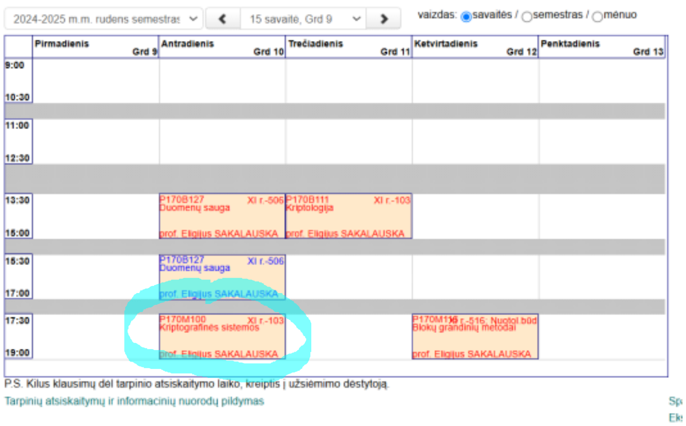
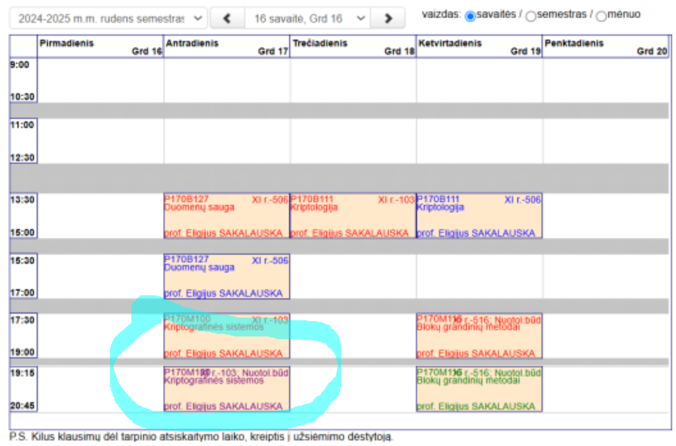
010_012 Schnorr-Id-Sign-NIZKP

Poster Reports (PR) Topics

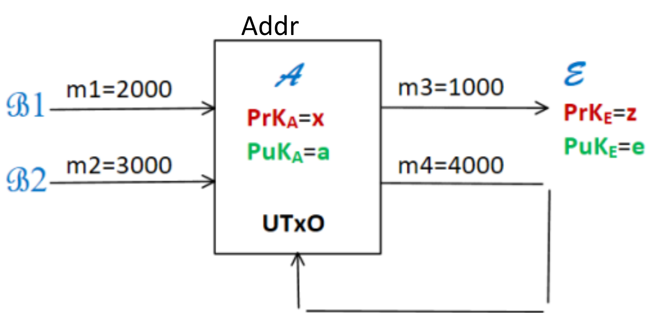
https://docs.google.com/document/d/1seFnKGesTFkSUDGiSXoJpS_MXTY4z2vh/edit?usp=sharing&ouid=111502255334918748828&rtpof=true&sd=true





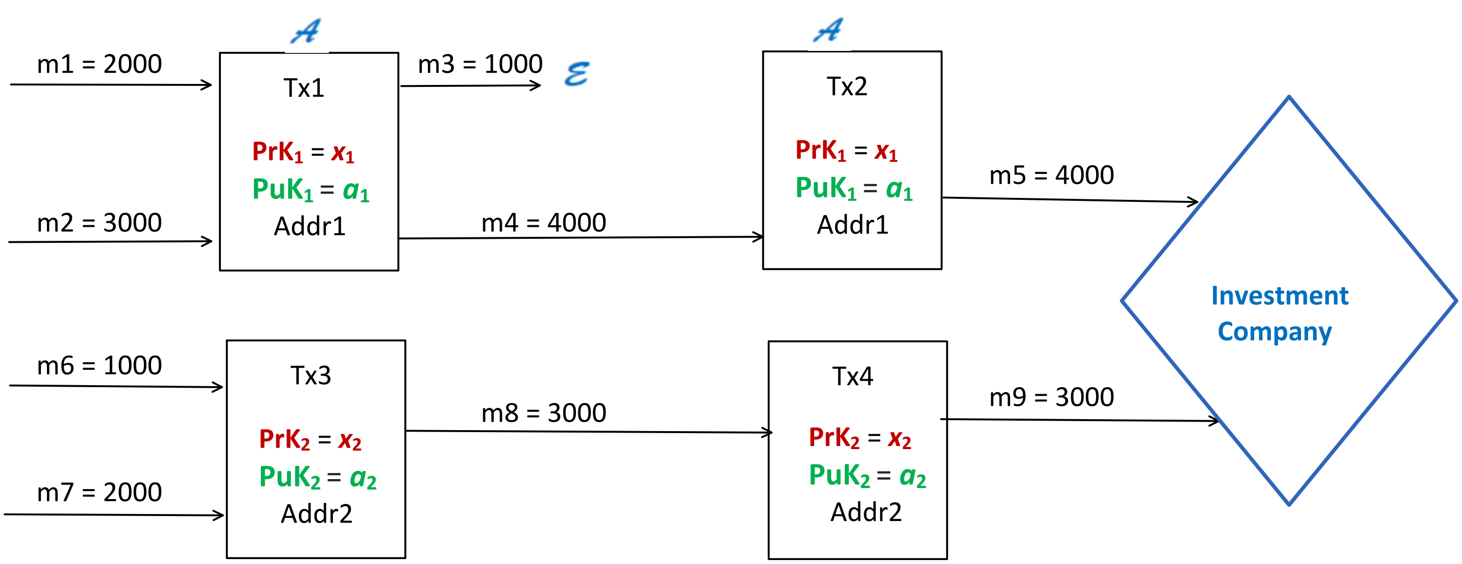
C:\Users\Eligijus\Documents\REKLAMA   ZKP_Pasaka

