Koliokviumas: Balandžio 8 d. 17:00-18:30 per Zoom
https://imimsociety.net/en/
Prisiregistruoti: Pa VardasKK
https://imimsociety.net/en/14-cryptography
Uždavinys yra užskaitomas, jeigu paskutiniame žingsnyje jūs paspaudžiate mygtuką [Get Reward]
Pastaba: vienu metu perkate tiktai 1 uždavinį ir jį išsprendus perkate kitą.

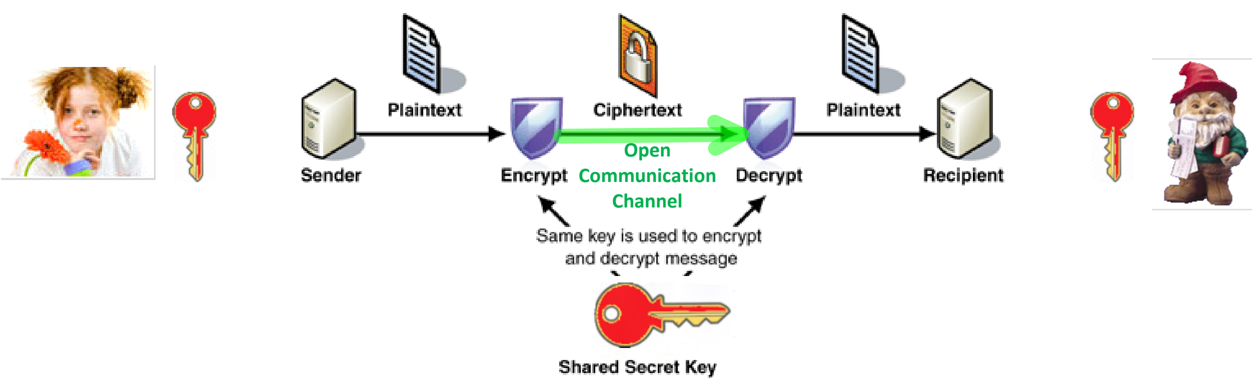
Cryptography:
Information confidentiality, integrity, authenticity & person identification

Symmetric Cryptography -------------------- Asymmetric Cryptography Public Key Cryptography

Symmetric encryption
H-functions, Message digest
HMAC H-Message Authentication Code

Asymmetric encryption
E-signature - Public Key Infrastructure - PKI
E-money, Blockchain
E-voting
Digital Rights Management - DRM (Marlin)
Etc.

Symmetric - Secret Key Encryption - Decryption



Public Key Cryptography - PKC

Principles of Public Key Cryptography
Instead of using single symmetric key shared in advance by the parties for realization of symmetric cryptography, asymmetric cryptography uses two ***mathematically*** related keys named as private key and public key we denote by **PrK** and **PuK** respectively.
PrK is a secret key owned ***personally*** by every user of cryptosystem and must be kept secretly. Due

to the great importance of **PrK** secrecy for information security we labeled it in red color. **PuK** is a non-secret ***personal*** key and it is known for every user of cryptosystem and therefore we labeled it by green color. The loss of **PrK** causes a dramatic consequences comparable with those as losing password or pin code. This means that cryptographic identity of the user is lost. Then, for example, if user has no copy of **PrK** he get no access to his bank account. Moreover his cryptocurrencies are lost forever. If **PrK** is got into the wrong hands, e.g. into adversary hands, then it reveals a way to impersonate the user. Since user's **PuK** is known for everybody then adversary knows his key pair (**PrK**, **Puk**) and can forge his Digital Signature, decrypt messages, get access to the data available to the user (bank account or cryptocurrency account) and etc.
Let function relating key pair (**PrK**, **Puk**) be ***F***. Then in most cases of our study (if not declared opposite) this relation is expressed in the following way:

$$\mathbf{PuK}=\boldsymbol{F}(\mathbf{PrK}).$$

In open cryptography according to Kerchoff principle function ***F*** must be known to all users of cryptosystem while security is achieved by secrecy of cryptographic keys. To be more precise to compute **PuK** using function ***F*** it must be defined using some parameters named as public parameters we denote by **PP** and color in blue that should be defined at the first step of cryptosystem creation. Since we will start from the cryptosystems based on discrete exponent function then these public parameters are

