1. As you know, ElGamal public key encryption enables re-encryption: from a ciphertext of m one can get easily a random ciphertext of m. However, it requires knowledge of the public key used to create this ciphertext.

a) • What happens if a wrong public key is used during re-encryption?

b) • Show that it is possible to re-encrypt without knowledge of the public key, if a ciphertext of M has the form

$$(\mathrm{PK}^k \cdot M, g^k, \mathrm{PK}^l, g^l) = (y^k \cdot M, g^k, y^l, g^l)$$

for random k, l.

c) • Show that it is possible to re-encrypt without the public key if ElGamal is used to encrypt only one bit where 0 is encrypted as (PK^k, g^k), while 1 is encrypted as $(\mathrm{PK}^k \cdot z, g^k)$ for an arbitrary $z \neq 1$.

$(\mathrm{PK}^k, g^k) = (y^k, g^k)$, $(\mathrm{PK}^k \cdot z, g^k) = (y^k \cdot z, g^k)$

ElGamal: $sk = x$, $pk = y = g^x$, obliczenia dzieją się w grupie G rzędu q z generatorem g (parametry (G, g, q) są jawne)

$\mathrm{Enc}_y(m)$:
- $k \leftarrow_\$ \mathbb{Z}_q^*$
- $c_1 := g^k$
- $c_2 := m \cdot y^k$
- return (c_1, c_2)

$\mathrm{Dec}_x(c = (c_1, c_2))$
- $m := c_2 / c_1^x = (m \cdot y^k)/(g^k)^x = (m \cdot y^k)/(g^x)^k = (m \cdot y^k)/y^k = m$
- return m

Re-enkrypcja: Mając $(g^k, m \cdot y^k)$ wybieramy r i liczymy $(g^k \cdot g^r, m \cdot y^k \cdot y^r) = (g^{k+r}, m \cdot y^{k+r})$ – to tak, jakbyśmy wybrali inne k

a) Mamy szyfrogram $(g^k, m\cdot y^k)$. Jeśli użyjemy $y' \neq y$ do re-enkrypcji:
wybieramy losowe r.

$$(g^k\cdot g^r, m\cdot y^k\cdot y'^r) = (g^{k+r}, m\cdot y^{k+r\alpha})$$

, gdzie α to logarytm dyskretny między y i y'.

$k+r \neq k+r\alpha$, więc to się nie odszyfruje:

$$m' = m\cdot y^{k+r\alpha}/(g^{k+r})^x = m\cdot y^{k+r\alpha}/y^{k+r} = m\cdot y^{r(\alpha-1)}$$

– czyli odbiorca nie odszyfruje, bo nie zna ani r, ani α.

b) Mamy szyfrogram $(y^k\cdot m, g^k, y^l, g^l)$

Re-enkrypcja: wybieramy losowe r, s

$$(y^k\cdot m\cdot (y^l)^r, g^k\cdot (g^l)^r, (y^l)^s, (g^l)^s) = (y^{k+lr}\cdot m, g^{k+lr}, y^{l\cdot s}, g^{l\cdot s}).$$

c) re-enkrypcja 0:

mamy (y^k, g^k).

wybieramy r i liczymy

$$((y^k)^r, (g^k)^r) = (y^{kr}, g^{kr})$$

re-enkrypcja 1:

mamy $(y^k\cdot z, g^k)$

Jeśli z może być dowolne:

wybieramy r i liczymy

$$((y^k\cdot z)^r, (g^k)^r) = (y^{kr}\cdot z^r, g^{kr})$$

jeśli z musi być ustalone, to potrzebujemy jeszcze dodatkowo mieć szyfrogram 0: (y^l, g^l).

Wybieramy r i liczymy:

$$(y^k\cdot z\cdot (y^l)^r, g^k (g^l)^r) = (y^{k+lr}\cdot z, g^{k+lr})$$

2. Recall Chinese Reminder Theorem.

a) • Show that for an RSA number $n = p \cdot q$ there a 4 numbers $x < n$ such that $x^2 = 1$. Two of them are obvious: 1 and $n - 1$. What are the other ones? They are called non-trivial roots of 1.

b) • Show that knowledge of a nontrivial root of 1 modulo n enables breaking RSA based on n.

$$X^2 \equiv 1 \mod n$$

$\downarrow$ z CRT

$$\begin{cases} x^2 \equiv 1 \mod p \\ x^2 \equiv 1 \mod q \end{cases}$$

to daje łącznie 4

a) rozwiązania:

I $\begin{cases} x \equiv 1 \mod p \\ x \equiv 1 \mod q \end{cases}$ $\quad (1,1) \xleftrightarrow{CRT} 1$

