Cryptography

Part 2

By: Killua4564

Index

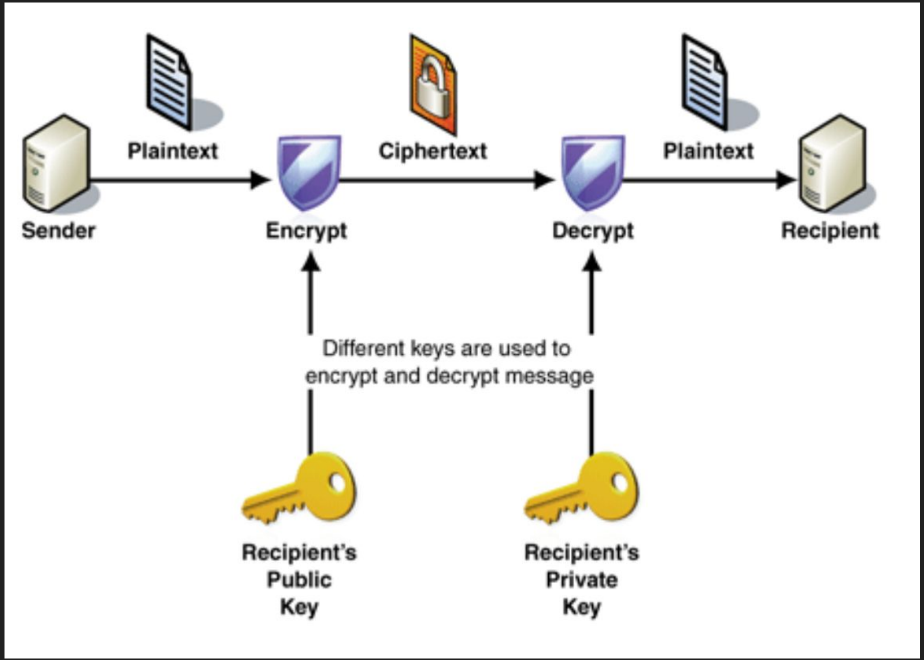
- Public-key cryptography
  - RSA
  - Diffie-Hellman
- Number theory
  - RSA
    - Euler's totient function
    - Fermat's little theorem
    - Euclidean algorithm
  - Discrete logarithm
  - Quadratic residue

Public-key cryptography

RSA

Public-key cryptography

- 公開金鑰密碼學
  - 又稱非對稱式密碼學
- 有公鑰和私鑰
  - 公鑰是公開的
  - 私鑰是不能公開的
  - 公鑰加密只能私鑰解密
  - 私鑰簽章只能公鑰驗證
- 應用途徑
  - TLS/SSL 連線加密
  - 金鑰交換 (key exchange)
  - 數位簽章 (Digital Signature)



RSA

- 由三個人名字縮寫組成
  - Ron Rivest (羅納德 李維斯特)
  - Adi Shamir (阿迪 薩莫爾)
  - Leonard Adleman (倫納德 阿德曼)
- 安全性
  - 因數分解問題 (keysize = 2048 ~ 4096 bits)
- 安全認證
  - PKCS#1
  - ANSI X9.31
  - IEEE 1363

RSA

- 金鑰生成
  - 選擇質數因子
    - p, q ∈ Prime
  - 計算模數 n 和 φ
    - $n = pq$, $\varphi(n) = (p - 1) * (q - 1)$
  - 選擇公鑰指數 e
    - $e \in N$, $1 < e < \varphi(n)$, $\gcd(e, \varphi(n)) = 1$
  - 計算私鑰指數 d
    - $d = \mathrm{inverse}(e, \varphi(n))$
- 金鑰組成
  - 公鑰 = (e, n)
  - 私鑰 = (d, n)

RSA

- 金鑰組成
  - 公鑰 = (e, n)
  - 私鑰 = (d, n)
- 加解密流程
  - $c = m^e \bmod n$
  - $m = c^d \bmod n$
- 簽章驗證流程
  - $sig = H(m)^d \bmod n$
  - $H(m) = sig^e \bmod n$
- 正確性
  - $c^d \bmod n \rightarrow m^{ed \bmod \varphi(n)} \bmod n \rightarrow m \bmod n \rightarrow m$
  - $sig^e \bmod n \rightarrow H(m)^{ed \bmod \varphi(n)} \bmod n \rightarrow H(m) \bmod n \rightarrow H(m)$

Factor modulus

- when $p == q$
  - $n = pq = p^2$
  - $\varphi(n) = p^2 - p$
- twin prime
  - $n1 = pq$
  - $n2 = (p + 2) * (q + 2) = pq + 2(p + q) + 4 = n1 + 2(p + q) + 4$
  - $\rightarrow p + q = (n2 - n1 - 4) / 2$
  - $\varphi(n1) = (p - 1) * (q - 1) = pq - (p + q) + 1 = n1 - (p + q) + 1$
  - $\varphi(n2) = (p + 1) * (q + 1) = pq + (p + q) + 1 = n1 + (p + q) + 1$
- Common factor attack
  - p, q reuse
  - $\gcd(n_i, n_j) = p_i = p_j$

- http://factordb.com/index.php
- https://github.com/bbuhrow/yafu
- https://www.alpertron.com.ar/ECM.HTM

Fermat's factorization method

- 當 $|p - q|$ 很小的時候使用
- 推導
  - $a = (p + q) / 2$
  - $b = (p - q) / 2$
  - $n = (a + b) * (a - b) = a^2 - b^2 \approx a^2$
  - $a \approx \mathrm{sqrt}(n)$
- 實作
  - 把 a 用 sqrt(n) 代入並迭代
  - 測試到 $a^2 - n$ 為完全平方數
  - $b = \mathrm{sqrt}(a^2 - n)$
  - $(p, q) = (a + b, a - b)$

```python
import gmpy2

def fermat(n: int) -> tuple[int, int]:
    a = gmpy2.isqrt(n) + 1
    b = a ** 2 - n
    while not gmpy2.iroot(b, 2)[1]:
        a += 1
        b = a ** 2 - n
    b = gmpy2.iroot(b, 2)[0]
    return (a + b, a - b)
```

