

Mid Term Exam (MTE) will be held on 14-th of April at 17:00 in the 118 class.

You can use your own computers with installed Octave and my .m files.

Otherwise, you must to install and launch Octave in class computer together with downloaded my .m files on them.

During the MTE you must solve 2 problems:

1. Diffie-Hellman Key Agreement Protocol - DH KAP.
2. Man-in-the-Middle Attack (MiMA) for Diffie-Hellman Key Agreement Protocol - DH KAP.

The problems are presented in the site:

[imimsociety.net](http://imimsociety.net)

In section 'Cryptography':

[Cryptography \(imimsociety.net\)](http://imimsociety.net)

Please register to the site and after that you receive 10 Eur virtual money to purchase the problems.

**Please purchase the only one problem at a time.**

If the solution is successful then you are invited to press the green button [Get reward].

Then 'Knowledge bank' will pay you the sum twice you have paid.

So, if the initial capital was 10 Eur of virtual money and you buy the problem of 2 Eur, then if the solution is correct your budget will increase up to 12 Eur.

You can solve the problems in imimsociety as many times as you wish to better prepare for MTE.

I advise you to try at first to solve the problem in 'Intellect' section to exercise the brains.

It is named as 'WOLF, GOAT AND CABBAGE TRANSFER ACROSS THE RIVER ALGORITHM'.

< <https://imimsociety.net/en/home/15-wolf-goat-and-cabbage-transfer-across-the-river-algorithm.html>>

The questions concerning the MTE you can ask at the end of the lectures.

The pictures of problems listed above are the following.



**Cryptography:**  
**Information confidentiality, integrity,**  
**authenticity & person identification**

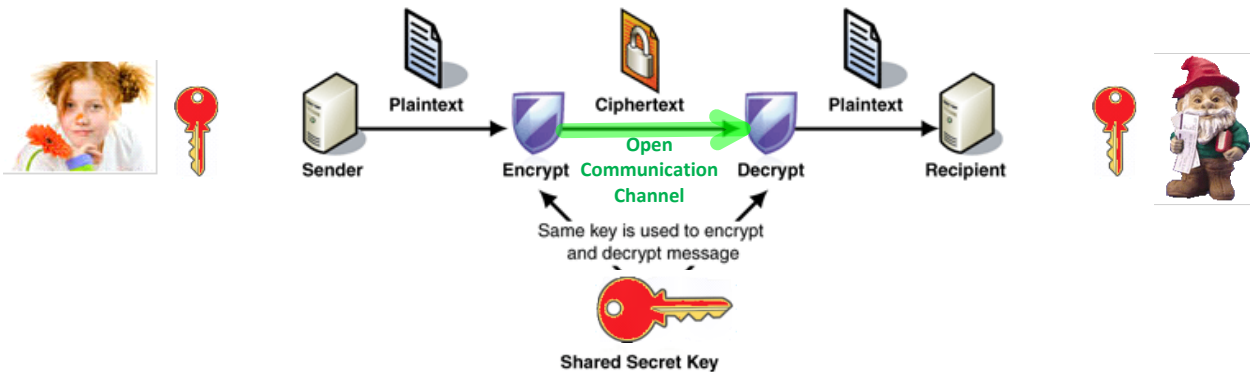
**Symmetric Cryptography ----- Asymmetric Cryptography**

# 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 a **red** color. **PuK** is a non-secret *personal* key and it is known for every user of cryptosystem and therefore we labeled it by a **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:

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

In open cryptography according to **Kerchhoff principle** function  $F$  must be known to all users of cryptosystem while security is achieved by the 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

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

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  $F^{-1}$ . Then

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

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  $F^{-1}$  is infeasible.

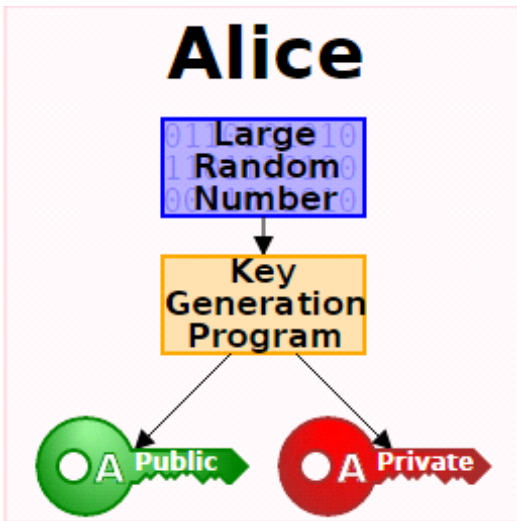
The one-wayness of **F** allow us to relate person with his/her **PrK** through the **PuK**. If **F** is 1-to-1, then the pair (**PrK**, **PuK**) is unique. So **PrK** could be reckoned as a unique secret parameter associated with certain person. This person can declare the possession or **PrK** by sharing 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**.

