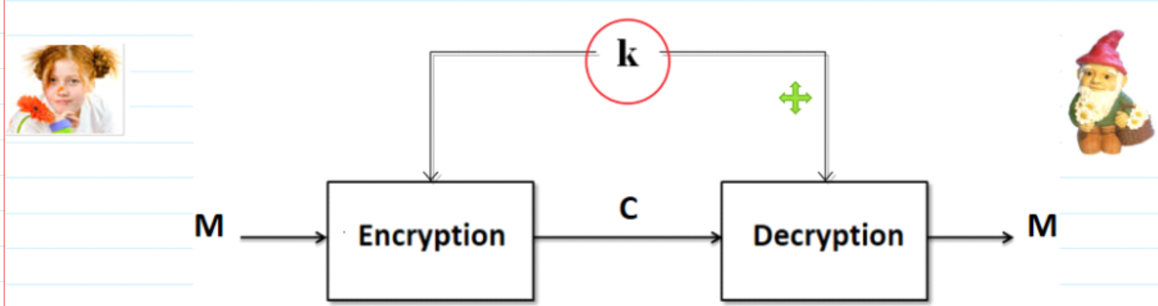
We agreed that MidTerm Exam will be held in October 31-th at 15:30 in 103f.
During the MidTerm Exam you must solve 5 problems in
https://imimsociety.net/en/14-cryptography
namely: DH-KAP, MIM Attack, RSA Sign., RSA Enc., AKAP with RSA.
Register to the site in the similar way as you are registering in eShop.
After that you will receive 10 Eur virtual money to purchase the problems.
Please purchase only one problem at time and after solving it purchase the next one.

Course Works (CW) list is presented in my Google drive
https://docs.google.com/document/d/1GDVZuRPtmQ5Z--IdqGunPx_3qOGSfrpR/edit?usp=sharing&ouid=111502255533491874828&rtpof=true&sd=true

Please choose topic and label it by the first letter of surname dot name, e.g. S.Name.
For some of topics the group project realization can take place.
Requirements for CW you can find in
http://crypto.fmf.ktu.lt/xdownload/
in files Course_Work

Symmetric encryption



AES-128, 192, 256

Public Key CryptoSystems - PKCS

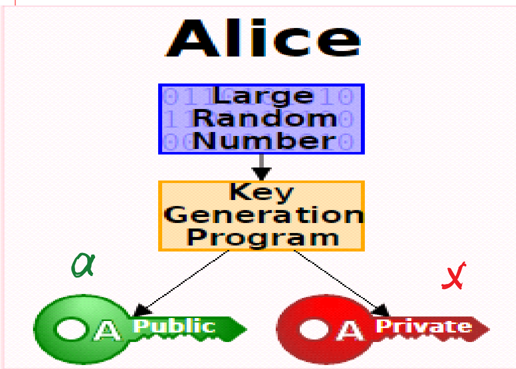
ElGamal Cryptosystem

1.Public Parameters generation

Generate strong prime number **p**.
Find a generator **g** in $Z_p^* = \{1, 2, 3, \ldots, p-1\}$ using condition.
Strong prime $p=2q+1$, where q is prime, then g is a generator of Z_p^* iff
$g^q \neq 1 \mod p$ and $g^2 \neq 1 \mod p$.
Declare **Public Parameters** to the network **PP** = (**p**, **g**); **p= 268435019**; **g=2.**

2^28-1= 268,435,455

```
>> int64(2^28-1)
ans = 268435455
>> dec2bin(ans)
ans = 1111 1111 1111 1111 1111 1111 1111
```



2.Key generation

- Randomly choose a private key ***x*** with
  $1 < x < p - 1$.
- Compute $a = g^{x} \bmod p$.
- The public key is **PuK** = a.
- The private key is **PrK** = x.