Pollard's p-1 algorithm

- 當 p - 1 光滑 (smooth) 時使用
  - p - 1 的質因數分解有很多元素
- 費馬小定理
  - $a^{p-1} \equiv 1 \pmod p \ (p|n)$
  - $\rightarrow a^{k(p-1)} \equiv 1 \pmod p \ (k \in N)$
  - $p|\gcd(a^{k(p-1)} - 1, n)$
- 實作
  - 找一個 $b \in N$ 且 $k(p - 1)|b$
  - 則 $\gcd(a^b - 1, n)$ 有可能算出來 p
  - 讓 b 用階層去迭代可以完全覆蓋到 $k(p - 1)$
  - 通常讓 a=2 運算起來比較快

```
961636823328237596348218824232909922601
376289738418409379676317994599355901125
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000001
```

```python
from Crypto.Util.number import GCD

def pollard(n: int) -> int:
    a, b = 2, 2
    while True:
        a = pow(a, b, n)
        p = GCD(a - 1, n)
        if 1 < p < n:
            return p
        b += 1
```

Williams's p+1 algorithm

```python
B = B or int(gmpy2.isqrt(n))
for A in tqdm.tqdm(gen_prime(), disable=True):
    v = A
    for i in tqdm.trange(1, B + 1, desc=f"A={A}"):
        v = mlucas(v, i)
        g = GCD(v - 2, n)
        if g > 1:
            return g
```

- 當 p + 1 光滑 (smooth) 時使用
    - p - 1 光滑時也可以使用
    - 但計算 p - 1 光滑會比 Pollard 慢
- 設 B = p + 1 質因數分解的最大質數
    - 則設 $M = \{\forall x, x \in \text{Prime}, x \leq B\}$ 的積，滿足 $p+1|M$
- 但 M 非常大且 $M \nmid n$，所以借助盧卡斯數列 (Lucas sequence)
    - 由計算很大的 M 改成計算 $V_M(A, 1)$ 去分解 n
    - 在 i = 1, 2, 3, ..., M 的情況下做 $\gcd(V_i(A, 1) - 2, n)$ 去嘗試分解 n
- 實作細節
    - A 通常選擇不是 2 的質數，選合數容易重複跑嘗試過的 A
    - 考慮到 p + 1 可能會有重根多因子，所以 M 會直接用 B! 去跑

Williams's p+1 algorithm - Lucas sequence

- 盧卡斯數列有兩類
  - P, Q ∈ Z
  - $U_0(P,Q) = 0$ $U_1(P,Q) = 1$ $U_n(P,Q) = P \cdot U_{n-1}(P,Q) - Q \cdot U_{n-2}(P,Q)\ ,\ n > 1$
  - $V_0(P,Q) = 2$ $V_1(P,Q) = P$ $V_n(P,Q) = P \cdot V_{n-1}(P,Q) - Q \cdot V_{n-2}(P,Q)\ ,\ n > 1$
- 表達式
  - 由遞歸式得知特徵方程為 $x^2 - Px + Q = 0$
    - 得 λ = (P ± sqrt(P^2 - 4Q)) / 2
  - 設兩根為 a, b 且判別式為 D，則得
    - U_n(P, Q) = (a^n - b^n) / sqrt(D)
    - V_n(P, Q) = a^n + b^n

Williams's p+1 algorithm - Lucas sequence

- 盧卡斯數列
  - $U_n(P, Q) = (a^n - b^n) / \mathrm{sqrt}(D)$
  - $V_n(P, Q) = a^n + b^n$
- V 的特性
  - $V_{m+n} = V_m V_n - Q^n V_{m-n}$
  - $V_{2n} = V_n^2 - 2Q^n$
  - $V_{mn}(P, Q) = V_m(V_n(P, Q), Q^n)$
- 特殊名稱
  - $U_n(1, -1)$: 斐波那契數
  - $V_n(1, -1)$: 盧卡斯數
  - $U_n(2, -1)$: 佩爾數

$$U_0(P,Q) = 0 \qquad U_1(P,Q) = 1 \qquad U_n(P,Q) = P \cdot U_{n-1}(P,Q) - Q \cdot U_{n-2}(P,Q)\ ,\ n > 1$$

$$V_0(P,Q) = 2 \qquad V_1(P,Q) = P \qquad V_n(P,Q) = P \cdot V_{n-1}(P,Q) - Q \cdot V_{n-2}(P,Q)\ ,\ n > 1$$

```python
def mlucas(b: int, k: int) -> int:
    # It returns k-th V(b, 1)
    v1, v2 = b % n, (b ** 2 - 2) % n
    for bit in bin(k)[3:]:
        if int(bit):
            v1, v2 = (v1 * v2 - b) % n, (v2 ** 2 - 2) % n
        else:
            v1, v2 = (v1 ** 2 - 2) % n, (v1 * v2 - b) % n
    return v1
```

Common modulus attack

- 共模攻擊
- 相同 m, n 不同 e 產生不同 c
  - $m^{e1} \bmod n = c1$
  - $m^{e2} \bmod n = c2$
- 若滿足 gcd(e1, e2) = 1，則有線性方程滿足 s1 * e1 + s2 * e2 = 1
  - 注意 (s1, s2) 為方程上一組解 (egcd)
    - s1 = inverse(e1, e2)
    - s2 = (1 - s1 * e1) / e2
- $c1^{s1} * c2^{s2} \bmod n$
  - $\rightarrow m^{e1*s1} * m^{e2*s2} \bmod n$
  - $\rightarrow m^{e1*s1+e2*s2} \bmod n$
  - $\rightarrow m$

