RSA

By: Killua4564

$ whoami

- Killua4564 / Cheng Yan
- NTUST MIS
- Crypto, Misc, Reverse
- https://killua4564.github.io/

RSA in Crypto

- 古典密碼學
  - 單表代換加密
  - 多表代換加密
  - 其他類型
- 現代密碼學
  - 對稱加密 / 區塊加密 (Block mode)
    - DES
    - AES
  - 非對稱加密 / 公開金鑰加密
    - RSA
    - Diffie–Hellman
    - ECC
  - 雜湊函數
- 編碼方式

Python Tools

- gmpy
- gmpy2
- owiener
- pycrypto / pycryptodome
- sagemath
- sympy
- tqdm

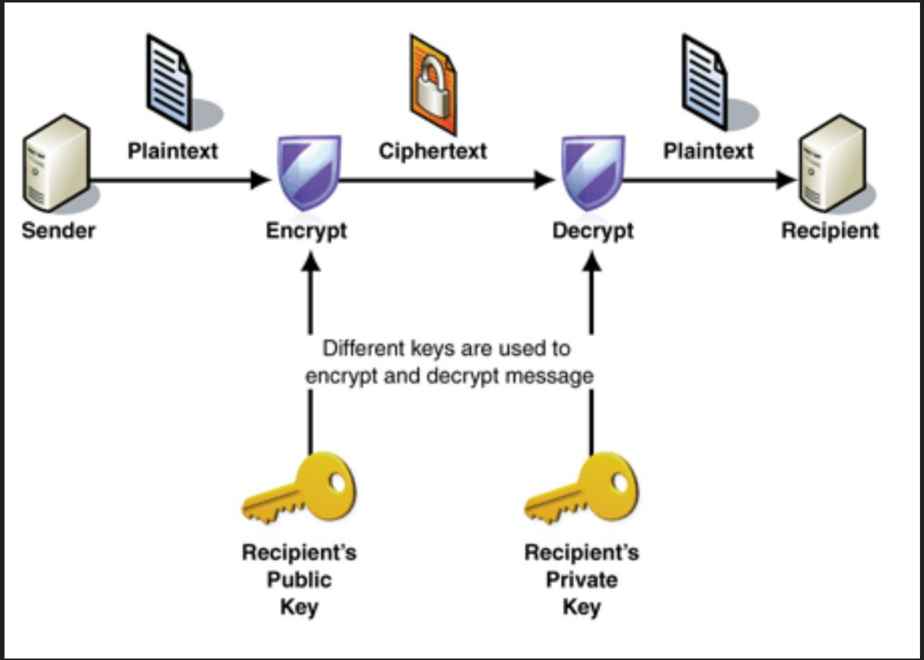
Introduction

非對稱加密演算法

公開金鑰加密系統

廣泛用於

- TLS/SSL 連線加密
- ssh 認證金鑰



Theory

明文 ** 公鑰 % 模數 = 密文

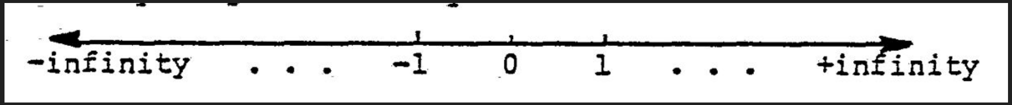
密文 ** 私鑰 % 模數 = 明文

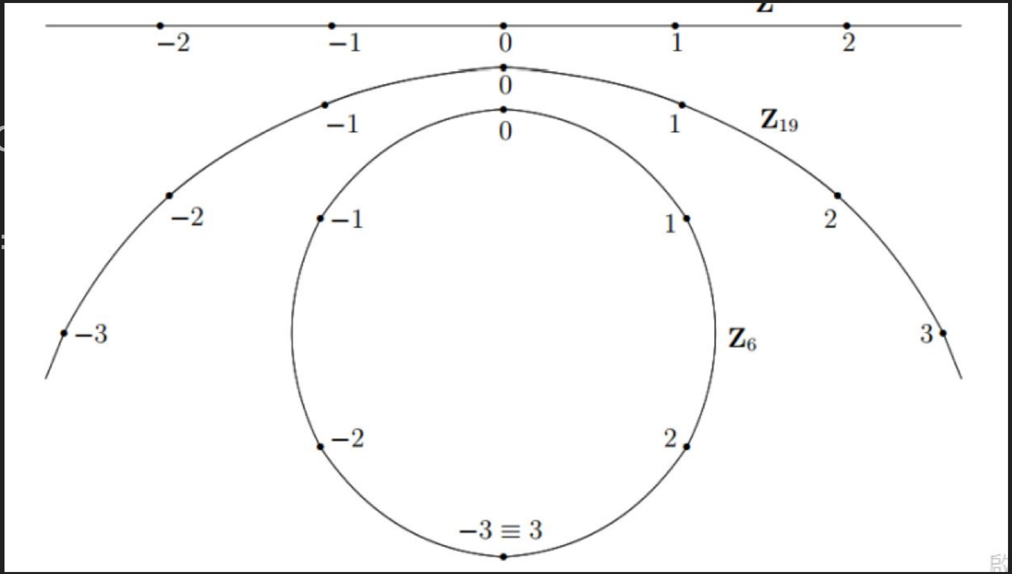
plaintext ** publicKey % modulus = c

ciphertext ** privateKey % modulus =

m ** e % n = c

c ** d % n = m





Theory

明文 ** 公鑰 % 模數 = 密文

密文 ** 私鑰 % 模數 = 明文

plaintext ** publicKey % modulus = ciphertext

ciphertext ** privateKey % modulus = plaintext

m ** e % n = c

c ** d % n = m

Theory

p, q = very big prime

$n = p * q$

$\varphi(n) = (p - 1) * (q - 1)$

選擇小於 $\varphi(n)$ 正整數 e, 並滿足 $\gcd(e, \varphi(n)) = 1$

$d = \text{invert}(e, \varphi(n))$, $ed \equiv 1 \pmod{\varphi(n)}$

m ** e % n = c pow(m, e, n)

c ** d % n = m pow(c, d, n)

Theorem (Euler's totient function)

φ(n)表示「在比 n 小的正整數中，跟 n 互值數字的個數」