II $\begin{cases} x \equiv -1 \mod p \\ x \equiv -1 \mod q \end{cases}$ $\quad (-1,-1) \xleftrightarrow{CRT} n-1$

III $\begin{cases} x \equiv 1 \mod p \\ x \equiv -1 \mod q \end{cases}$ $\quad (1,-1) \xleftrightarrow{CRT} \tilde{X} = 1 + (q-2)p(p^{-1} \mod q)$

IV $\begin{cases} x \equiv -1 \mod p \\ x \equiv 1 \mod q \end{cases}$ $\quad (-1,1) \xleftrightarrow{CRT} \hat{X} = 1 + (p-2)q(q^{-1} \mod p)$

Mapa z $\mathbb{Z}_p^* \times \mathbb{Z}_q^* \to \mathbb{Z}_n$ (przypomnienie)

$(x_1, x_2) \longrightarrow X$

$$X = x_1(q^{-1} \mod p)q + x_2(p^{-1} \mod q)p$$
$$= x_1 + (x_2 - x_1)p(p^{-1} \mod q)$$
$$= x_2 + (x_1 - x_2)q(q^{-1} \mod p)$$

To przejście wykorzystuje własność znaną z tw. Euklidesa

$$p(p^{-1} \mod q) + q(q^{-1} \mod p) = 1$$

bo $\gcd(p,q) = 1$

b)

$$\gcd(\tilde{x} - 1, n) = p$$

$$\gcd(\hat{x} - 1, n) = q$$

3. One of the problems of RSA is its computational complexity. In order to compute $m^d \bmod n$ the basic trick is as follows:

- find a binary representation of $d = d_u d_{u-1} \ldots d_0$ (we can assume that $d_u = 1$)
- put $c_u = m$ and inductively compute $c_{j-1} = c_j^2 \bmod n$ if $d_{j-1} = 0$, else $c_{j-1} = c_j^2 \cdot m \bmod n$.

a) Show that $c_0 = m^d \bmod n$.

b) Estimate computational complexity of this exponentiation method.

Ad a) $c_{j-1} = c_j^2 \cdot m^{d_{j-1}} \bmod n$

$c_u = m = m^{d_u}$ (zakładamy, że $d_u = 1$)

$c_{u-1} = c_u^2 \cdot m^{d_{u-1}} = (m^{d_u})^2 \cdot m^{d_{u-1}} = m^{2d_u + d_{u-1}} = m^{\overline{d_u d_{u-1}}}$ (gdzie $\overline{d_u d_{u-1} \ldots d_0}$ to d zapisane bitowo)

$c_{u-2} = (c_{u-1})^2 \cdot m^{d_{u-2}} = (m^{2d_u + d_{u-1}})^2 \cdot m^{d_{u-2}} = m^{2^2 d_u + 2^1 d_{u-1} + d_{u-2}} = m^{\overline{d_u d_{u-1} d_{u-2}}}$

$\vdots$

$c_0 = m^{2^u d_u + 2^{u-1} d_{u-1} + \ldots + 2 \cdot d_1 + d_0} = m^{\overline{d_u d_{u-1} \ldots d_1 d_0}} = m^d \quad \square$

Ad b) Operację wykonujemy u razy ($\lfloor \log_2(d) \rfloor$). W każdym kroku wykonujemy jeden kwadrat i maksymalnie jedno dodatkowe mnożenie modulo.

1° Koszt mnożenia (wiemy, że operacje dzieją się modulo n, więc mnożone liczby mają maksymalnie długości $\log_2(n)$):

→ typowe naiwne mnożenie to $O((\log_2(n))^2)$

→ mnożenie z wykorzystaniem algorytmu Karatsuby to $O\left((\log_2(n))^{\log_2 3}\right)$

2° Koszt kwadratu jest zbliżony ale mniejszy od kosztu mnożenia.

3° Koszt redukcji modulo ($x \bmod n$, gdzie $x < n^2$)

→ naiwna redukcja: co najwyżej $\log_2(n)$ odejmowań liczby długości $\log_2(n)$, czyli $O((\log_2(n))^2)$

→ do zastanowienia się: redukcja Barretta, redukcja Montgomery'ego

Z 1°, 2° i 3° koszt rundy to w naiwnej implementacji $O((\log_2(n))^2)$.

Czyli koszt całego potęgowania to $O(\log_2(d) \cdot (\log_2(n))^2)$.

4. Complexity of RSA encryption and decryption can be reduced by application of Chinese Remainder Theorem. Check how to do it and what we can gain regarding the computation cost.

RSA z CRT opisane zostało w notatkach do pierwszych ćwiczeń.

Druga część zadania do wykonania eksperymentalnie jako zadanie domowe.