Low public exponent attack

- 低指數公鑰攻擊、低加密指數攻擊
- e 很小的時候 (例如 e=3) 直接爆破出來 m
- $m^e \bmod n = c$
  - $\rightarrow m^3 \bmod n = c$
  - $\rightarrow m^3 = c + k * n$
- 窮舉一下 k 開 e 方根
- 目的
  - 為了節省加密和驗證的時間 $(F_x = 2^{2^x} + 1)$
  - 通常 e 會從 Fermat number 挑
    - 也通常挑 $F_0=3$, $F_2=17$, $F_4=65537$

Wiener's attack

- 維納攻擊，是種低解密指數攻擊
- 當 e 非常大導致 d 很小時用 (e, n) 直接推算 d
- 當 $d < 1/3 * n^{1/4}$ 和 $|p - q| < \min(p, q)$ 條件符合時可以利用 (e, n) 來估計 (d, φ(n))
- $ed \equiv 1 \pmod{\varphi(n)}$
  - $\rightarrow k \in N, ed = k * \varphi(n) + 1$
  - $\rightarrow e / \varphi(n) = k / d + 1 / (d * \varphi(n))$
  - $\rightarrow e / \varphi(n) \approx k / d$
  - $\rightarrow e / n \approx k / d$

Wiener's attack - Lemma 1

- 滿足 $|p - q| < \min(p, q)$
  - 也就是 $q < p < 2q$
- $n - \varphi(n)$
  - $\rightarrow n - (p - 1) * (q - 1)$
  - $\rightarrow n - pq + p + q - 1$
  - $\rightarrow p + q - 1 < 3\text{sqrt}(n)$
- 得到 $n - \varphi(n) < 3\text{sqrt}(n)$

Wiener's attack - Lemma 2

- 滿足 $d < 1/3 * n^{1/4}$
- $ed \equiv 1 \pmod{\varphi(n)}$
  - $\rightarrow ed = k * \varphi(n) + 1$
  - $\rightarrow k * \varphi(n) = ed - 1$
  - $\rightarrow k * \varphi(n) < ed$
  - $\rightarrow k * \varphi(n) < \varphi(n) * d$
  - $\rightarrow k < d < 1/3 * n^{1/4}$
  - $\rightarrow k < 1/3 * n^{1/4}$
- 得到 $k < 1/3 * n^{1/4}$

Wiener's attack - Lemma 3

- 滿足 $d < 1/3 * n^{1/4}$
  - $\rightarrow d < 1/3 * n^{1/4}$
  - $\rightarrow 3d < n^{1/4}$
  - $\rightarrow 2d < n^{1/4}$
  - $\rightarrow 1 / 2d > 1 / n^{1/4}$
- 得到 $1 / n^{1/4} < 1 / 2d$

Wiener's attack - proof

- Lemma 1: $n - \varphi(n) < 3\text{sqrt}(n)$
- Lemma 2: $k < 1/3 * n^{1/4}$
- Lemma 3: $1 / n^{1/4} < 1 / 2d$
- 計算 $|e / n - k / d|$
  - $\rightarrow |(ed - nk) / dn| = |(1 + k\varphi(n) - nk) / dn|$
  - $\rightarrow (k(n - \varphi(n)) - 1) / dn < (3k * \text{sqrt}(n) - 1) / dn < 3k * \text{sqrt}(n) / dn$
  - $\rightarrow 3k * \text{sqrt}(n) / dn < n^{3/4} / dn = 1 / dn^{1/4}$
  - $\rightarrow 1 / dn^{1/4} < 1 / 2d^2$
- 發現 e / n 和 k / d 相差小於 $1 / 2d^2$
  - 可利用 e / n 將 k / d 逼出來

Wiener's attack

- e / n ≈ k / d
- 連分數

$$\frac{e}{N} = \frac{17993}{90581} = \cfrac{1}{5 + \cfrac{1}{29 + \cdots + \cfrac{1}{3}}} = [0, 5, 29, 4, 1, 3, 2, 4, 3]$$

- 推算 (k, d)

$$\frac{k}{d} = 0, \frac{1}{5}, \frac{29}{146}, \frac{117}{589}, \frac{146}{735}, \frac{555}{2794}, \frac{1256}{6323}, \frac{5579}{28086}, \frac{17993}{90581}$$

Wiener's attack

- 最後 $\varphi(n)$ 呢？
  - $ed = k * \varphi(n) + 1$
  - $\rightarrow \varphi(n) = (ed - 1) / k$
- 用途 1 : 驗證 d
  - assert d == inverse(e, φ(n))
- 用途 2 : 分解 (p, q)
  - $\varphi(n) = (p - 1) * (q - 1) = pq - (p + q) + 1 = n - (p + q) + 1$
  - $\rightarrow p + q = n - \varphi(n) + 1$
  - 考慮方程 $x^2 - (p + q)x + pq = 0$
    - 則 x 的兩個解為 (p, q)

LSB Oracle attack

- 情境：給密文 c 回傳解密後明文 mod r 的結果
  - 假設 r = 2，則回傳明文的最後一個 bit
- 利用同態加密 (Homomorphic encryption) 的特性製成 $r^{ke}c \bmod n$ 送出
  - 如果 $0 \le m \le n/2$，則 $2m \bmod 2 = 0$
    - 如果 $0 \le m \le n/4$，則 $4m \bmod 2 = 0$
    - 如果 $n/4 \le m \le n/2$，則 $(4m - n) \bmod 2 = 1$
  - 如果 $n/2 \le m \le n$，則 $(2m - n) \bmod 2 = 1$
    - 如果 $n/2 \le m \le 3n/4$，則 $(4m - 2n) \bmod 2 = 0$
    - 如果 $3n/4 \le m \le n$，則 $(4m - 3n) \bmod 2 = 1$
  - 依此類推可以 oracle 出來原始的明文 m
    - 其中 $(r^k m - in) \bmod r$ 的值可以由 $(-in \bmod r)$ 得出

