Mid Term Exam (MTE) will be held on **30-th of March, at 13:30 contactly in 506** a. contactly.
Please participate with your own computers with installed Octave and my .m files.
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
In section 'Cryptography':
Cryptography (imimsociety.net)

Please register to the site and after that you receive 10 Eur virtual money to purchase the problems.
For registration you should input the first 2 letters of your Surname and full Name, e.g. John Smith
Should register as **Sm John**.

Please purchase the only one problem at a time.

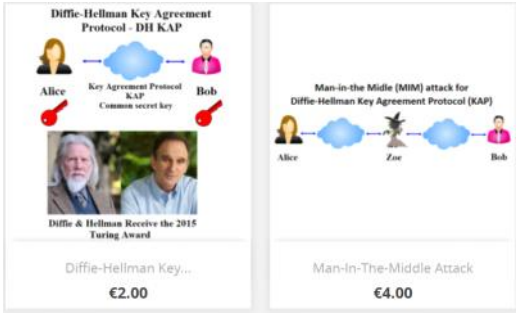
If the solution is successful then you are invited to press the green button [**Get reward**].
No any other declaration about the solution results is required.
If the solution failed, then you must press the button [**Go Back**] on the **top-left** side.

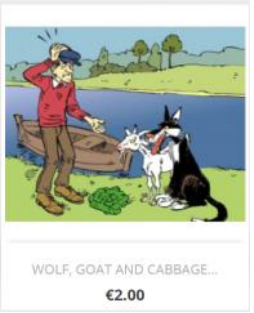
Then 'Knowledge bank' will pay you the sum twice you have paid.
So, if the initial capital was 10 Eur of virtual money and you buy the problem of 2 Eur, then if the solution is correct your budget will increase up to 12 Eur.

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.





Cryptography:
Information confidentiality, integrity, authenticity & person identification

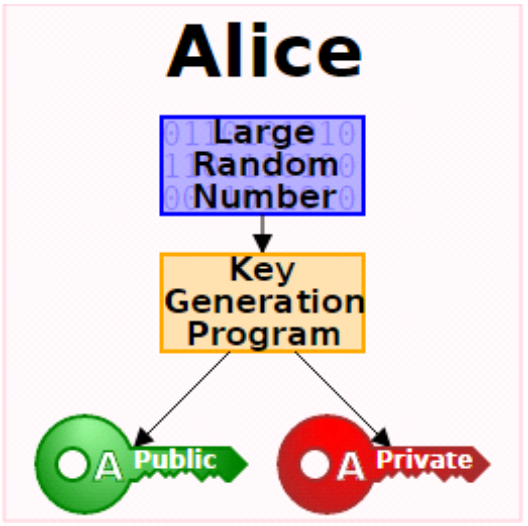
Symmetric Cryptography ---------- Asymmetric Cryptography

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.

Asymmetric - Public Key Cryptography = PKC



PrK and **PuK** are related
PuK = **F**(**PrK**)
F is **one-way function** - OWF
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.

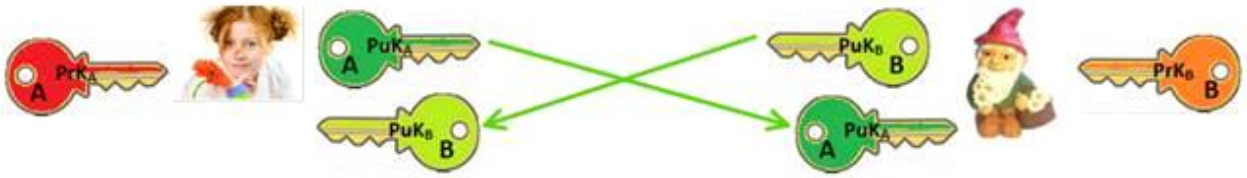
PrK = ***x*** <-- **randi** ==> **PuK** = ***a*** = g^x **mod p**
Public Parameters PP = **(p, g)**

Threaths of insecure PrK generation

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

$p \sim 2^{2048} \Rightarrow |p| \cong 2048$ bits
$p \sim 2^{28} \Rightarrow |p| \cong 28$ bits

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

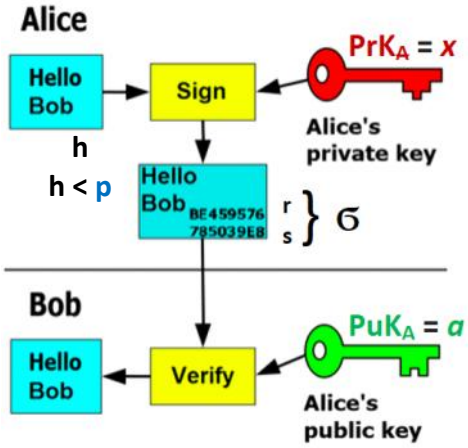


Message m < p

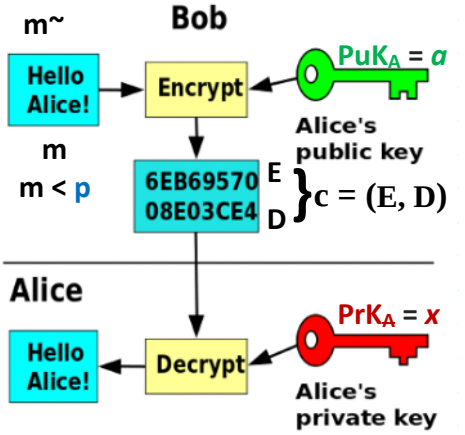
Asymmetric Signing - Verification
Sign(PrK$_A$, h) = σ = (r, s)
V=Ver(PuK$_A$, h, σ), V$\in$ {True, False} $\equiv$ {1, 0}

