Mid Term Exam (MTE) will be held on 14-th of April at 17:00 part of you contactly, part on-line.
The information concerning participation I will send to you personally by e-mail.
Participating on-line should have your personal computers with installed Octave and my .m files.
Otherwise, I redirect you to the computer class where you will have MatLab installed.
Then you should download my .m files on class computers.
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.
The registration should contai 2 frirst letters of your Surname and full your Name, e.g. **Sm John** for **John Sm**ith.

Please purchase the only one problem at a time.

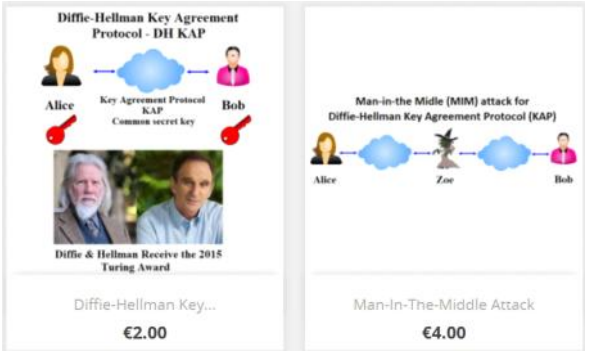
If the solution is successful then you are invited to press the green button [Get reward].
No any other declaration about the solution results is required.

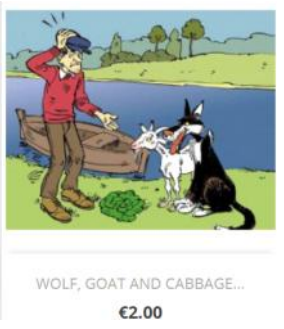
Then 'Knowledge bank' will pay you the sum twice you have paid.
So, if the initial capital was 10 Eur of virtual money and you buy the problem of 2 Eur, then if the solution is correct your budget will increase up to 12 Eur.

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

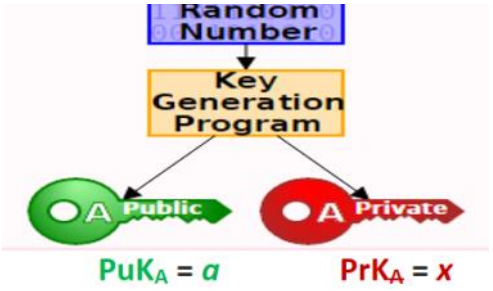
I advise you to try at first to solve the problem in 'Intellect' section to exercise the brains.
It is named as 'WOLF, GOAT AND CABBAGE TRANSFER ACROSS THE RIVER ALGORITHM'.
< https://imimsociety.net/en/home/15-wolf-goat-and-cabbage-transfer-across-the-river-algorithm.html>

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





PP = (p, g).
Strong prime number p in real cryptography is of order : $p \sim 2^{2048}$
Strong prime number p in our examples is of order: $p \sim 2^{28}$
>> p=genstrongprime(28)
Key generation



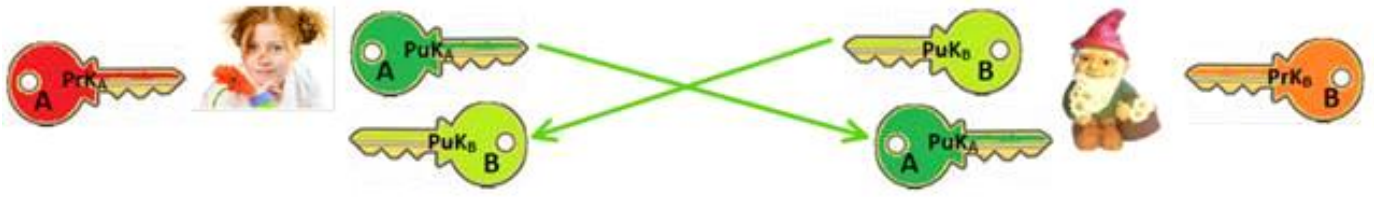
>> p=genstrongprime(28)