Bleichenbacher attack

- 又稱 million message attack
- 發生在 TLS 使用 PKCS#1 v1.5 版本的 RSA 公開金鑰交換
    - 其中 padding 會在 message 前面加上固定的 0x00 0x02 去做填充

00	02	padding string	00	data block

    - 而在 TLS 中如果 unpad 失敗會回覆 Bad data
    - 可以利用同態加密的特性做到 MSB Oracle
- 收到密文 c 之後遍歷每個 s，計算每個 $s^e c \bmod n$ 去嘗試 unpad
    - 如果 unpad 成功表示 $2B \le sm \bmod n < 3B$ (B 表示所有低位)
    - 就可以推出 $(2B + kn) / s \le m < (3B + kn) / s$
    - 雖然不知道 k 但可以遍歷所有 k 直到不滿足 $0 < m < n$
    - 把所有 s 中 k 的可能性做交集則可以擠出 m
- 後續影響：[DROWN attack]

Public-key cryptography

Diffie-Hellman

Diffie-Hellman

- 簡稱 D-H，是金鑰交換協定，發表於 RSA 之前
    - Whitfield Diffie (惠特菲爾德·迪菲)
    - Martin Hellman (馬丁·赫爾曼)
- 安全性
    - 離散對數問題
- 交換流程
    - 選定質數 p，並在 p 的循環群 G 下找出原根 g
        - 原根 (Primitive root) 是指 g 滿足 $\forall x < \varphi(p), g^x \bmod p \neq 1$
    - A, B 兩人各自有秘密 a, b
    - 各自計算 $ga = g^a \bmod p$, $gb = g^b \bmod p$ 傳給對方
    - 收到後再各自計算共享金鑰 $g^{ab} = ga^b = gb^a \pmod p$
- 容易受到中間人攻擊

ElGamal

- 基於 D-H 的加密和簽章系統，PGP 中有應用
- 金鑰生成
  - 選個生成元 g 產生 $p - 1$ 階的循環群 G (放在公鑰裡公開)
    - 想成有個原根 g 在模 $G(=n)$ 群裡面有 $p - 1(=\varphi(n))$ 階
  - 選擇密鑰 $x \in G$ 滿足 $1 < x < p - 1$，並計算公鑰 $y = g^x \bmod p$
- 加密流成
  - 選擇隨機數 $k \in G$, $1 < k < p - 1$，並計算 $c1 = g^k \bmod p$
  - 計算共享秘密 $s = y^k = g^{xk}$ 和加密明文 $c2 = ms = mg^{xk} \bmod p$
  - 做成密文 $(c1, c2)$
- 解密流程
  - 計算共享秘密 $s = c1^x = g^{xk} \bmod p$
  - 解密出明文 $c2 * s^{-1} = m * g^{xk} * g^{-xk} \bmod p = m$

ElGamal

- 基於 D-H 的加密和簽章系統，PGP 中有應用
- 金鑰生成
  - 選個生成元 g 產生 p - 1 階的循環群 G (放在公鑰裡公開)
    - 想成有個原根 g 在模 G(=n) 群裡面有 p - 1(=φ(n)) 階
  - 選擇密鑰 $x \in G$ 滿足 $1 < x < p - 1$，並計算公鑰 $y = g^x \bmod p$
- 簽章流成
  - 選擇隨機數 $k \in G, 1 < k < p - 1$，並計算 $r = g^k \bmod p$
  - 計算簽章 $s = (H(m) - xr) * k^{-1} \bmod \varphi(p) \neq 0$
  - 做成數位簽章 (r, s)
- 驗證流程
  - 計算正確的簽章 $g^{H(m)} \bmod p$
  - 比對 $y^r * r^s = g^{xr} * g^{k * (H(m) - xr) * k\text{^}-1} = g^{H(m)} \bmod p$

ElGamal

- 缺陷
  - 密文中的 c2 可以乘上任何倍數 k 讓解密出來直接變 km mod q
    - 可以做 Homomorphic 和 oracle
  - 如果有 (m1, m2) 滿足 $H(m1) \equiv H(m2) \pmod{\varphi(p)}$
    - 則兩者簽章會相同
  - 如果簽章不做雜湊，可以偽造簽章，但 m 不可控
    - 選擇隨機數對 $(e, v) \in G$, $1 \le (e, v) < p - 1$，滿足 $\gcd(v, \varphi(p)) = 1$
    - $r = g^e y^v \bmod p$，$s = -rv^{-1} \bmod \varphi(p)$ 則 $m = es \bmod \varphi(p)$
  - 同理如果不做雜湊且 (m1, m2) 滿足 $m1 \equiv m2 \pmod{\varphi(p)}$
    - 兩者簽章也會相同

DSA (Digital Signature Algorithm)

- 數位簽章算法
- 由 ElGamel 變體而來
- 聯邦資訊處理標準 (FIPS) 之一
- 用於數位簽章標準 (DSS)
- 提供資訊鑑定 (message authentication) 的性質
  - 完整性 (integrity)：接收方可以驗證訊息是完整、未被修改的
  - 不可否認性 (non-repudiation)：傳送方不能否認他們簽署的訊息
- 參數選擇
  - H 使用 SHA2，若 bits 數太多則採用高位，長度為 N
  - 金鑰長度建議 2048 bits，必須為 64 的倍數，長度為 L
  - (L, N) 需符合 (1024, 160), (2048, 224), (2048, 256), (3072, 256) 其中一種