Asymmetric Signing - Verification

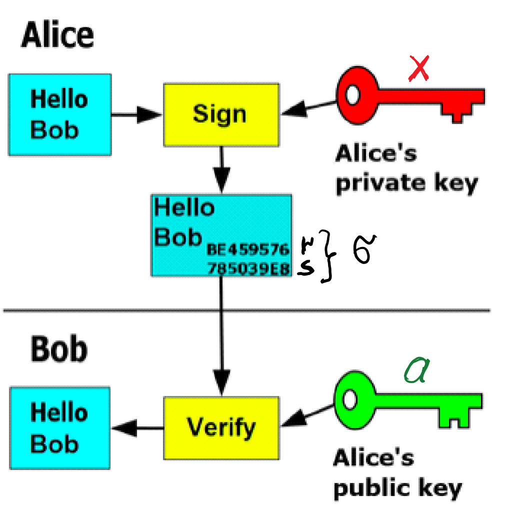
σ=Sig(PrK$_A$, m)

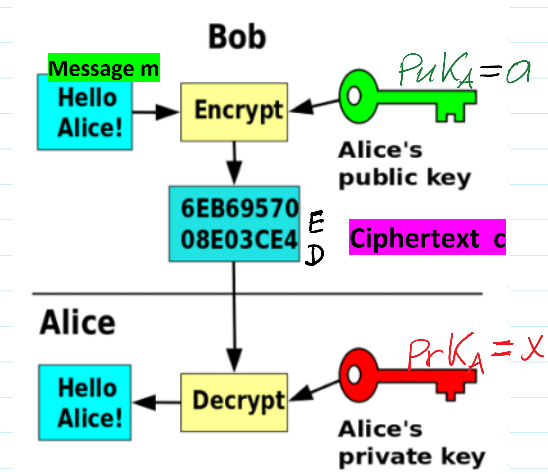
V=Ver(PuK$_A$, σ, m), V∈{True, False}≡{1, 0}

Asymmetric Encryption - Decryption

c=Enc(PuK$_A$, m)

m=Dec(PrK$_A$, c)





El-Gamal E-Signature

The **ElGamal signature scheme** is a digital signature scheme which is based on the difficulty of computing discrete logarithms.
It was described by Taher ElGamal in 1984. The ElGamal signature algorithm is rarely used in practice.
A variant developed at NSA and known as the Digital Signature Algorithm is much more widely used.
The ElGamal signature scheme allows a third-party to confirm the authenticity of a message sent over an insecure channel.

From <https://en.wikipedia.org/wiki/ElGamal_signature_scheme>

EC Gamal sign. → Digital Signature Alg. (DSA) NSA
→ Elliptic Curve DSA – ECDSA Certicom

3.***Signature creation***

To sign any finite message ***M*** the signer performs the following steps using public parametres **PP**.

- Compute $h=H(M)$.
- Choose a random k such that $1 < k < p - 1$ and gcd$(k, p - 1) = 1$.
- $k^{-1} \bmod (p-1)$ computation: $k^{-1} \bmod (p-1)$ **exists** if gcd$(k, p - 1) = 1$, i.e. **k** and **p-1** are relatively prime.

k^{-1} can be found using either Extended Euclidean algorithmt or Euler theorem or .....

>> k_m1=mulinv(k,p-1) % k^{-1}mod (p-1) computation.

- Compute $r=g^k \bmod p$
- Compute $s=(h-xr)k^{-1} \bmod (p-1) \dashrightarrow h=xr+sk \bmod (p-1)$,

Signature $\sigma=(r,s)$

4.***Signature Verification***

A signature $\sigma=(r,s)$ on message M is verified using Public Parameters **PP**=(p, g) and **PuK$_A$=a**.

1. Bob computes $h=H(M)$.
2. Bob verifies if $1<r<p-1$ and $1<s<p-1$.
3. Bob calculates $V1=g^h \bmod p$ and $V2=a^r r^s \bmod p$, and verifies if $V1=V2$.

The verifier Bob accepts a signature if all conditions are satisfied and rejects it otherwise.

5.**Correctness**

The algorithm is correct in the sense that a signature generated with the signing algorithm will always be accepted by the verifier.