Key generation

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

1. Identification.

If person can prove that he/she knows **PrK** corresponding to his/her **PuK** without revealing any information about **PrK** then everybody can trust that he is communicating with person posessing (**PrK**, **Puk**) key pair. This kind of proof is named as ***Zero Knowledge Proof*** (**ZKP**) and plays a very important role in cryptography. It is very useful to realize identification, Digital Signatures and many other cryptographically secure protocols in internet. In many cryptographic protocols, especially in identification protocols **PrK** is named as **witness** and **PuK** as a **statement** for **PrK**. Every actor is having the corresponding key pair (**PrK$_A$**, **PuK$_A$**) and all **PuK** are exchanged between the users using open communication channel as indicated in figure below.
Let Bob is sure that **PuK$_A$** is indeed of Alice and wants to tell Alice that he intends to send her his photo with chamomile flowers dedicated for Alice. But he wants to be sure that he is communicating only with Alice itself and with nobody else. He hopes that at first Alice will prove him that she knows her secret **PrK$_A$** using **ZKP** protocol. In general, this protocol is named as **Identification protocol**, it is interactive and has 3 communications to exchange the following data named as ***commitment***, ***challenge*** and ***response***.



Registration phase: Bank generates **PrK$_A$** = **x** and **PuK$_A$** = **a** to Alice and hands over this data in smart card, or in other crypto chip in Alice's smart phone, or in software for Smart ID.

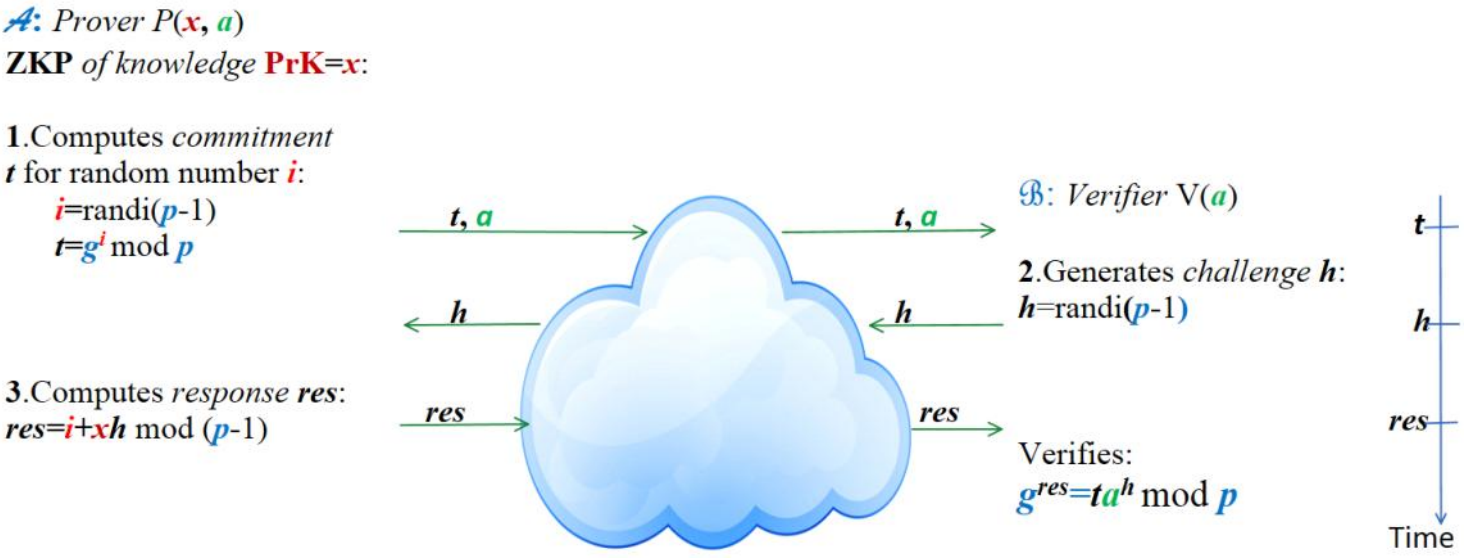
Schnorr Id Scenario: **Alice** wants to prove **Bank** that she knows her Private Key - **PrK$_A$** = **x** which corresponds to her Public Key - **PuK$_A$**= a not revealing **PrK$_A$**: Zero Knowledge Proof - **ZKP** Protocol execution between **Alice** and **Bank** has time limit.
Alice's computation resources has a limit --> protocol must be computationally effective.
PrK$_A$=x is called a **witness** and corresponding **PuK$_A$**=a=g^x **mod** p is called a **statement**.
This protocol is initiated by Alice and has the following three communications.