$\varphi(1) = 1$

$\varphi(2) = 1$

$\varphi(3) = 2$

$\varphi(4) = 2$

$\varphi(5) = 4$

$\varphi(6) = 2$

$\varphi(7) = 6$

$\varphi(8) = 4$

$\varphi(9) = 6$

$\varphi(10) = 4$

$\varphi(11) = 10$

$p \in$ Prime, $\varphi(p) = p - 1$

$k \in$ N, $\varphi(p$ ** $k) = p$ ** $k - p$ ** $(k - 1)$

$q \in$ Prime, $\varphi(p * q) = \varphi(p) * \varphi(q) = (p - 1) * (q - 1)$

Theorem (Fermat's little theorem)

$a \in N, p \in \text{Prime}$

$a ** (p - 1) \% p = 1$

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

引入上一頁的歐拉函數

$\Rightarrow a ** \varphi(p) \equiv 1 \pmod{p}$

Theorem (Euclidean algorithm)

$147 = 1071 - 462 * 2$

$21 = 462 - 147 * 3$

$\Rightarrow 21 = 462 - (1071 - 462 * 2) * 3$

$\Rightarrow 21 = 462 * 7 - 1071 * 3$

$ax + by = \gcd(a, b)$

$\Rightarrow ax = \gcd(a, b) - by$

$\Rightarrow ax \equiv \gcd(a, b) \pmod{b}$

$ax \equiv \gcd(a, b) \equiv 1 \pmod{b}$

$\Rightarrow x = \mathrm{invert}(a, b)$

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

Theory

p, q = very big prime

$n = p * q$

$\varphi(n) = (p - 1) * (q - 1)$

選擇小於 $\varphi(n)$ 正整數 e, 並滿足 $\gcd(e, \varphi(n)) = 1$

$d = \text{invert}(e, \varphi(n))$, $ed \equiv 1 \pmod{\varphi(n)}$

m ** e % n = c　　　　pow(m, e, n)

c ** d % n = m　　　　pow(c, d, n)

