} \right) \cup \{(a : 0 : 0) : a \in K\} \\
&= \{(u : v : 1) : u, v \in K\} \cup \left(\bigcup_{b \in K \setminus \{0\}} \left\{(b^{-1}a : 1 : 0) : a \in K\right\} \right) \cup \{(1 : 0 : 0)\} \\
&= \{(u : v : 1) : u, v \in K\} \cup \{(c : 1 : 0) : c \in K\} \cup \{(1 : 0 : 0)\}.
\end{aligned}$$

□

Example 10.3. Consider $\mathbb{P}^2(\mathbb{F}_7)$.



10.1.3 Line

Line

Definition 10.3. On the projective plane $\mathbb{P}^2(K)$, the set of points satisfying the following homogeneous linear equation is called a **line**:

$$\alpha X + \beta Y + \gamma Z = 0, \quad \alpha, \beta, \gamma \in K \setminus \{0\}.$$

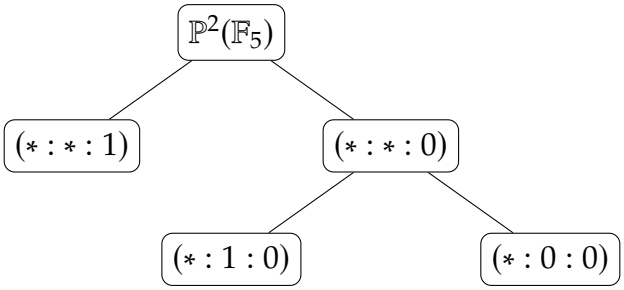
Therefore, the line $L(\alpha, \beta, \gamma)$ can be expressed as follows:

$$L(\alpha, \beta, \gamma) := \left\{(a : b : c) \in \mathbb{P}^2(K) : \alpha a + \beta b + \gamma c = 0\right\}.$$

Example 10.4. Find the points on the following line defined in $\mathbb{P}^2(\mathbb{F}_5)$.

$$f_1 : 2X+3Y+Z=0, \quad f_2 : 2X+2Y+3Z=0, \quad f_3 : 3X+Y+3Z=0, \quad f_4 : Z=0.$$

Sol.



(f_1) $2X + 3Y + Z = 0$;

$$\begin{cases} 2X + 3Y = 0 & : Z = 0 \quad \cdots\cdots \text{(i)} \\ 2X + 3Y + 1 = 0 & : Z = 1 \quad \cdots\cdots \text{(ii)} \end{cases}$$

(i)

$$\begin{aligned} 2X + Y &= 0 \\ 6X + 9Y &= 0 \\ X + 4Y &= 0 \\ X - Y &= 0 \\ X &= Y \end{aligned}$$

(ii)

$$\begin{aligned} 2X + Y + 1 &= 0 \\ 6X + 9Y + 3 &= 0 \\ X + 4Y + 3 &= 0 \\ X - Y + 3 &= 0 \\ Y &= X + 3 \end{aligned}$$

Therefore,

$$(1:1:0) \quad \text{and} \quad \begin{cases} (0:3:1) \\ (1:4:1) \\ (2:0:1) \\ (3:1:1) \\ (4:2:1). \end{cases}$$

(f_2) $2X + 2Y + 3Z = 0$;

$$\begin{cases} 2X + 2Y = 0 & : Z = 0 \quad \cdots\cdots \text{(i)} \\ 2X + 2Y + 3 = 0 & : Z = 1 \quad \cdots\cdots \text{(ii)} \end{cases}$$

(i)

$$\begin{aligned} 2X + 2Y &= 0 \\ 6X + 6Y &= 0 \\ X + Y &= 0 \\ X &= -Y \\ X &= 4Y \end{aligned}$$

(ii)

$$\begin{aligned} 2X + 2Y + 3 &= 0 \\ 6X + 6Y + 9 &= 0 \\ X + Y + 4 &= 0 \\ Y &= -X - 4 \\ Y &= 4X + 1 \end{aligned}$$

Therefore,

$$(4:1:0) \quad \text{and} \quad \begin{cases} (0:1:1) \\ (1:0:1) \\ (2:4:1) \\ (3:3:1) \\ (4:2:1). \end{cases}$$

(f_3) $3X + Y + 3Z = 0$;

$$\begin{cases} 3X + Y = 0 & : Z = 0 \quad \cdots\cdots \text{(i)} \\ 3X + Y + 3 = 0 & : Z = 1 \quad \cdots\cdots \text{(ii)} \end{cases}$$

(i)

$$\begin{aligned} 3X + Y &= 0 \\ 6X + 2Y &= 0 \\ X + 2Y &= 0 \\ X &= 3Y \end{aligned}$$