DSA (Digital Signature Algorithm)

- 參數選擇
  - H 使用 SHA2，若 bits 數太多則採用高位，長度為 N
  - 金鑰長度建議 2048 bits，必須為 64 的倍數，長度為 L
  - (L, N) 需符合 (1024, 160), (2048, 224), (2048, 256), (3072, 256) 其中一種
- 系統建置
  - 選擇 N bits 質數 q
  - 選擇 L bits 質數 p，滿足 $q|p-1$
  - 選擇 h 滿足 $1 < h < p-1$ (通常 $h=2$)
  - 計算 $g = h^{(p-1)/q} \bmod p \neq 1$
  - 則演算法系統公用參數為 (p, q, g)
- 金鑰生成
  - 選擇私鑰 x 滿足 $1 < x < q-1$
  - 計算公鑰 $y = g^x \bmod p$ 並公開

DSA (Digital Signature Algorithm)

- 金鑰生成
  - 選擇私鑰 x 滿足 $1 < x < q-1$
  - 計算公鑰 $y = g^x \bmod p$ 並公開
- 簽章流程
  - 選擇隨機數 k 滿足 $1 < k < q-1$
  - 計算 $r = (g^k \bmod p) \bmod q \neq 0$
  - 計算 $s = k^{-1}(H(m) + xr) \bmod q \neq 0$
  - 做成簽章 (r, s)
- 驗證流程
  - 計算 $u1 = H(m) * s^{-1} \bmod q = kH(m) * (H(m) + xr)^{-1} \bmod q$
  - 計算 $u2 = rs^{-1} \bmod q = kr * (H(m) + xr)^{-1} \bmod q$
  - 比對 $(g^{u1}y^{u2} \bmod p) \bmod q = (g^{k * (H(m) + xr) * (H(m) + xr)\text{^}-1} \bmod p) \bmod q = r$

Reuse Nonce Attack

- 考慮簽章 (r, s)
  - $r = (g^k \bmod p) \bmod q$
  - $s = k^{-1}(H(m) + xr) \bmod q$
- 如果隨機數 k 被洩露，則私鑰 x 可被算出來
  - $x = r^{-1}(sk - H(m)) \bmod q$
- 當 k 重複用在兩個簽章時，k 就可被算出來
  - $s1 = k^{-1}(H(m1) + xr) \bmod q$
  - $s2 = k^{-1}(H(m2) + xr) \bmod q$
  - $\rightarrow s1 - s2 = k^{-1}(H(m1) - H(m2)) \bmod q$
  - $\rightarrow k = (H(m1) - H(m2)) * (s1 - s2)^{-1} \bmod q$
- 同理 ElGamal 也有這個缺陷

Number theory

RSA

Euler's totient function

$\varphi(n)$ = 小於 n 的正整數中跟 n 互質的數的個數

- $\varphi(1) = 1$
- $\varphi(2) = 1$
- $\varphi(3) = 2$
- $\varphi(4) = 2$
- $\varphi(5) = 4$
- $\varphi(6) = 2$
- $\varphi(7) = 6$
- $\varphi(8) = 4$
- $\varphi(9) = 6$
- $\varphi(10) = 4$
- $\varphi(11) = 10$

- $p \in$ Prime
  - $\varphi(p) = p - 1$
- $p \in$ Prime, $k \in N$
  - $\varphi(p^k) = p^k - p^{k-1}$
- $p, q \in$ Prime
  - $\varphi(pq) = \varphi(p) * \varphi(q) = (p-1) * (q-1)$

Fermat's little theorem

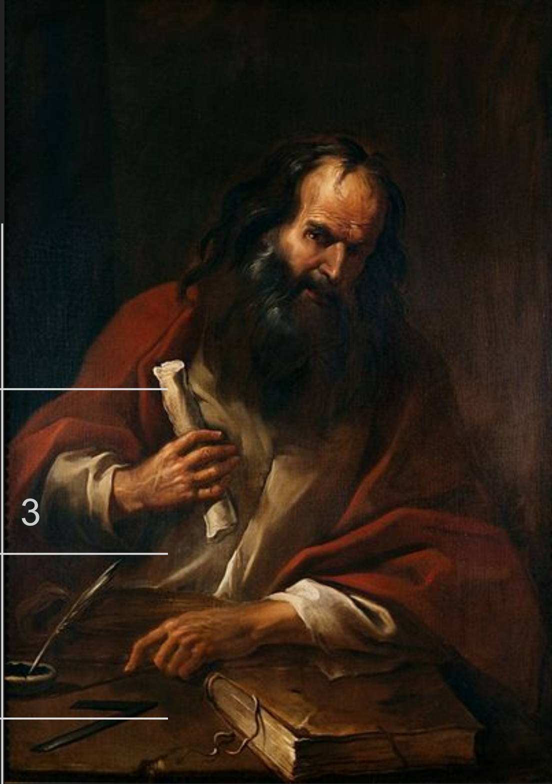
- 定理
  - $a \in N, p \in Prime, gcd(a, p) = 1$
  - $a^{p-1} \equiv 1 \pmod p$
- 證明
  - 設 $S = \{1, 2, 3, 4, \ldots, p - 1\}$
  - $S1 = \{\forall x \in S, ax \bmod p\}$
  - $S = Permutation(S1) = S1$
  - $S * a^{p-1} \equiv S \pmod p$
- 歐拉擴充
  - $a, n \in N, gcd(a, n) = 1$
  - $a^{\varphi(n)} \equiv 1 \pmod n$

Euclidean algorithm

- 147 = 1071 - 462 * 2
- 21 = 462 - 147 * 3
- → 21 = 462 - (1071 - 462 * 2) * 3
- → 21 = 462 * 7 - 1071 * 3