Theory

n = p * q

d = invert(e, φ(n))

c = m ** e % n

c ** d % n

⇒ (m ** e) ** d % n

⇒ m ** ed % n

⇒ m ** (ed % φ(n)) % n

⇒ m ** 1 % n

⇒ m

Python Time

from Crypto.Util.number import long_to_bytes, inverse

- Invert
  - d = int(sympy.invert(e, phi))
  - d = gmpy2.invert(e, phi)
  - d = inverse(e, phi)
- 數字轉文字
  - long_to_bytes(m)
  - binascii.unhexlify(hex(m)[2:])
  - hex(m)[2:].decode("hex")

```python
from Crypto.PublicKey import RSA

data = """-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAn+lRm+vbe7IYXqXwlMHi
O/92HL5KImbnmExlGDl+vzlxvs1KgmEDDqERGRikyecnbWvBXfXxmPlY0VVha+Hl
0CJi6bqMl553TQ4PyRBPlMriFkW91mLdhgYmj2nKsMddJ4heQ64jqQuFMsKOAwZo
xEOcpKtjHF1/zr9fsi6TrHMw0yVNAhCfCiXNqz+Rh5483ZdeaxFDzGSHbqc+WSN6
sWHzc/etpQE+UpO/M4uIEc45drJzkRcy4MkY+zkVU5ma77/fzfGxcBFAcyWSevlj
n+GoVwJnYmPkKA6Rjj0wb8JjBtZ9nA4/24/mw7MnmgYc1QyJbo6WRs75+uX8IHH0
vQIDAQAB
-----END PUBLIC KEY-----"""

key = RSA.importKey(data)

e = key.e
n = key.n
```

Factor Attack

- when $p == q$

  $n = p * q = p ** 2$

  $\varphi(n) = p ** 2 - p$
- twin prime

  $n1 = p * q, \quad n2 = (p + 2) * (q + 2)$

  $\varphi(n1) = (p-1) * (q-1) = pq - (p + q) + 1 = n1 - (p + q) + 1$

  $\varphi(n2) = (p+1) * (q+1) = pq + (p + q) + 1 = n1 + (p + q) + 1$

  $n2 = (p+2) * (q+2) = pq + 2 * (p + q) + 4 = n1 + 2 * (p + q) + 4$

  $p + q = (n2 - n1 - 4) / 2$
- Common Factor Attack

  p, q reuse $\Rightarrow$ $gcd(ni, nj) = pi = pj$

Factor Attack (Pollard Algorithm)

使用時機 : 當 p-1 光滑(smooth)時

$a, b, n, k \in N, p \in Prime$
滿足 $gcd(a, p) = 1$ 和 $p \mid n$

根據費馬小定理
$\Rightarrow a ** (p - 1) \equiv 1 \pmod p$
$\Rightarrow a ** k(p - 1) \equiv 1 \pmod p$
$\Rightarrow p \mid gcd(a ** k(p - 1) - 1, n)$

若 $p - 1 \mid b$ 成立且滿足 $gcd(a ** b - 1, n) > 1$
則 $gcd(a ** b - 1, n) = p$
代 a=2 去窮舉一下很快會有結果 但此演算法不一定能成功

p=9132400715036908229752508016230
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000001

Factor Attack (Pollard Algorithm)

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

Factor Attack (Fermat's Factorization method)

使用時機: |p - q| 很小的時候

Let a = (p + q) / 2   b = (p - q) / 2

n = (a + b) * (a - b) = a ** 2 - b ** 2

因為 |p - q| 很小，所以 n 會略等於 a 的平方

把 a 用 sqrt(n) 代入，測試 a ** 2 - n 是否為平方數

若 a ** 2 - n 為平方數，則 (p, q) 為 (a + b, a - b)

Factor Attack (Fermat's Factorization method)

```python
def fermat(n):
    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)
```

Common Modulus Attack

共模攻擊
使用時機:相同明文、不同公鑰、相同餘數、有對應密文

m ** e1 % n = c1
m ** e2 % n = c2

若滿足 gcd(e1, e2) = 1, 則有線性方程滿足 s1e1 + s2e2 = 1
其中 s1 = invert(e1, e2) 且 s2 = invert(e2, e1)

c1 ** s1 * c2 ** s2
⇒ (m ** (e1 * s1)) * (m ** (e2 * s2)) % n
⇒ m ** (e1 * s1 + e2 * s2) % n
⇒ m