The signature generation implies

$$h=xr+ks \bmod (p-1)$$

Hence Fermat's little theorem implies that all operations in the exponent are computed mod (p-1)

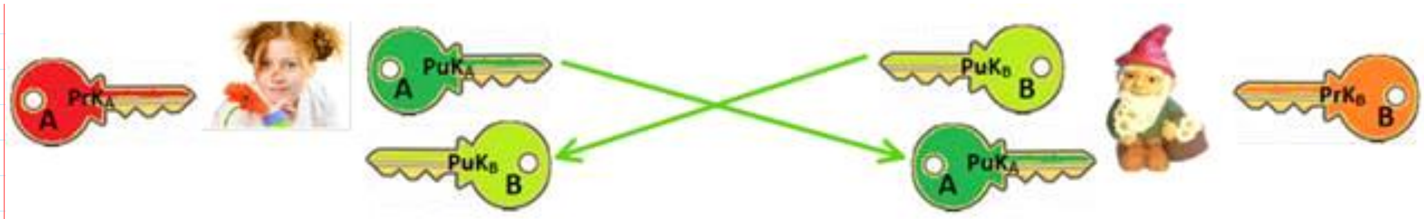
$$\underbrace{g^h \bmod p}_{V1}=g^{(xr+ks) \bmod (p-1)} \bmod p = g^{xr}g^{ks} = (g^x)^r(g^k)^s = \underbrace{a^r r^s \bmod p}_{V2}$$

Asymmetric Encryption-Decryption: El-Gamal Encryption-Decryption

p=268435019; g=2.

Let message m needs to be encrypted, e.g. m = 111222.

$\Rightarrow m < p \Rightarrow m \bmod p = m.$



$\mathcal{A}$: $\xrightarrow{PuK_A = a}$ $\mathcal{B}$: is able to encrypt m to $\mathcal{A}$: $m < p$

$\mathcal{B}$: $r \leftarrow randi(\mathcal{Z}_p^*)$

$$\left.\begin{array}{l} E = m \cdot a^r \bmod p \\ D = g^r \bmod p \end{array}\right\} c = (E, D) \longrightarrow$$

$\mathcal{A}$: is able to decrypt $c = (E, D)$ using her $PrK_A = x$.

1. $D^{-x \bmod (p-1)} \bmod p$
2. $E \cdot D^{-x} \bmod p = m$

$$(-x) \bmod (p-1) = (0 - x) \bmod (p-1) =$$
$$= (p - 1 - x) \bmod (p-1)$$

D^{-x} mod ***p*** computation using Fermat theorem:
If ***p*** is prime, then for any integer ***a*** holds **$a^{p-1} = 1 \bmod p$**.

$$D^{p-1} = 1 \bmod p \quad / \cdot D^{-x}$$
$$\underbrace{D^{p-1} \cdot D^{-x}} = 1 \cdot D^{-x} \bmod p \Rightarrow \underbrace{D^{p-1-x}} = D^{-x} \bmod p$$

$$\boxed{D^{-x} \bmod p = D^{p-1-x} \bmod p}$$

Correctness

$$Enc_{PuK_A}(m, r) = c = (E, D) = (E = m \cdot a^r \bmod p ; D = g^r \bmod p)$$

$$Dec_{PrK_A}(c) = E \cdot D^{-x} \bmod p = m \cdot a^r \cdot (g^r)^{-x} \bmod p =$$
$$= m \cdot (\underbrace{g^x}_{a})^r \cdot g^{-rx} = m \cdot g^{xr} \cdot g^{-rx} = m \cdot g^{xr - rx} \bmod p = m \cdot g^0 \bmod p =$$
$$= m \cdot 1 \bmod p = m \bmod p = m$$

since $m < p$

If $m > p \rightarrow m \bmod p \neq m$; $27 \bmod 5 = 2 \neq 27$.

If $m < p \rightarrow m \bmod p = m$; $19 \bmod 31 = 19$.

ASCII

$\frac{2048}{8} =$

If $m < p \rightarrow m \bmod p = m$; $19 \bmod 31 = 19$. $\frac{2048}{8} =$

Decryption is correct if $m < p$. $= 256$ char.

ElGamal encryption is probabilistic: encryption of the same message m two times yields the different cyphertexts c_1 and c_2.