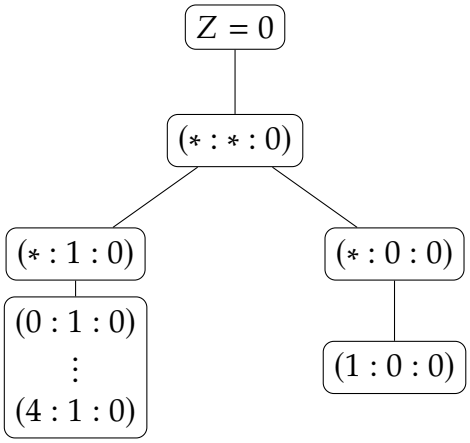
(ii)

$$\begin{aligned} 3X + Y + 3 &= 0 \\ Y &= -3X - 3 \\ Y &= 2X + 2 \\ Y &= 2(X + 1) \end{aligned}$$

Therefore,

$$(3:1:0) \quad \text{and} \quad \begin{cases} (0:2:1) \\ (1:4:1) \\ (2:1:1) \\ (3:3:1) \\ (4:0:1). \end{cases}$$

(f_4) $Z = 0$;



Therefore,

$$(1:0:0) \quad \text{and} \quad \begin{cases} (0:1:0) \\ (1:1:0) \\ (2:1:0) \\ (3:1:0) \\ (4:1:0). \end{cases}$$

□

Proposition 10.3. *Let $|K| = p$. Then*

(1) Every line in the projective plane $\mathbb{P}^2(K)$ contains $\frac{p^2-1}{p-1} = p + 1$ points.

(2) Any two distinct lines in the projective plane $\mathbb{P}^2(K)$ intersect at exactly one point.

Proof. (1)

(2)

□

Proposition 10.4. *The line passing through any two points $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$ in the projective plane $\mathbb{P}^2(K)$ is defined as follows:*

$$(y_2z_1 - y_1z_2)X + (z_2x_1 - z_1x_2)Y + (x_2y_1 - x_1y_2)Z = 0.$$

and is unique.

10.2 Plane Projective Curve

Plane Projective Curve

Definition 10.4. The **projective plane curve** (simply projective curve) C defined over the field K is determined by the homogeneous polynomial

$$F(X,Y,Z) \in K[X,Y,Z] : F(X,Y,Z) = 0.$$

Remark 10.6 (Affine to Projective). content...

Remark 10.7 (Projective to Affine). content...

Note.

Affine Curve	Projective Curve	Point at Infinity
$Y = aX + b$	$Y = aX + bZ$	$(1 : a : 0)$
$X^2 - Y^2 = 1$	$X^2 - Y^2 = Z^2$	$(1 : \pm 1 : 0)$
$X^2 + Y^2 = 1$	$X^2 + Y^2 = Z^2$	$\{(X,Y) : X^2 + Y^2 = 0\}$
$Y^2 = X^3 + aX + b$	$Y^2Z = X^3 + aXZ^2 + bZ^3$	$(0 : 1 : 0)$
$X^2 + Y^2 = 1 - X^2Y^2$	$X^2Z^2 + Y^2Z^2 = Z^4 - X^2Y^2$	$(1 : 0 : 0), (0 : 1 : 0)$

10.3 Singularity

10.4 Genus

10.5 Elliptic Curves

Elliptic Curve

Definition 10.5. In $\mathbb{P}^2(K)$, an algebraic curve of genus 1 that is non-singular is referred to as an **elliptic curve**.

Theorem 10.5. *All elliptic curves are isomorphic to a Weierstrass algebraic curve in the projective plane* $\mathbb{P}^2(\overline{K})$.

Note.

Name	Equation	Discriminant
Short Weierstrass	$Y^2Z = X^3 + aXZ^2 + bZ^3$	$-16(4a^3 + 27b^2)$
Montgomery	$BY^2Z = X^3 + AXZ^2 + XZ^3$	$B(A^2 - 4)$
Edwards	$X^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$	$d(d-1)$
Hessian	$X^3 + Y^3 + Z^3 = dXYZ$	-

10.6 Weierstrass Curve

Weierstrass Curve

Definition 10.6. The following cubic algebraic curve

$$E : F(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3 = 0,$$

with $(a_j \in K)$, defined over $\mathbb{P}^2(K)$, is referred to as a **Weierstrass algebraic curve**.

Note.

$$\begin{aligned}
a_jXYZ &\implies j + 2 + 3 + 0 = 6 &&\implies j = 1,\\
a_jYZ^2 &\implies j + 3 + 0 = 3 &&\implies j = 3,\\
a_jX^2Z &\implies j + 2^2 + 0 = 2 &&\implies j = 2,\\
a_jXZ^2 &\implies j + 2 + 0^2 = 4 &&\implies j = 4,\\
a_jZ^3 &\implies j + 0^3 = 6 &&\implies j = 6.
\end{aligned}$$

10.6.1 Non-Singularity

10.6.2 Weierstrass Equations in terms of Field Characteristic

Chapter 11

Elliptic Curve Group