Common Modulus Attack

一切看似很完美...... 解出來根本不像flag…

```
ip149-185:share_modulus Killua4564$ python3 script.py
b"\x02[O\x16B=L\xf4\xcc\xc7\x93j@\x07\xed\\^\x99M\xbf\xaa\xf17i\x19\x19\xf0U\xcaa\xfd\t[\xb1^'\xdd\xcez\xa6\xcaVl8\x95H\xf7\xfe\x96\xc3=\x83\x8d\x
ec\xff\xc2\x0c\x1c\xb1m\xf4\xd2\xbau?\x8f\x8d\xc5\xe7\x0f\x07\x9ct \x9c\xd6\x05\xc1!\x03t\n\x91K\xe3H\x8e0\xb8\xa4\xdff}O\xfbkv27`P\x08M\x0f\xbf\x
19d{\xac\x9c\xcd3\x0c\x18\xb2^\xc5i\xd2yw\xa4\x03,|\x9bz\x86`&\xb8n\x89\x94g\x1b\xc8C[?&\x1b\xc0\xaf\xc6;\xcf}W\xfb,\x92-@\xfc\xfc\xbe!\x863\x1e\x
93\xaf\xe2\x0e\x05z\x9b\x9c~\xef\x9b\x9b\xed?Z)\x81\xb1\xf1\r1\x9f\x94k\x90E/c\xe8>\x8c\x18\x92)k\xf9\x9d\xfc[MMP\xe3}\xc0=\xc6)/cwT$\x89\x1a\x829
\xdd\x0bx\xc1\xdf\xee\x13\xb8PD\xa7\xfev\x9c\x13-\x81\xd5\xbbe-\xd0\xc0.\xf3\x0eK\x95\xb5Fq\xb3r|\xa4:u\xc7w"
```

注意到 (s1, s2)，他們必須是線性方程的”一組”解，所以分開算 invert 並不是一組的

所以算出 s1 後，因為 s1e1 + s2e2 = 1，所以 s2 = (1 - s1e1) / e2

Common Modulus Attack

一切看似很完美...... 解出來根本不像flag…

```
ip149-185:share_modulus Killua4564$ python3 script.py
b"\x02[O\x16B=L\xf4\xcc\xc7\x93j@\x07\xed\\^\x99M\xbf\xaa\xf17i\x19\x19\xf0U\xcaa\xfd\t[\xb1^'\xdd\xcez\xa6\xcaVl8\x95H\xf7\xfe\x96\xc3=\x83\x8d\x
ec\xff\xc2\x0c\x1c\xb1m\xf4\xd2\xbau?\x8f\x8d\xc5\xe7\x0f\x07\x9ct \x9c\xd6\x05\xc1!\x03t\n\x91K\xe3H\x8e0\xb8\xa4\xdff}O\xfbkv27`P\x08M\x0f\xbf\x
19d{\xac\x9c\xcd3\x0c\x18\xb2^\xc5i\xd2yw\xa4\x03,|\x9bz\x86`&\xb8n\x89\x94g\x1b\xc8C[?&\x1b\xc0\xaf\xc6;\xcf}W\xfb,\x92-@\xfc\xfc\xbe!\x863\x1e\x
93\xaf\xe2\x0e\x05z\x9b\x9c~\xef\x9b\x9b\xed?Z)\x81\xb1\xf1\r1\x9f\x94k\x90E/c\xe8>\x8c\x18\x92)k\xf9\x9d\xfc[MMP\xe3}\xc0=\xc6)/cwT$\x89\x1a\x829
\xdd\x0bx\xc1\xdf\xee\x13\xb8PD\xa7\xfev\x9c\x13-\x81\xd5\xbbe-\xd0\xc0.\xf3\x0eK\x95\xb5Fq\xb3r|\xa4:u\xc7w"
```

注意到 (s1, s2)，他們必須是線性方程的”一組”解，所以分開算 invert 並不是一組的

所以算出 s1 後，因為 s1e1 + s2e2 = 1，所以 s2 = (1 - s1e1) / e2

一切看似很完美...... python 的 pow 噴錯了...

ValueError: pow() 2nd argument cannot be negative when 3rd argument specified

若 c2 ** s2 ≡ x ** (-s2)   (mod n)

⇒ (c2 ** s2)(x ** s2) ≡ 1 (mod n)

⇒ (c2 * x) ** s2 ≡ 1   (mod n)

⇒ x = invert(c2, n)