Everyone is having the corresponding key pair (**PrK<sub>A</sub>**, **PuK<sub>A</sub>**) and (**PrK<sub>B</sub>**, **PuK<sub>B</sub>**) 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

$$\text{PuK} = F(\text{PrK})$$

**F** is one-way function

Having **PuK** it is infeasible to find

$$\text{PrK} = F^{-1}(\text{PuK})$$

$F(x)=a$  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.

$$\text{PrK} = x \leftarrow \text{randi} \implies \text{PuK} = a = g^x \text{ mod } p$$

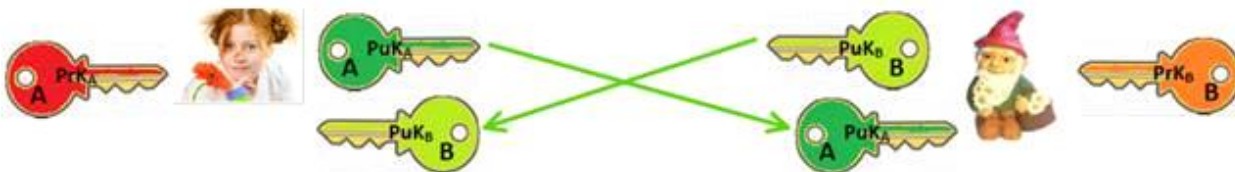
**Public Parameters PP = (p, g)**

#### Threats of insecure PrK generation

$$\mathcal{Z}_p^* = \{1, 2, 3, \dots, p-1\}; * \text{ mod } p$$

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

$$p \sim 2^{28} \implies |p| \cong 28 \text{ bits}$$



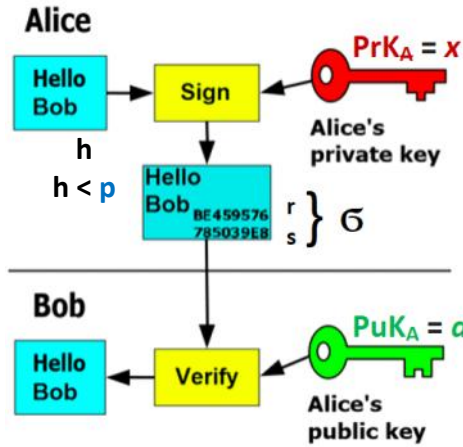
Message  $m < p$

Message  $m < p$

### Asymmetric Signing - Verification

$$\text{Sign}(\text{PrK}_A, h) = \sigma = (r, s)$$

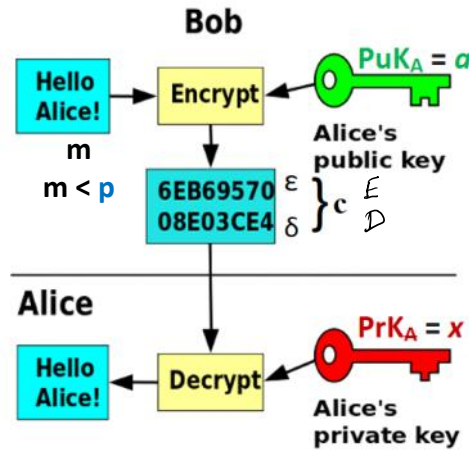
$$\text{V}=\text{Ver}(\text{PuK}_A, h, \sigma), \text{V} \in \{\text{True}, \text{False}\} \equiv \{1, 0\}$$