$$\mathbf{PP} = (\boldsymbol{p}, \boldsymbol{g}).$$

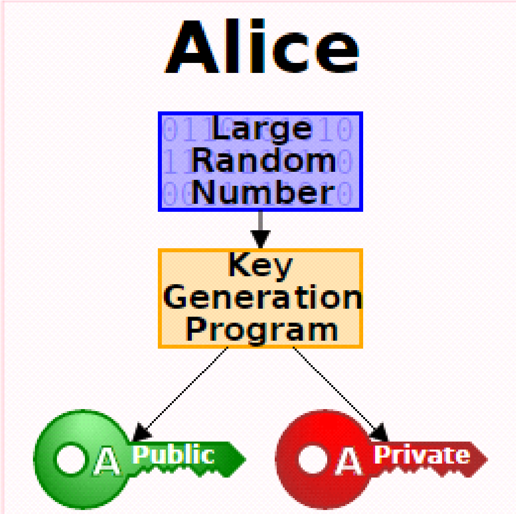
Notice that relation represents very important cause and consequence relation we name as the direct relation: when given **PrK** we compute **PuK**.
Let us imagine that for given ***F*** we can find the inverse relation to compute **PrK** when **PuK** is given. Abstractly this relation can be represented by the inverse function $\boldsymbol{F}^{-1}$. Then

$$\mathbf{PrK}=\boldsymbol{F}^{-1}(\mathbf{PuK}).$$

In this case the secrecy of **PrK** is lost with all negative consequences above. To avoid these undesirable consequences function ***F*** must be **one-way function** – OWF. In this case informally OWF is defined in the following way:
1. The computation of its direct value **PuK** when **PrK** and ***F*** in are given is effective.
2. The computation of its inverse value **PrK** when **PuK** and ***F*** are given is infeasible, meaning that to find $\boldsymbol{F}^{-1}$ is infeasible.

The one-wayness of ***F*** allow us to relate person with his/her **PrK** through the **PuK**. If ***F*** is 1-to-1, then the pair (**PrK**, **Puk**) is unique. So **PrK** could be reckoned as a unique secret parameter associated with certain person. This person can declare the possession or **PrK** by sharing his/her **PuK** as his public parameter related with **PrK** and and at the same time not revealing **PrK**.
So, every user in asymmetric cryptography possesses key pair (**PrK**, **PuK**). Therefore, cryptosystems based on asymmetric cryptography are named as **Public Key CryptoSystems** (PKCS).
We will consider the same two traditional (canonical) actors in our study, namely Alice and Bob. Everybody is having the corresponding key pair ($\mathbf{PrK_A}$, $\mathbf{PuK_A}$) and ($\mathbf{PrK_B}$, $\mathbf{PuK_B}$) and are exchanging with their public keys using open communication channel as indicated in figure below.

Asymmetric - Public Key Cryptography



PrK and PuK are related

PuK = F(PrK)

F is one-way function

Having **PuK** it is infeasible to find

PrK = F^{-1}(PuK)

F(*x*)=*a* is OWF, if:

1.It easy to compute *a*, when *F* and *x* are given.

2.It is infeasible compute *x* when *F* and *a* are given.

PrK = *x* <-- randi ==> PuK = *a* = g^x mod p

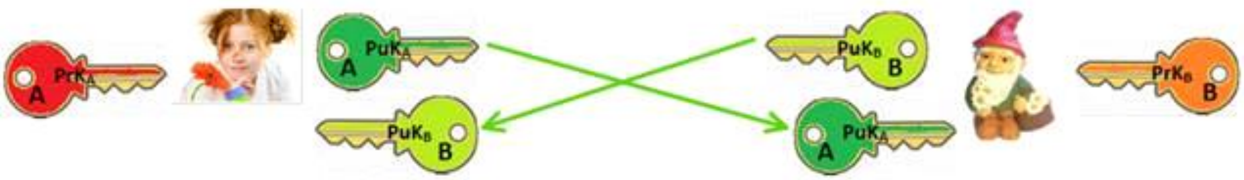
Public Parameters PP = (p, g)