› Documents › 100 MOKYMAS   S-id, S-sig.pptx



$cert_A : CA(x_{CA}, a_{CA})$

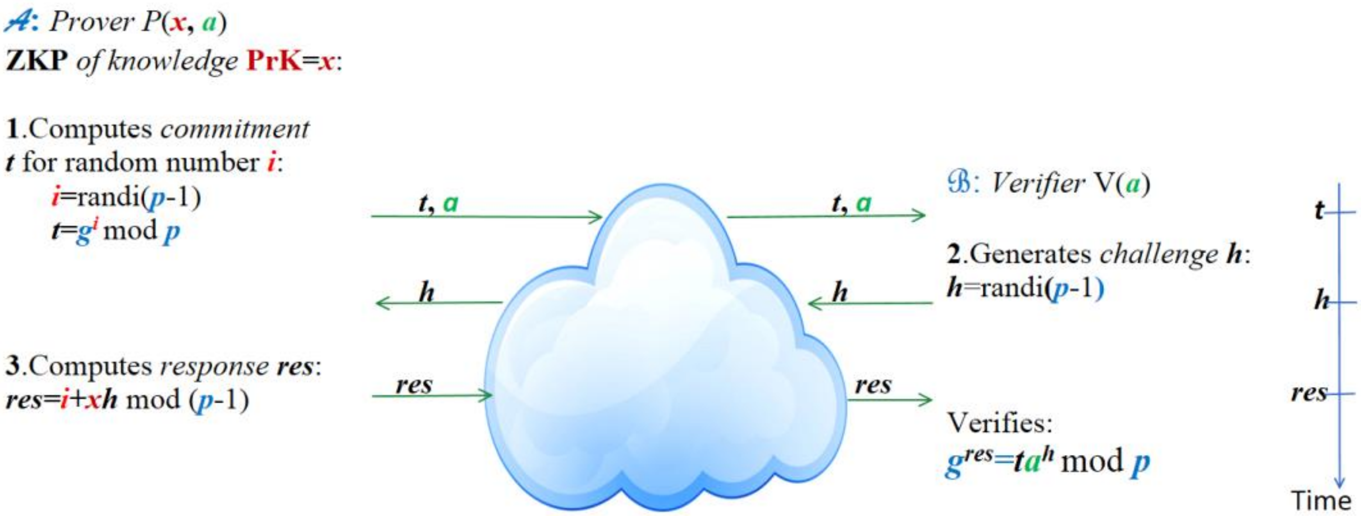
$Sign(x_{CA}, a, hData) = \sigma = (r_A, s_A)$



Public parameters: **PP** = (p, g);   **p=268435019; g=2;**

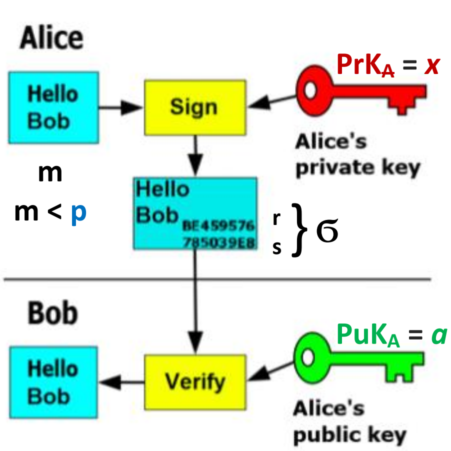
Schnorr Identification: Zero Knowledge Proof - ZKP   **PP** = (p, g).