P(x, a) - **Prover** - **Alice** **V**(a) - **Verifier** - **Bank**

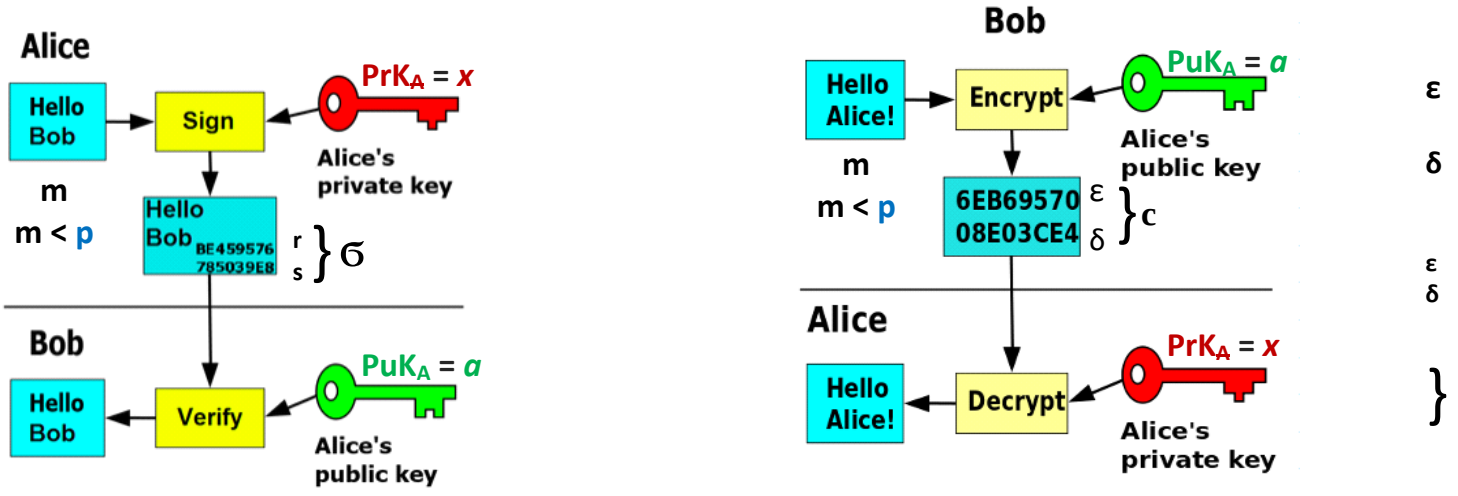
C:\Users\Eligijus\Documents\REKLAMA ZKP_Pasaka

Schnorr Identification: Zero Knowledge Proof - ZKP $PP = (p, g)$.

Schnorr Id is interactive protocol, but not recurent as it realized to prove the mirckle words.
Schnorr Id Scenario: **Alice** wants to prove **Bank** that she knows her Private Key - $PrK_A = x$ which corresponds to her Public Key - $PuK_A = a = g^x \bmod p$ not revealing $PrK_A = x$.



Correctness:
$g^{res} \bmod p = g^{i+xh \bmod (p-1)} \bmod p = g^i g^{xh} \bmod p = t(g^x)^h \bmod p = ta^h \bmod p$.



Schnorr Signature Scheme **(S-Sig).**

In general, to create a signature on the message of any finite length ***M*** parties are using cryptographic secure H-function (message digest).
In Octave we use H-function
>> hd28('…') % the input '…' of this function represents a string of symbols between the commas.
% the output of this function is decimal number having at most 28 bits.