Hastad’s Broadcast Attack

中國剩餘定理 (Chinese Remainder Theorem)
對加密的指數做攻擊
使用時機：e 固定不變，有數個 n 和對應的 c

典故：孫子算經 第26題 物不知數

	N = 3 * 5 * 7	
$x \equiv 2 \pmod 3$	N1 = 5 * 7 = 35	d1 = invert(N1, n1) = 2
$x \equiv 3 \pmod 5$	N2 = 3 * 7 = 21	d2 = invert(N2, n2) = 1
$x \equiv 2 \pmod 7$	N3 = 3 * 5 = 15	d3 = invert(N3, n3) = 1

x = (c1d1N1 + c2d2N2 + c3d3N3) % N
x = (140 + 63 + 30) % 105 = 233 % 105 = 23

Wiener's Attack

對解密指數做攻擊
使用時機:e非常大 d很小的時候

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 $(d, \varphi(n))$

$ed \equiv 1 \pmod{\varphi(n)}$

$\Rightarrow e * d = k * \varphi(n) + 1$ $(k \in N)$

$\Rightarrow e / \varphi(n) = k / d + 1 / (d * \varphi(n))$ (divide by $d * \varphi(n)$)

$\Rightarrow e / \varphi(n) \approx k / d$

$\Rightarrow e / n \approx k / d$

Wiener's Attack - Lemma 1

因為 $|p - q| < \min(p, q)$，所以可以說 $q < p < 2q$

$n - \varphi(n)$

$\Rightarrow n - (p - 1)(q - 1)$

$\Rightarrow n - pq + p + q - 1$

$\Rightarrow p + q - 1 < 3 * \mathrm{sqrt}(n)$

得到 $n - \varphi(n) < 3 * \mathrm{sqrt}(n)$

Wiener's Attack - Lemma 2

如果滿足 $d < (1/3) * n^{**}(1/4)$

$ed \equiv 1 \pmod{\varphi(n)}$

$\Rightarrow e * d = k * \varphi(n) + 1$

$\Rightarrow k * \varphi(n) = e * d - 1$

$\Rightarrow k * \varphi(n) < e * d$

$\Rightarrow k * \varphi(n) < \varphi(n) * d$

$\Rightarrow k < d < (1/3) * n^{**}(1/4)$

$\Rightarrow k < (1/3) * n^{**}(1/4)$

得到 $k < (1/3) * n^{**}(1/4)$

Wiener's Attack - Lemma 3

如果滿足 $d < (1/3) * n**(1/4)$

$d < (1/3) * n**(1/4)$
$\Rightarrow 3d < n**(1/4)$
$\Rightarrow 2d < n**(1/4)$
$\Rightarrow 1 / 2d > 1 / n**(1/4)$

得到 $1 / n**(1/4) < 1 / 2d$

Wiener's Attack - Proof

Lemma 1: $n - \varphi(n) < 3 * sqrt(n)$

Lemma 2: $k < (1/3) * n**(1/4)$

Lemma 3: $1 / n**(1/4) < 1 / 2d$

$|e / n - k / d| = |(ed - nk) / nd| = |(1 + k\varphi(n) - nk) / nd|$

$\Rightarrow (k(n - \varphi(n)) - 1) / nd < (3k * sqrt(n) - 1) / nd < 3k * sqrt(n) / nd$ (Lemma 1)

$\Rightarrow 3k * sqrt(n) / nd < 3 * (1/3) * n**(3/4) / nd = 1 / d * n**(1/4)$ (Lemma 2)

$\Rightarrow 1 / d * n**(1/4) < 1 / (d * 2d) = 1 / 2d**2$ (Lemma 3)

得到 e / n 和 k / d 相差小於 1 / 2d**2

因此可以利用 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

- $\varphi(n)$怎麼不見了?

$e * d = k * \varphi(n) + 1$
$\Rightarrow \varphi(n) = (e * d - 1) / k$

- 我們還要$\varphi(n)$做啥?
  - d = invert(e, phi)
  - 拿到 p+q 和 p*q 進一步去分解 (p, q)
    $\varphi(n) = (p-1) * (q-1) = pq - (p+q) + 1 = n - (p+q) + 1$
    $\Rightarrow p+q = n - \varphi(n) + 1$
    生成 x ** 2 - (p + q) x + pq = 0
    則 x 的兩根解為 (p, q)
- 簡單來說就是要驗證 RSA

After Lecture…

- Homomorphic
- LSB Oracle Attack
- Known High Bits
- Williams's p + 1 Algorithm
- Bleichenbacher
- Modular sqrt Algorithm / Tonelli-Shanks algorithm
- Pohlig–Hellman / Discrete logarithm
- Coppersmith Method
- Boneh-Durfee's Attack

THE END

Thanks for listening