- 如果 a, b 互質 (gcd(a, b) = 1)
- ax + by = gcd(a, b) = 1
- → ax ≡ 1 (mod b)
- → x = inverse(a, b)
- → x = pow(a, -1, b)

	1071	462	
2	924		
	147	462	
		441	3
	147	21	
7	147		
	0	21	



extended Euclidean algorithm

```
def egcd(a: int, b: int) -> tuple[int, int, int]:
    if a == 0:
        return (b, 0, 1)
    g, y, x = egcd(b % a, a)
    return (g, x - (b // a) * y, y)


def inverse(a: int, b: int) -> int:
    g, x, y = egcd(a, b)  # ax + by = g
    if g == 1:
        return x % b
    raise ValueError("base is not invertible for the given modulus.")
```

Number theory

Discrete logarithm

Discrete logarithm

- 問題描述
  - 在模 p 的循環群 G 中找到一原根 g
  - 對於任意 x ∈ G 做 $h = g^x \bmod p$
  - 給定 (g, h, p) 求 x 的問題
- 安全性
  - 對於非量子電腦沒有 polynomial time 的算法
  - 在選定足夠安全的 G 前提下
    - 也不存在有效率的 exponential time 算法
- 實作
  - sympy.discrete_log
  - sage.groups.generic.discrete_log

CRT (Chinese Remainder Theorem)

- 又稱孫子定理，古稱韓信點兵、鬼谷算、物不知數
- 最早出自於「孫子算經」第二十六題「物不知數」
  - 有物不知其數，三三數之剩二，五五數之剩三，七七數之剩二。問物幾何？
- 問題
  - $x \equiv ri \pmod{ni}$ 給定多組 (ri, ni)，求 x
- 解法
  - 假設所有 ni 都是兩兩互質的
  - 定義 N 為 prod(ni) 且 Ni = N / ni
  - $x = \Sigma(ri * \text{inverse}(Ni, ni) * Ni) \bmod N$
- 證明
  - 因為 inverse(Ni, ni) * Ni 保證了模 ni 時餘 1 同時模其他 nj 時餘 0
  - 跟拉格朗日插值法 (Lagrange interpolating polynomial) 的原理類似

CRT (Chinese Remainder Theorem)

- 以物不知數舉例，由題目得知
  - $x \equiv 2 \pmod 3$
  - $x \equiv 3 \pmod 5$
  - $x \equiv 2 \pmod 7$
- 計算
  - N = 3 * 5 * 7 = 105
  - N1 = 5 * 7 = 35
  - N2 = 3 * 7 = 21
  - N3 = 3 * 5 = 15
- 求解
  - x = 2 * inverse(35, 3) * 35 + 3 * inverse(21, 5) * 21 + 2 * inverse(15, 7) * 15
  - x = 233 mod 105 = 23
- 實作
  - sage.arith.misc.CRT

```python
@functools.cache
def chinese_remainder(n: tuple[int], r: tuple[int]) -> int:
    result = 0
    prod = functools.reduce(lambda a, b: a * b, n)
    for ni, ri in zip(n, r):
        Ni = prod // ni
        result += ri * inverse(Ni, ni) * Ni
    return result % prod
```

Baby-step giant-step

- 給定 $h = g^x \bmod n$，求 x
- 一種中途相遇 (meet-in-the-middle) 算法
- 概念
  - 將 G 切成 sqrt(n) 份並把每份紀錄一個值
  - 遍歷 h 直到踩到有紀錄過的值，進而推算出 x
- 舉例
  - Giant-step
    - 對於每個 sqrt(n) 的倍數做紀錄
    - 儲存 $g^{k*sqrt(n)} \bmod n$, $0 \le k \le sqrt(n)$ 進 hashtable
  - Baby-step
    - 對 h 每次乘上 g 造成 g^{x+i} 的形式
    - 如果 g^{x+i} 在 hashtable 裡面，則 $x = k * sqrt(n) - i$

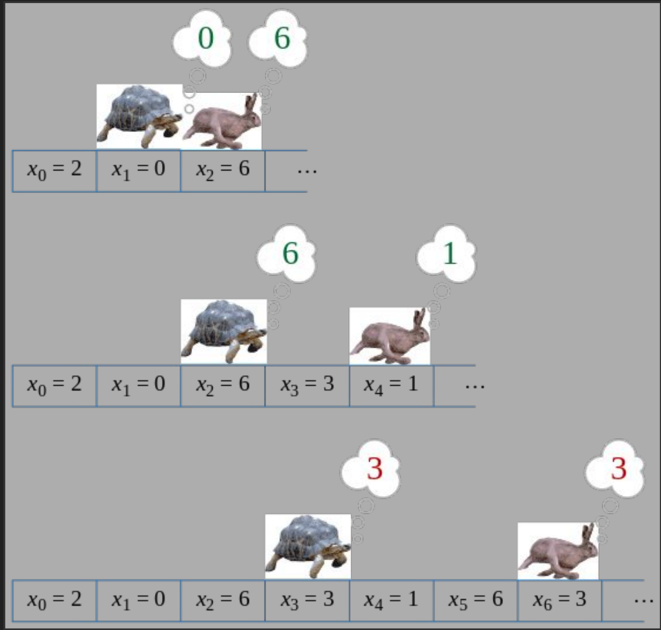
Baby-step giant-step

- giant: 紀錄每個 $g^{k*sqrt(n)}$ mod n
- baby: 走訪 $h * g^i = g^{x+i}$
- 相遇後回傳 x = k * sqrt(n) - i

```python
# 1. restore pow(g, k * sqrt(n), p)
# 2. match i with h * pow(g, i, p) % p
# 3. return x = (k * sqrt(n) - i) % n
def bsgs(g: int, h: int, n: int) -> int:
    giant: dict[int, int] = {}
    sqrt = int(gmpy2.isqrt(n)) + 1
    gs, gks = pow(g, sqrt, p), 1
    for k in tqdm.trange(sqrt, leave=False):
        giant[gks] = k
        gks = gks * gs % p
    for i in tqdm.trange(sqrt, leave=False):
        try:
            k = giant[h]
            return (k * sqrt - i) % n
        except KeyError:
            h = h * g % p
    raise ValueError(f"Can't find ord in subgroup {n}")
```