Let ***M*** be a message in string format to be signed by **Alice** and sent to **Bob**: >> M='Hello Bob'
For signature creation **Alice** uses public parameters **PP=(*p*, *g*)** and
Alice's key pair is $PrK_A=x$, $PuK_A= a = g^x \bmod p$.

Alice chooses at random ***u***, 1<***u***<***p***-1 and computes first component ***r*** of his signature:

$$r=g^u \bmod p. \qquad (2.19)$$

Alice computes H-function value ***h*** and second component ***s*** of her signature:

$$h=H(M||r), \qquad (2.20)$$

$$s=u+xh \bmod (p-1). \qquad (2.21)$$

Alice's signature on ***h*** is **σ=(*r*, *s*)**. Then **Alice** sends ***M*** and **σ** to **Bob**.

After receiving ***M'*** and **σ**, **Bob** according to (2.20) computes ***h'***

$$h'=H(M'||r),$$

and verifies if

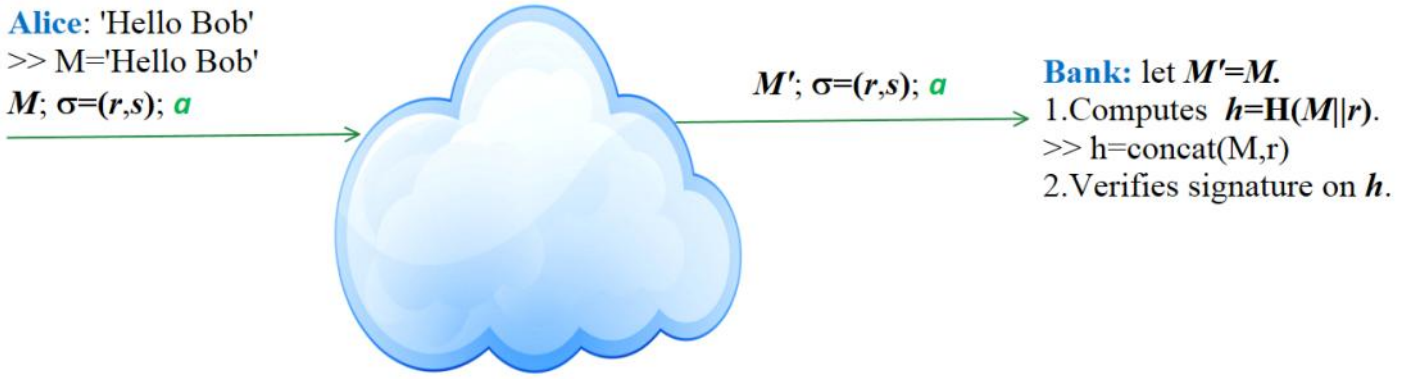
$$g^s \bmod p = ra^{h'} \bmod p. \qquad (2.22)$$

V1 V2

Symbolically this verification function we denote by

$$Ver(a,\sigma,h')=V\in\{True, False\}\equiv\{1, 0\}. \qquad (2.23)$$

This function yields ***True*** if (2.22) is valid if: $h=h'$ and $PuK_A= a =F(PrK_A)= g^x \bmod p$.
and: $M=M'$




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

>> x=int64(randi(p-1))
x = 89089011
>> a=mod_exp(g,x,p)
a = 221828624

>> m='Hello Bob'
m = Hello Bob
>> u=int64(randi(p-1))
u = 228451192
>> r=mod_exp(g,u,p)
r = 33418907
```

```
>> g_s=mod_exp(g,s,p)
g_s = 185672370
V1=g_s;
>> a_h=mod_exp(a,h,p)
a_h = 263774143
>>  V2=mod(r*a_h,p)
V2 = 185672370
```

```
>> cc=concat(m,r)
cc = Hello Bob33418907      % cc is a string type variable
>> cc=concat(m,'33418907')
cc = Hello Bob33418907
>> ccc=concat(m,'r')
ccc = Hello Bobr

>> h=hd28(cc)
h = 104824510
>>  s=mod((u+x*h),p-1)
s = 147250342
```

104824510