$p \sim 2^{2048}$ ⟹ $|p| \cong 2048$ bits

$p \sim 2^{28}$ ⟹ $|p| \cong 28$ bits

Threaths of insecure PrK generation

$\mathbb{Z}_p^* = \{1, 2, 3, \ldots, p-1\}$; $*$ mod p



Message m < p

Asymmetric Signing - Verification

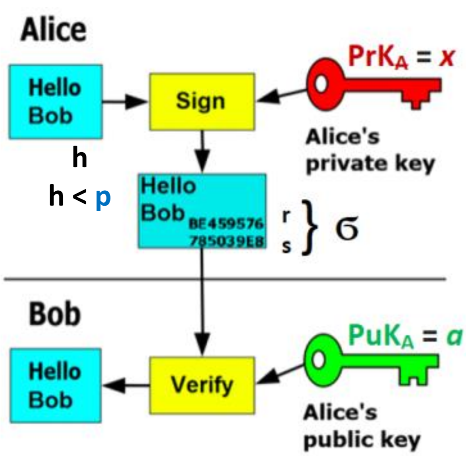
Sign(PrK_A, h) = б = (r, s)

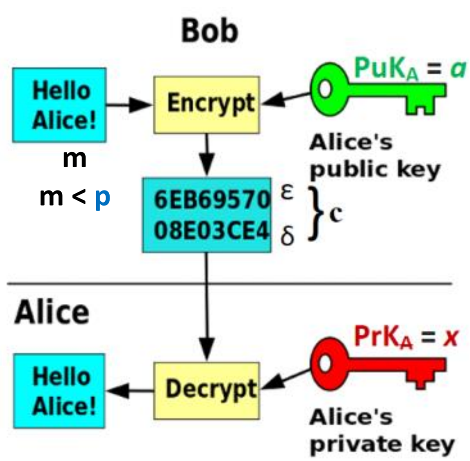
V=Ver(PuK_A, h, б), V$\in\{\text{True, False}\} \equiv \{1, 0\}$

Asymmetric Encryption - Decryption

c=Enc(PuK_A, m)

m=Dec(PrK_A, c)





ElGamal Cryptosystem

1.Public Parameters generation **PP** = (*p*, *g*).

Generate strong prime number **p**: >> p=genstrongprime(28) % strong prime of 28 bit length

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

p= 268435019; **g=2;**

2^28-1= 268,435,455

```
>> 2^28-1
ans = 2.6844e+08
>> int64(2^28-1)
ans = 268435455
```

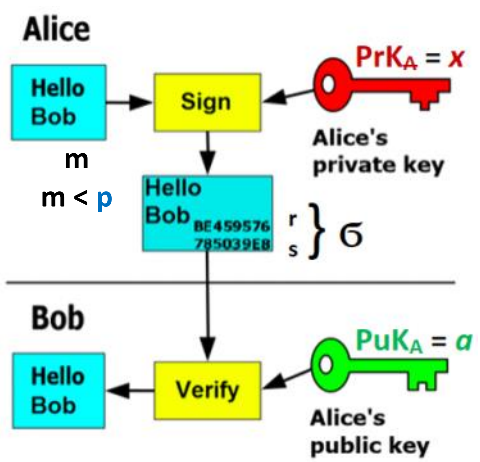
PrK = *x* <-- randi ==> PuK = *a* = g^x mod p

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



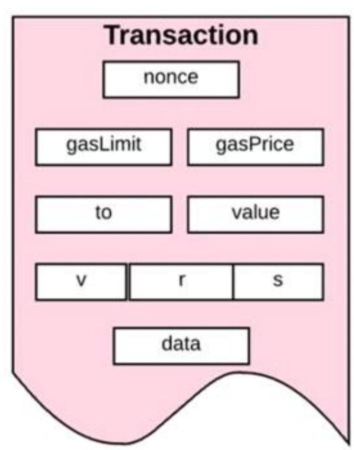
Signature creation for message ***M*** >> ***p***.

1. Compute decimal h-value $h=H(M)$; $h<p$.
2. Generate >> ***i*** =int64(randi(***p***-1)) % such that gcd(***i***,***p***-1)=1.
3. Compute i^{-1} mod (***p***-1). You can use the function
   >> i_m1=mulinv(i, p-1); >>mod(i * i_m1 , p-1)=1.
4. Compute $r=g^i \bmod p$.
5. Compute $s=(h-xr)i^{-1} \bmod (p-1)$.
6. Signature on h-value ***h*** is σ = (***r***,***s***)
   Sign(***x***,***h***) = σ = (***r***,***s***).

```
>> p=int64(genstrongprime(28))

>> p= int64(268435019)
p = 268435019
>> g=2
g = 2
```

```
>> i=randi(p-1)
i = 1.1728e+08
>> i=int64(randi(p-1))
i = 47250243
>> gcd(i,p-1)
ans = 1
>> i_m1=mulinv(i,p-1)
i_m1 = 172715821
>> mod(i*i_m1,p-1)
ans = 1
```



Tx = ' nonce || gasLimit || gasPrice || to || value || data '

$h = H(T_x) \longrightarrow \sigma = (r, s) = \mathrm{Sign}(PrK, h)$

1. ***Signature creation***

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

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

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

>> i_m1=mulinv(i,p-1) % i^{-1}mod (p-1) computation.