Asymmetric Encryption - Decryption
c=Enc(PuK$_A$, m)
m=Dec(PrK$_A$, c)

Alice PrK$_A$ = x m~ Bob



Authenticity



Confidentiality

ElGamal Cryptosystem

1.Public Parameters generation $PP = (p, g)$.
Generate strong prime number **p**: >> p=genstrongprime(28) % strong prime of 28 bit length
Strong prime number ***p***: defines the set Z_p^*= {1, 2, 3, …, **p**-1}, where multiplication operations **mod p** are defined. $Z_p^*$is an algebraic group where division operations are defined as well.
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

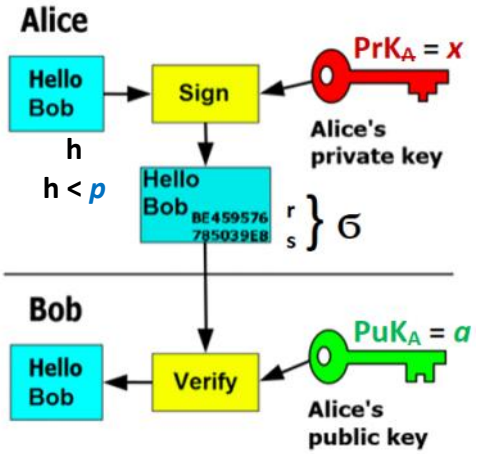
PrK = *x* <-- randi ==> PuK = *a* = g^x mod p

Compatibility relations of modular arithmetic:
(a + b) mod p = (a mod p + b mod p) mod p. >> mod(a+b,p)
(a * b) mod p =((a mod p) * (b mod p)) mod p. >> mod(a*b,p)
a^e mod p = (a mod p)^e mod p. **>> mod_exp(a,e,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 invented 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.

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



Authenticity

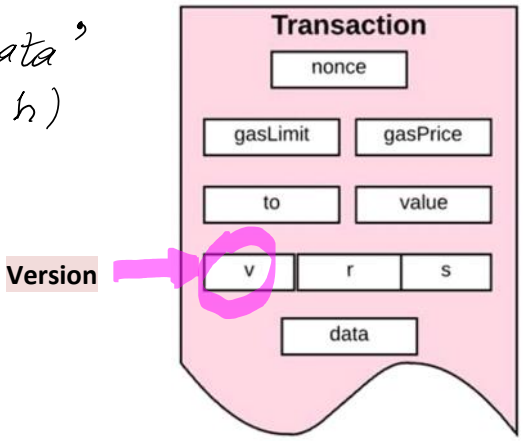
Signature creation for message ***M*** >> ***p***.

1. Compute decimal h-value ***h***=***H***(***M***); ***h***<***p***: SHA-256.
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
   >> gcd(i, p-1)
   ans = 1
   >> i_m1=mulinv(i, p-1);
1. Compute ***r*** = g^i **mod** ***p***.
2. Compute ***s*** = (***h*** - ***x***r)i^{-1} mod (***p***-**1**).
3. Signature on h-value ***h*** is σ = (***r***, ***s***)
   Sign(***x***, ***h***) = σ = (***r***, ***s***).

T_x = "nonce || gasLimit || gasPrice || to || value || data"

h = H(T_x) → σ = (r, s) = Sign(PrK, h)

GasLimit is evaluated by Wey.
1 Wey = 10^{-18} Eth --> 1 Eth = 10^{18} Wey.
Usuausally 1Gas is thousands of Wey's.



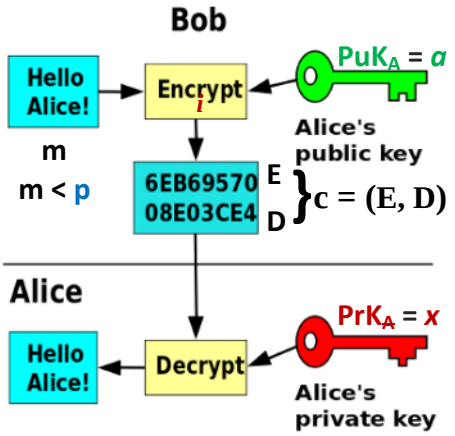
Asymmetric Encryption-Decryption: El-Gamal Encryption-Decryption

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***.