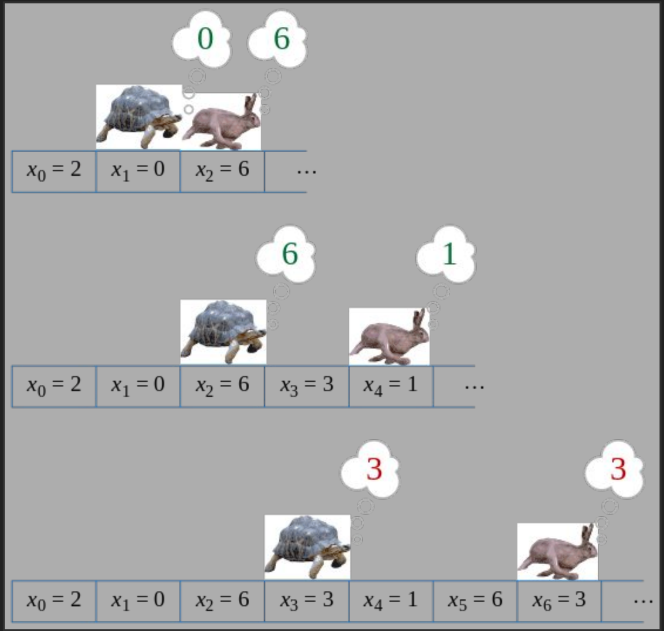
Pollard’s rho algorithm

- 給定 $h = g^x \bmod n$，求 x
- 利用循環偵測 (cycle detection) 的方法求解
  - 對於任意方程 f 有數列定義為 $x_i = f(x_{i-1})$
  - 則雖然 x 看似隨機，一但重複就會循環
- 概念
  - 如果對於 (g, h) 能找到一組 $g^a h^b = g^A h^B$
  - 則 $x = (A - a) * \text{inverse}(b - B, \varphi(n)) \bmod \varphi(n)$
- 使用 Floyd's cycle-finding algorithm 遍歷
  - 定義 tortoise 和 hare 形容去偵測 $x_i == x_{2i}$
- 定義偽隨機函數 $f: G \to G$
  - 讓 a, b 有對應的 $G * Z \to Z$ 函數



$$f(x) = \begin{cases} \beta x & x \in S_0 \\ x^2 & x \in S_1 \\ \alpha x & x \in S_2 \end{cases}$$

Pollard's rho algorithm



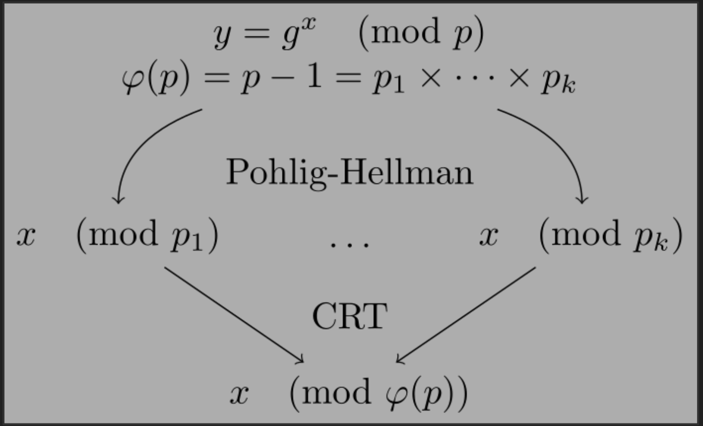
$$f(x) = \begin{cases} \beta x & x \in S_0 \\ x^2 & x \in S_1 \\ \alpha x & x \in S_2 \end{cases}$$

```python
# 1. check g == h to avoid b1 == b2
# 2. calculate until x_i == x_2i
def pollard_rho(g: int, h: int, n: int) -> int:
    def new(x: int, a: int, b: int) -> tuple[int]:
        if x % 3 == 1:
            return (g * x % p, (a + 1) % n, b)
        if x % 3 == 2:
            return (h * x % p, a, (b + 1) % n)
        return (x * x % p, 2 * a % n, 2 * b % n)

    if g == h:
        return 1

    x1, a1, b1 = 1, 0, 0
    x2, a2, b2 = 1, 0, 0
    for _ in tqdm.trange(n, leave=False):
        x1, a1, b1 = new(x1, a1, b1)
        x2, a2, b2 = new(x2, a2, b2)
        x2, a2, b2 = new(x2, a2, b2)
        if x1 == x2:
            r = inverse(b2 - b1, n) // GCD(b2 - b1, n)
            return (a1 - a2) * r % n
    raise ValueError(f"Can't find ord in subgroup {n}")
```

Pohlig-Hellman algorithm



- 給定 $y = g^x \bmod p$，求 x
- 發明者
  - Stephen Pohlig
  - Martin Hellman
- 概念
  - 因為 g 在模 p 的 G 上，所以 x 在模 φ(p) 的 G 上
  - 如果 φ(p) 可以被有效分解成子群 (subgroup)
  - 則可以在子群各自做離散對數得到因子
    - Baby-step Giant-step
    - Pollard’s rho algorithm
  - 之後再用 CRT 組回來解決 G 的問題