- Compute $\mathbf{r=g^i \bmod p}$
- Compute $\mathbf{s=(h-xr)i^{-1} \bmod (p-1)} \dashrightarrow \mathbf{h=xr+is \bmod (p-1)}$

Signature $\boldsymbol{\sigma}$**=(r,s)**

$$\left.\begin{aligned} s &= (h-x\cdot r)\cdot i^{-1} \quad / i \\ s\cdot i &= (h-x\cdot r)\cdot \underbrace{\cancel{i^{-1}}\cdot\cancel{i}} \\ h-xr &= s\cdot i \longrightarrow h = x\cdot r + i\cdot s \end{aligned}\right\} \bmod p$$

2. ***Signature Verification***

A signature $\boldsymbol{\sigma=(r,s)}$ on message $\boldsymbol{M}$ is verified using Public Parameters **PP**=(**p**, **g**) and $\mathbf{PuK_A=a}$.

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

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

3. **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

$\mathbf{h=xr+is \bmod (p-1)}$

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

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

$PP = (p, g)$

$\mathcal{A}$: $z \leftarrow \mathrm{randi}(p-1)$

$v = g^z \bmod p$

$\left\{\begin{array}{l}\text{Dear } \mathcal{B} \text{ I am } \mathcal{A} \\ \text{and I am sending} \\ \text{you my } PuK = v\end{array}\right\} \longrightarrow$ $\mathcal{B}$: Believes that $PuK = v$ is of $\mathcal{A}$

$m =$ 'Bob get out'

$\sigma = \mathrm{Sign}(z, m) = (r, s)$ $\xrightarrow{m,\ \sigma = (r, s)}$ $\mathcal{B}$: verifies the signature σ on m using $PuK = v$ and verification passes.

$\sigma = sign(z, m) = (r, s)$

on m using PuK = V and verification passes.

Before Bob verifies any signature with someone PuK he must be sure that this PuK is got from the certain person, e.g. A but not from anybody else!

It is achieved by creation of PKI - Public Key Ifrastructure when Trusted Third Party (TTP) such as Certification Authority is introduced.

CA is issuing PuK Certificates for any user by signing PuK when user proves his/her identity to CA.

A: Identification Card - ID

$PrK_A = x$; $PuK_A = a$.

ID

PuK_A

CA: PrK_{CA} ; PuK_{CA}.

$h_A = H(PuK_A || Data_A)$

$Sign(PrK_{CA}, h_A) = \sigma_A$

$Cert_A$

$Cert_A = ' PuK_A || Data_A || \sigma_A '$

$h_A = H(PuK_A || Data_A)$

B: $Ver(PuK_{CA}, \sigma_A, h_A) = True$

Is sure that certificate is True

Since CA is TTP & B can download PuK_{CA} using his browser with known to everyone link

https://CertificationAuthority.Tusted.com

https://certicom.com

```
>> p= int64(268435019)
p = 268435019
>> g=2;
>> x =int64(randi(p-1))
x = 65770603
>> a=mod_exp(g,x,p)
a = 232311991
>> M='Hello Bob...'
M = Hello Bob...
>> h=hd28(M)
h = 150954921
```

```
>> i =int64(randi(p-1))
i = 201156232
>> gcd(i,p-1)
ans = 2
>> i =int64(randi(p-1))
i = 35395315
>> gcd(i,p-1)
ans = 1
>> i_m1=mulinv(i,p-1)
i_m1 = 192754179
>> mod(i*i_m1,p-1)
ans = 1
```

```
>> r=mod_exp(g,i,p)
r = 172536234
>> hmxr=mod(h-x*r,p-1)
hmxr = 20262153
>> s=mod(hmxr*i_m1,p-1)
s = 44575091
```

```
>> g_h=mod_exp(g,h,p)
g_h = 241198023
>> V1=g_h
V1 = 241198023

>> a_r=mod_exp(a,r,p)
a_r = 49998673
>> r_s=mod_exp(r,s,p)
r_s = 111993804
>> V2=mod(a_r*r_s,p)
V2 = 241198023
```

Till this place

Asymmetric Encryption-Decryption: **El-Gamal Encryption-Decryption**

p=268435019; g=2;

Let message ***m~*** needs to be encrypted, then it must be encoded in decimal number ***m***: 1< ***m*** < ***p***.
E.g. ***m*** = 111222. Then ***m* mod *p* = *m***.