27 mod 54 = 27
27 mod 21 = 6 ≠ 27

A : PuK_A = a → B : is able to encrypt m to A: m < p



Confidentiality

B: $i \leftarrow randi(p-1)$

$E = m \cdot a^i \bmod p$

$D = g^i \bmod p$

$c = (E, D) \longrightarrow A$

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

$-\frac{p-1}{p-1}\ \Big|\frac{p-1}{1}$

0

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

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

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

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

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

```
>> p=int64(268435019)
p = 268435019
>> g=2
g = 2
>> x=int64(randi(p-1))
x = 144332522
>> a=mod_exp(g,x,p)
a = 99915647
```

```
% Verification x+(-x) = 0 mod(p-1)
>> mx=mod(-x,p-1)
mx = 124102496
>> mod(x+mx,p-1)
ans = 0
```

```
% Verification D^x * D^-x =1 mod p
>> i=int64(randi(p-1))
i = 17305576
>> D=mod_exp(g,i,p)
D = 43916598
>> D_x=mod_exp(D,x,p)
D_x = 251866400
>> D_mx=mod_exp(D,mx,p)
D_mx = 64836527
>> mod(D_x*D_mx,p)
ans = 1
```

Encryption:

```
>> m=111222;
>> i=int64(randi(p-1))
i = 17305576
>> a_i=mod_exp(a,i,p)
```

Decryption: m=111222

```
>> D_mx=mod_exp(D,mx,p)
D_mx = 64836527
>> mm=mod(E*D_mx,p)
```

```
>> E=mod(m*a_i,p)

>> D=mod_exp(g,i,p)
D = 43916598
```

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 $m < p$

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.

$PP = (p, g)$

$\mathcal{L}_0$: $z \leftarrow randi(p-1)$

$v = g^z \bmod p$

{ Dear B I am A and I am sending you my $PuK = v$ } → B: believes that $PuK = V$ is of A

$m =$ 'Bob get out'

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

$m, \sigma = (r, s)$ →

B: verifies the signature σ 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

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

$Sign(PrK_{CA}, PuK_A || Data_A) =$

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

ID

PuK_A

$Sign(PrK_{CA}, PuK_A || Data_A) = \sigma_A$.

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

PuK_A | $Cert_A$

$Cert_A$

$\mathcal{B}$: $Ver(PuK_{CA}, Cert_A) = True$

Is sure that PuK_A is of $\mathcal{A}$

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

https://CertificationAuthority.Trusted.com

https://certicom.com

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

1-st encryption:

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

$E_1 = m \cdot a^{i_1} \bmod p$

$D_1 = g^{i_1} \bmod p$

$\} C_1 = (E_1, D_1)$

$i_1 \neq i_2$

2-nd encryption

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

$E_2 = m \cdot a^{i_2} \bmod p$

$D_2 = g^{i_2} \bmod p$

$\} C_2 = (E_2, D_2)$

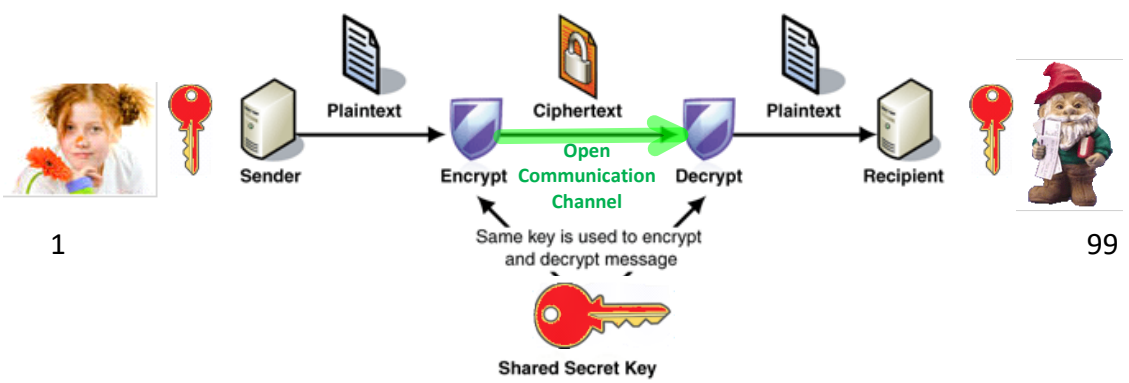
$c_1 \neq c_2$

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 --> Alan Turing (the same fragment in message)



1

99

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.
To 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
$E_k(M) = AES_k(M) = C$

3) Signs ciphertext C

3.1) $h = H(C)$
3.2) $Sign(PrK_A = x, h) = \tilde{G} = (r, s)$

$c, C \rightarrow$
$\tilde{G}, PuK_A$
$Cert_A$

1.1. Verify if PuK_A and $Cert_A$ are valid?
1.2. Verify if $\tilde{G}$ on $h = H(C)$ is valid?
$h = H(C)$
$Ver(PuK_A, \tilde{G}, h') = True$
2. $Dec(PrK_B, c) = k$
3. $D_k(C) = AES_k(C) = M$.

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.