### Asymmetric Encryption - Decryption

$$c = \text{Enc}(\text{PuK}_A, m)$$

$$m = \text{Dec}(\text{PrK}_A, c)$$



## 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  $Z_p^* = \{1, 2, 3, \dots, p-1\}$  using condition.

Strong prime  $p=2q+1$ , where  $q$  is prime, then  $g$  is a generator of  $Z_p^*$  iff

$$g^q \neq 1 \pmod p \text{ and } g^2 \neq 1 \pmod p.$$

Declare **Public Parameters** to the network  $PP = (p, g)$ ;

$$p = 268435019; g = 2;$$

$$2^{28} - 1 = 268,435,455$$

$$\begin{aligned} &>> 2^{28} - 1 \\ &\text{ans} = 2.6844e+08 \\ &>> \text{int64}(2^{28} - 1) \\ &\text{ans} = 268435455 \end{aligned}$$

$$\text{PrK} = x \leftarrow \text{randi} \implies \text{PuK} = a = g^x \pmod p$$

Compatibility relations of modular arithmetic:

$$(a + b) \pmod p = (a \pmod p + b \pmod p) \pmod p.$$

$$(a * b) \pmod p = ((a \pmod p) * (b \pmod p)) \pmod p.$$

$$a^p \pmod p = (a \pmod p)^p \pmod p.$$

$p = 11 : 38 \pmod{11} = 5$

$$\begin{array}{r} 38 \\ - 33 \\ \hline 5 \end{array}$$

**Fermat little theorem:** If  $p$  is prime, then for any integer  $a$  holds  $a^p = a \pmod p$ .

1. We may assume that  $a$  is in the range  $0 \leq a \leq p - 1$ .

This is a simple consequence of the laws of modular arithmetic; we are simply saying that we may first reduce  $a$  modulo  $p$  since

$$a^p \pmod p = (a \pmod p)^p \pmod p.$$

2. It suffices to prove that for  $a$  in the range  $1 \leq a \leq p - 1$ .

Indeed, if the previous assertion holds for such  $a$ , multiplying both sides by  $a$  yields the original form of the theorem.

$$a^{p-1} = 1 \pmod p = a^0 = 1 \pmod p.$$

$$a^p = a \pmod p / a^{-1} \pmod p$$

$$0 \pmod{(p-1)} = 0$$

$$a^p \cdot a^{-1} = a \cdot a^{-1} \pmod p$$

$$a^{p-1} = 1 \pmod p = a^0 = 1$$

$$\frac{p-1}{p-1} \mid \frac{p-1}{1}$$

0 exponent is equivalent to (p-1) exponent  $\therefore p-1 \pmod{(p-1)} = 0$

$$s = xh + i$$

$$g^s \pmod p = g^{xh+i} \pmod p = g^{(xh+i) \pmod{(p-1)}} \pmod p.$$

$$s = (xh+i) \pmod{(p-1)}$$

### 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.

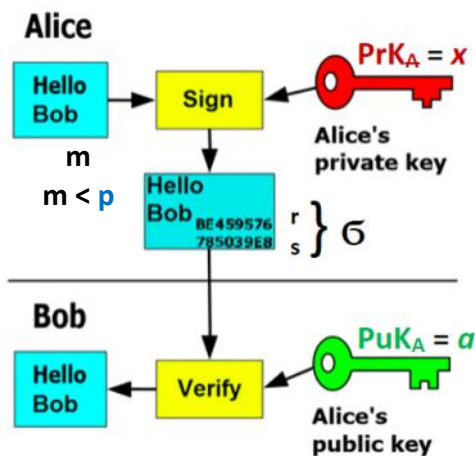
$$d\lg(g^x) = d\lg(a)$$

$$x \cdot d\lg g = d\lg(a)$$

$$x \cdot 1 = d\lg(a)$$

EC Gamal sign.  $\rightarrow$  Digital Signature Alg. (DSA) NSA

$\rightarrow$  Elliptic Curve DSA - ECDSA Certicom



Signature creation for message  $M \gg p$ .

1. Compute decimal h-value  $h=H(M)$ ;  $h < p$ .
  2. Generate  $i = \text{int64}(\text{randi}(p-1)) \% \text{ such that } \text{gcd}(i, p-1) = 1$ .
  3. Compute  $i^{-1} \pmod{(p-1)}$ . You can use the function  $\gg i\_m1 = \text{mulinv}(i, p-1)$ ;  $\% \text{ mod}(i \cdot i\_m1, p-1) = 1$
  4. Compute  $r = g^i \pmod p$ .
  5. Compute  $s = (h - xr)i^{-1} \pmod{(p-1)}$ .  $\frac{i}{i} = 1$
  6. Signature on h-value  $h$  is  $\sigma = (r, s)$ .  $1 \pmod{(p-1)} = 1$
- $\text{Sign}(x, h) = \sigma = (r, s)$