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

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

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

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

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

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

$(p-1) \bmod (p-1) = 0$ since $p-1 - (p-1) = 0$; $(p-1) : (p-1) = 1$

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

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

>> D_mx = mod_exp(D, p-1-x, p-1)

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

$D^{p-1} = 1 \bmod p \quad / \cdot D^{-x \bmod (p-1)} \bmod p$

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

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

Correctness

$Enc(PuK_A = a, i, m) = c = (E, D) = (E = m \cdot a^i \bmod p; D = g^i \bmod p)$

$Dec(PrK_A = x, c) = E \cdot D^{-x} \bmod p = m \cdot a^i \cdot (g^i)^{-x} \bmod p =$

$= m \cdot (\underbrace{g^x}_{a})^i \cdot g^{-ix} = m \cdot g^{xi} \cdot g^{-ix} = m \cdot g^{xi-ix} \bmod p = m \cdot g^0 \bmod p =$

$= m \cdot 1 \bmod p = m \bmod p = m = 111222$

since $m < p$

$= m \cdot 1 \bmod p = m \bmod p = m = 111222$

since $m < p$

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

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

Decryption is correct if $m < p$.

ASCII: 8 bits per char.

$\frac{2048}{8} = 256$ 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:

$i_1 \leftarrow randi(\mathbb{Z}_p^*)$

$\left.\begin{array}{l} E_1 = m \cdot a^{i_1} \bmod p \\ D_1 = g^{i_1} \bmod p \end{array}\right\} c_1 = (E_1, D_1)$

$i_1 \neq i_2$

$c_1 \neq c_2$

2-nd encryption

$i_2 \leftarrow randi(\mathbb{Z}_p^*)$

$\left.\begin{array}{l} E_2 = m \cdot a^{i_2} \bmod p \\ D_2 = g^{i_2} \bmod p \end{array}\right\} c_2 = (E_2, D_2)$

Enigma

Necessity of probabilistic encryption.

Encrypting the same 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

Enigma

Authenticated Key Agreement Protocol using ElGamal Encryption and Signature.

Hybrid encryption for a large files combining asymmetric and symmetric encryption method.

Hybrid encryption. Let ***M*** be a large finite length file, e.g. of gigabytes length.

Then to encrypt this file using asymmetric encryption is extremely ineffective since we must split it into millions of parts having 2048 bit length and encrypt every part separately.

The solution can be found by using **asymmetric encryption** together with **symmetric encryption**, say AES-128.

It is named as **hybrid encryption method**.

For this purpose the **Key Agreement Protocol** (**KAP**) using **asymmetric encryption** for the same symmetric secret key ***k*** agreement must be realized and encryption of ***M*** realized by **symmetric encryption** method, say AES-128.

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

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

A: $PrK_A = x$; $PuK_A = a$.
$PuK_B = b$.

B: $PrK_B = y$; $PuK_B = b$.
$PuK_A = a$.

1) $k \leftarrow randi(2^{128})$
$i_k \leftarrow randi(2^{128})$

$Enc(PuK_B = b, i_k, k) = c = (E, D)$

2) M - large file to be encrypted

$E_k(M) = AES_k(M) = G$

3) Signs ciphertext G

3.1) $h = H(G)$

3.2) $Sign(PrK_A = x, h) = \sigma = (r, s)$

$\xrightarrow{c,\ G}$
σ, PuK_A
$Cert_A$

1.1. Verify if PuK_A and $Cert_A$ are valid?

1.2. Verify if σ on $h = H(G)$ is valid?

$h' = H(G)$

$Ver(PuK_A, \sigma, h') = True$

2. $Dec(PrK_B, c) = k$

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

A was using so called encrypt-and-sign (E-&-S) paradigm.
(E-&-S) paradigm is recomended to prevent so called choosen ciphertext Attacks - CCA: it is most strong attack but most complex in realization.

Till this place