Pohlig-Hellman algorithm

- G 轉換成 pi subgroup
  - 要把 G 限縮成 pi 子群則要映射 g, h 到子群
    - $gi = g^{(p-1)/pi} \bmod p$
    - $hi = h^{(p-1)/pi} \bmod p$
  - 進而算出 pi 子群的結果 discrete_log(gi, hi, pi)
- G 轉換成 pi^k subgroup
  - 對於所有 pi^j $(0 \le j < k)$ 來說都是在 pi 子群裡面
    - $gi = g^{(p-1)/pi} \bmod p$
  - 但每個 j 會影響 pi^k 子群的結果，所以要遍歷 j 並把結果加權加總起來
  - 在計算每個 hj 時需要扣除之前累計的偏差
    - $hj = (g^{-xj}h)^{(p-1)/pi^{**}(j+1)}$
    - $x_{j+1} = xj + \text{discrete\_log}(gi, hj, pi) * pi^j$

Pohlig-Hellman algorithm

```python
def pohlig_hellman(g: int, h: int, p: int, factors: tuple[tuple[int, int]]) -> int:
    ords = []
    # if all factors are once power
    for pi, _ in factors:
        gi = pow(g, (p - 1) // pi, p)
        hi = pow(h, (p - 1) // pi, p)
        ords.append(bsgs(gi, hi, pi))
    # otherwise
    g_inv = inverse(g, p)
    for pi, power in factors:
        gi, xi = pow(g, (p - 1) // pi, p), 0
        for j in range(power):
            hi = h * pow(g_inv, xi, p) % p
            hj = pow(hi, (p - 1) // pi ** (j + 1), p)
            xi += bsgs(gi, hj, pi) * pi ** j
        ords.append(xi)
    # end if
    return chinese_remainder(tuple(i ** j for i, j in factors), tuple(ords))
```

Number theory

Quadratic residue

Quadratic residue

- 如果能找到一個整數 x 滿足 $x^2 \equiv a \pmod{p}$
  - 則 a 為模 p 的二次剩餘
- 相反的如果 a 找不到 x 來滿足
  - 則 a 為模 p 的二次非剩餘 (quadratic nonresidue)
- 因為 $x^2 \equiv (-x)^2 \equiv a \pmod{p}$
  - 所以在模 p 底下能滿足的 a 只會有 $(p + 1) / 2$ 個

Legendre symbol

```python
@functools.cache
def legendre_symbol(a: int, p: int) -> int:
    ls = pow(a, (p - 1) // 2, p)
    return -1 if ls == p - 1 else ls
```

- 描述 a 是否是在模 p 底下的二次剩餘
- $$\begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \text{ and } a \not\equiv 0 \pmod{p}, \\ -1 & \text{if } a \text{ is a non-quadratic residue modulo } p, \\ 0 & \text{if } a \equiv 0 \pmod{p}. \end{cases}$$
- 公式 : ls(a, p) = $a^{(p-1)/2}$ mod p
- 是個積性函數 (Multiplicative function)
  - ls(a, p) * ls(b, p) = ls(ab, p)

Tonelli-Shanks algorithm

- 用來找出模運算的平方根 (modular square root)
- 主要邏輯 (這邊 p 為質數，也適用於 p^k)
  - 用 legendre symbol 判斷是否 a 為二次剩餘
  - 找一個二次非剩餘的整數 z
  - 將 p - 1 拆成 $2^S Q$ 的形式，其中 Q 為奇數
  - 讓 $R = a^{(Q+1)/2}$, $t = a^Q$, $c = z^Q$, $M = S$ 進入迴圈
    - 因為 $R^2 = ta$，如果 t = 1 則回傳 R
    - 找一個最小整數 i 滿足 $0 < i < M$ 和 $t^{2^{**}i} = 1$
    - 計算 $b = c^{2^{**}(M-i-1)}$
    - 讓 $R = Rb$, $t = tb^2$, $c = b^2$, $M = i$
      - $R^2 = tab^2$

```python
z = 2
while legendre_symbol(z, p) != -1:
    z += 1

q, s = p - 1, 0
while q % 2 == 0:
    s += 1
    q //= 2

r = pow(a, (q + 1) // 2, p)
t = pow(a, q, p)
c = pow(z, q, p)
m = s
while True:
    if t == 1:
        return r

    i = 0
    while pow(t, 2 ** i, p) != 1:
        i += 1

    b = pow(c, 2 ** (m - i - 1), p)
    r = r * b % p
    t = t * pow(b, 2, p) % p
    c = pow(b, 2, p)
    m = i
```

Tonelli-Shanks algorithm

- 擴充加速
  - 判斷為二次剩餘後，滿足 $p \bmod 4 = 3$
  - 因為 $p \bmod 4 = 3$，所以 $p + 1$ 為 4 的倍數
    - $a^{(p-1)/2} = 1$
    - $\rightarrow a^{(p+1)/2} = a$
    - $\rightarrow (a^{(p+1)/4})^2 = a$
  - 則可以計算 $a^{(p+1)/4}$ 直接對 a 開模方

```
if p % 4 == 3:
    return pow(a, (p + 1) // 4, p)
```

Tonelli-Shanks algorithm

- n 為和數的情況
  - 對二次剩餘 a 開模方
  - 先將 n 質因數分解
  - 把 a 與因數做 Tonelli-Shanks
  - 之後再用 CRT 組起來
    - 每個 remainder 的正負值都要嘗試看看

```python
def power2roots(a: int, modulars: list[int], k: int):
    if k == 0:
        yield a
        return

    for i in power2roots(a, modulars, k - 1):
        remainders: tuple[int | None] = tuple(
            tonelli_shanks(i, p) for p in modulars
        )
        if not all(remainders):
            continue

        for remainder in itertools.product(*[
            (r, m - r) for m, r in zip(modulars, remainders)
        ]):
            yield chinese_remainder(modulars, remainder)
```