1-st encryption:

$r_1 \leftarrow randi(\mathbb{Z}_p^*)$

$$\left.\begin{array}{l} E_1 = m \cdot a^{r_1} \bmod p \\ D_1 = g^{r_1} \bmod p \end{array}\right\} C_1 = (E_1, D_1)$$

2-nd encryption

$r_2 \leftarrow randi(\mathbb{Z}_p^*)$

$$\left.\begin{array}{l} E_2 = m \cdot a^{r_2} \bmod p \\ D_2 = g^{r_2} \bmod p \end{array}\right\} C_2 = (E_2, D_2)$$

$r_1 \neq r_2$

$C_1 \neq C_2$

Necessity of probabilistic encryption.
Encrypting a message with textbook RSA always yields the same ciphertext, and so we actually obtain that any deterministic scheme must be insecure for multiple encryptions.

Tavern episode

Key agreement protocol using ElGamal encryption

How to encrypt large data file : Hybrid enc-dec method.

1. Parties must agree on common symmetric secret k for symmetric block cipher, e.g. AES-128, 192, 256 bits.

$\mathcal{B}$: 1) $k \leftarrow randi(2^{256})$

$\mathcal{I}_0$

$Enc(PuK_A, k) = c = (E, D) \xrightarrow{c} \mathcal{A}$:

$Dec(PrK_A, c) = k$

2) M - large file to be ecrypted

$E_k(M) = AES_k(M) = G \xrightarrow{G}$

$D_k(G) = AES_k(G) = M.$

3) $Sig(PrK_B, c) = \sigma = (r, s) \xrightarrow{\sigma}$

(1) Verify if PuK is valid

3) $Sig(PrK_B, c) = \sigma = (r, s)$ $\xrightarrow{\sigma}$

① Verify if PuK_B is valid

② Verify if σ is valid

Homomorphic property of ElGamal encryption

Let we have 2 messages m_1, m_2 to be encrypted

$r_1 \leftarrow randi(\mathbb{Z}_p^*)$ $\qquad$ $r_2 \leftarrow randi(\mathbb{Z}_p^*)$

$E_1 = m_1 \cdot a^{r_1} \bmod p$ $\qquad$ $E_2 = m_2 \cdot a^{r_2} \bmod p$

$D_1 = g^{r_1} \bmod p$ $\qquad$ $D_2 = g^{r_2} \bmod p$

$$Enc_{a, r_1, r_2}(m_1 \cdot m_2) = c_{12} = (E_{12}, D_{12})$$

$$E_{12} = m_1 \cdot m_2 \cdot a^{r_1 + r_2} \bmod p = (\underbrace{m_1 \cdot a^{r_1} \bmod p}_{E_1} \cdot \underbrace{m_2 \cdot a^{r_2} \bmod p}_{E_2}) \bmod p$$

$$E_{12} = E_1 \cdot E_2 \bmod p$$

$$D_{12} = g^{r_1 + r_2} \bmod p = (\underbrace{g^{r_1} \bmod p}_{D_1} \cdot \underbrace{g^{r_2} \bmod p}_{D_2}) \bmod p$$

$$D_{12} = D_1 \cdot D_2 \bmod p$$

$$\boxed{Enc_{a, r_1, r_2}(m_1 \cdot m_2) = c_1 \cdot c_2 \bmod p}$$

Multiplicative isomorphism

Multiplicatively additive isomorphism

$Enc(m_1 + m_2) = c_1 + c_2$ ⇐ Pascal Paillier encryption.

One special encryption is instead of m_1, m_2 encryption to encrypt messages $n_1 = g^{m_1}$, $n_2 = g^{m_2}$

$Enc(m_1 + m_2) = c_1 \circ c_2$

Homomorphic encryption: cloud computation with encrypted data.

Paillier encryption scheme is additively-multiplicative homomorphic and has a potentially nice applications in blockchain, public procurement, auctions, gamblings and etc.

$$\mathbf{Enc(Puk, m_1+m_2) = c_1 \bullet c_2.}$$