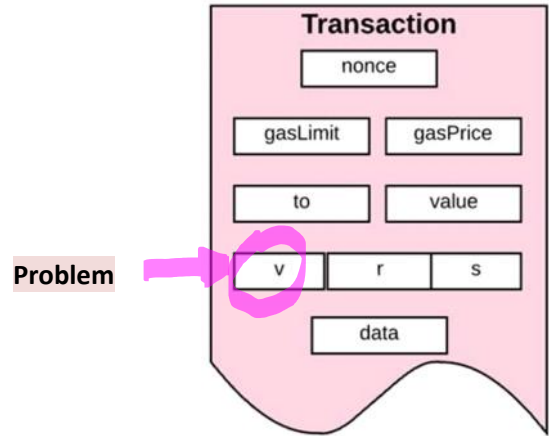
$$\gg m \cdot r = \text{mod}(-x \cdot r, p-1)$$

$$\gg \text{mod}(x \cdot r - m \cdot r, p-1) = 0$$

```

>> 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

```



$$T_x = \text{'nonce || gasLimit || gasPrice || to || value || data'}$$

$$h = H(T_x) \xrightarrow{\text{green arrow}} \sigma = (r, s) = \text{Sign}(PrK, h)$$

### 1. Signature creation

To sign any finite message  $M$  the signer performs the following steps using public parameters  $PP$ .

- Compute  $h=H(M)$ .
- Choose a random  $i$  such that  $1 < i < p - 1$  and  $\text{gcd}(i, p - 1) = 1$ .
- Compute  $i^{-1} \text{ mod } (p-1)$ :  $i^{-1} \text{ mod } (p-1)$  exists if  $\text{gcd}(i, p - 1) = 1$ , i.e.  $i$  and  $p-1$  are relatively prime.  
 $i^{-1}$  can be found using either Extended Euclidean algorithm or Euler theorem or ....

```

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

```

- Compute  $r=g^i \text{ mod } p$
- Compute  $s=(h-xr)i^{-1} \text{ mod } (p-1) \rightarrow h= xr+is \text{ mod } (p-1)$

Signature  $\sigma=(r, s)$

$$\begin{aligned}
 s &= (h - xr) \cdot i^{-1} \quad / \quad i \\
 s \cdot i &= (h - xr) \cdot \cancel{i^{-1}} \cdot i \\
 h - xr &= s \cdot i \xrightarrow{\text{green arrow}} h = xr + i \cdot s \quad \left. \vphantom{h = xr + i \cdot s} \right\} \text{ mod } p
 \end{aligned}$$

### 2. 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  $V1=g^h \text{ mod } p$  and  $V2=a^r r^s \text{ mod } p$ , 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

$$h = xr + is \text{ mod } (p-1)$$

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

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

**V1**
**(a)**
**(r)**
**V2**

Till this [place

$$PP = (P, g)$$

So:  $z \leftarrow \text{randi}(p-1)$   
 $v = g^z \bmod p$

{ Dear B I am A  
 and I am sending  
 you my  $PuK=v$  }

B: Believes that  
 $PuK=v$  is of A

$m = \text{'Bob get out'}$

$$\sigma = \text{Sign}(z, m) = (r, s)$$

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

B: verifies the signature  $\sigma$   
 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 Infrastructure 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

$PuK_A$

$$CA: PrK_{CA}; PuK_{CA}$$

$$\text{Sign}(PrK_{CA}, PuK_A || \text{Data}_A) = \sigma_A$$

$$\text{Cert}_A = \sigma_A, PuK_A, \text{Data}_A$$

$PuK_A \mid \text{Cert}_A$

$$B: \text{Ver}(PuK_{CA}, \text{Cert}_A) = \text{True}$$

Is sure that  $PuK_A$  is of A

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

<https://CertificationAuthority,Trusted.com>

<https://certicom.com>

```

>> p= int64(268435019)    >> i =int64(randi(p-1))    >> r=mod_exp(g,i,p)        >> g_h=mod_exp(g,h,p)
p = 268435019            i = 201156232            r = 172536234            g_h = 241198023
>> g=2;                  >> gcd(i,p-1)             >> hmxr=mod(h-x*r,p-1)    >> V1=g_h
>> x =int64(randi(p-1))  ans = 2                  >> s=mod(hmxr*i_m1,p-1)   V1 = 241198023
x = 65770603            >> i =int64(randi(p-1))  >> s=mod(hmxr*i_m1,p-1)   >> a_r=mod_exp(a,r,p)
>> a=mod_exp(g,x,p)     i = 35395315            s = 44575091            a_r = 49998673
a = 232311991          >> gcd(i,p-1)           >> r_s=mod_exp(r,s,p)    >> r_s = 111993804
>> M='Hello Bob...'    ans = 1                  >> V2=mod(a_r*r_s,p)     >> V2 = 241198023
M = Hello Bob...      >> i_m1=mulinv(i,p-1)    >> V2=mod(a_r*r_s,p)     >> V2 = 241198023
>> h=hd28(M)           i_m1 = 192754179
h = 150954921         >> mod(i*i_m1,p-1)      >> V2=mod(a_r*r_s,p)     >> V2 = 241198023
                        ans = 1

```

### 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  $m \bmod p = m$ .

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

$B$ :  $i \leftarrow \text{randi}(\mathcal{I}_p^*)$   
 $E = m \cdot a^i \bmod p$   
 $D = g^i \bmod p$   
 $C = (E, D)$   $\rightarrow$   $A$ : is able to decrypt  $C = (E, D)$  using her  $PrK_A = x$ .

$$(-x) \bmod (p-1) = (0-x) \bmod (p-1) = 1. D^{-x \bmod (p-1)} \bmod p$$

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

$(p-1) \bmod (p-1) = 0$  since  $\begin{matrix} -p-1 & (p-1) \\ p-1 & 1 \\ \hline 0 \end{matrix}$

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

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

$\gg D^{-x} = \text{mod\_exp}(D, p-1-x, p-1)$

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



$D^{-x} \bmod p$  computation using Fermat theorem:

If  $p$  is prime, then for any integer  $a$  holds  $a^{p-1} = 1 \bmod p$ .

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

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

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

Correctness

$$\text{Enc}(\text{PK}_A = a, i, m) = c = (E, D) = (E = m \cdot a^i \bmod p; D = g^i \bmod p)$$

$$\begin{aligned} \text{Dec}(\text{PK}_A = x, c) &= E \cdot D^{-x} \bmod p = m \cdot a^i \cdot (g^i)^{-x} \bmod p = \\ &= m \cdot \underbrace{(g^x)^i}_a \cdot g^{-ix} = m \cdot g^{xi} \cdot g^{-ix} = m \cdot g^{xi - ix} \bmod p = m \cdot g^0 \bmod p = \\ &= m \cdot 1 \bmod p = m \bmod p = m = 111222 \end{aligned}$$

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: 8 bits per char.

$$\frac{2048}{8} = 256 \text{ char.}$$

Decryption is correct if  $m < p$ .

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

1-st encryption:

$$\begin{aligned} i_1 &\leftarrow \text{rand}_i(\mathcal{L}_p^*) \\ E_1 &= m \cdot a^{i_1} \bmod p \\ D_1 &= g^{i_1} \bmod p \end{aligned} \left. \vphantom{\begin{aligned} i_1 \\ E_1 \\ D_1 \end{aligned}} \right\} c_1 = (E_1, D_1)$$

2-nd encryption

$$\begin{aligned} i_2 &\leftarrow \text{rand}_i(\mathcal{L}_p^*) \\ E_2 &= m \cdot a^{i_2} \bmod p \\ D_2 &= g^{i_2} \bmod p \end{aligned} \left. \vphantom{\begin{aligned} i_2 \\ E_2 \\ D_2 \end{aligned}} \right\} c_2 = (E_2, D_2)$$

$c_1 \neq c_2$

ElGamal

$$D_1 = g^{c_1} \text{ mod } p \quad \downarrow \quad D_2 = g^{c_2} \text{ mod } p$$

$c_1 \neq c_2$       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.

**AKAP: Asym.Enc & Digital Sign.**

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

1. Parties must agree on common symmetric secret key  $k$ .  
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)  $k \leftarrow \text{rand}_i(2^{128})$   
 $i_k \leftarrow \text{rand}_i(2^{128})$

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

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)  $Sign(PrK_A = x, h) = \tilde{\sigma} = (r, s)$

$\frac{c, G}{\tilde{\sigma}, PuK_A, Cert_A}$

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

1.2. Verify if  $\tilde{\sigma}$  on  $h = H(G)$  is valid?

$h' = H(G)$

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

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 recommended to prevent so called  
chosen ciphertext attacks - CCA: it is most strong attack  
but most complex in realization.