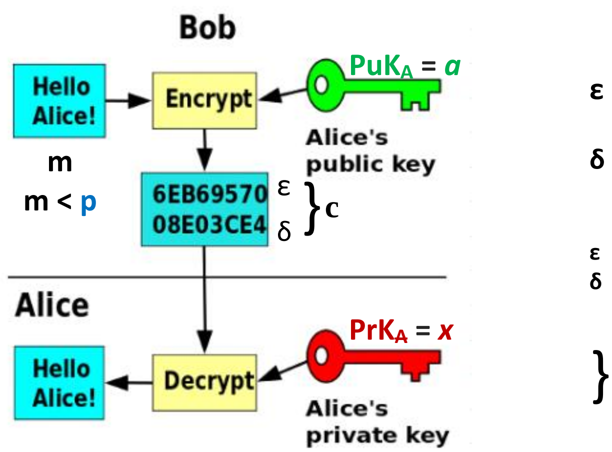
Schnorr Id is interactive protocol, but not recurent as it realized to prove the mirckle words.
Schnorr Id Scenario: **Alice** wants to prove **Bank** that she knows her Private Key - **PrK$_A$** = x which corresponds to her Public Key - **PuK$_A$**= a = g^x **mod** p not revealing **PrK$_A$**= x.



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 $\mathbf{PrK_A}=x$, $\mathbf{PuK_A}= a = g^x \bmod p$.

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

u <-- randi(p-1).

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

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

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

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

Alice's signature on h is $\sigma=(r, s)$. Then **Alice** sends M and σ to **Bob**.
Notice that it is infeasible to find x from (2.21), when s and h are given,
since there is 1 equation (*) and 2 unknowns u and x.

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

$$h'=\mathrm{H}(M'||r),$$

and verifies if

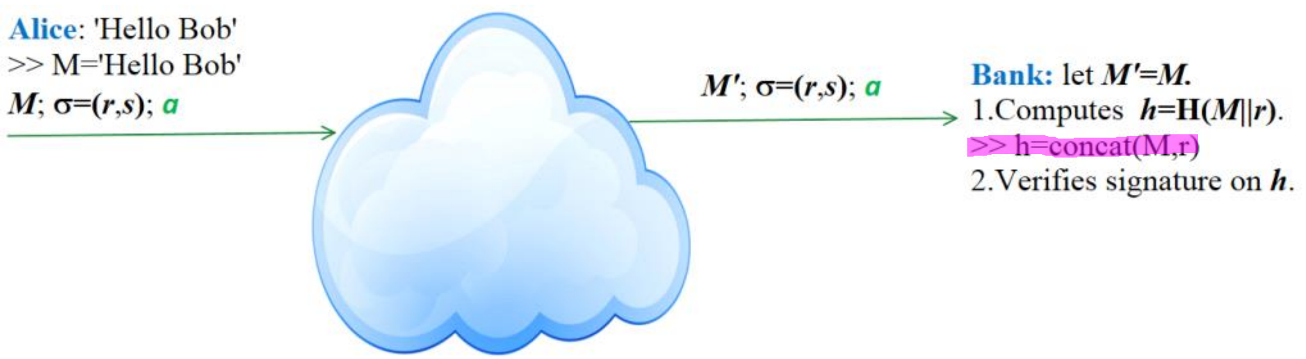
$$\underbrace{g^s \bmod p}_{\text{V1}} = \underbrace{ra^{h'} \bmod p}_{\text{V2}}. \qquad (2.22)$$

Symbolically this verification function we denote by

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

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

$h=\mathrm{H}(M||r)$ >> con=concat(M,r)
>> h=hd28(con)



Consequence --> Non-Interactive Zero Knowledge Proof - NIZKP.
Alice using the scheme of Schnorr Signature can prove a knowledge of any other secret in one or other way related with Discrete Exponent Function - DEF.
Let this secret is some integer i and then **Alice** using DEF computes so called **Statement** we denote by t for her secret i:

$$t = g^i \bmod p.$$

In this scenario Alice is called a ***Prover***.
Then **Alice** realizes a NIZKP of knowledge of i without revealing i by presenting this **Statement** t to the ***Verifier*** **Bob**.

A: $u \leftarrow \mathrm{randi}(\mathbb{Z}_{p-1})$; $\mathbb{Z}_{p-1}=\{0,1,2,\ldots,p-2\}$; $+,-,*,:$ mod $(p-1)$

$r = g^u \bmod p$ $\quad \sigma=(r,s), t, a \quad$ B: $h=H(a||t||r)$

$$r = g^u \bmod p$$
$$h = H(a \| t \| r)$$
$$s = u + ih \bmod (p-1)$$
$$\pi = (r, s)$$

$$\xrightarrow{\pi=(r,s),\ t,\ a}$$

B: $h = H(a \| t \| r)$

$$g^s = r \cdot t^h \bmod p$$

$$g^s = g^{u+ih \bmod (p-1)} \bmod p = g^u \cdot g^{ih} \bmod p = r \cdot (g^i)^h = r \cdot t^h \bmod p$$

Till this place

Fiat–Shamir heuristic allows to replace the interactive step 3 with a **non-interactive** random oracle access. In practice, we can use a cryptographic hash function instead.[7]

1. Lena wants to prove that she knows x such that $y \equiv g^x$ without revealing x.
2. Lena picks a random $v \in \mathbb{Z}_q^*$ and computes $t = g^v$.
3. Lena computes $c = H(g, y, t)$, where H is a cryptographic hash function.
4. Lena computes $r = v - cx \bmod \varphi(q)$. The resulting proof is the pair (t, r).
5. Anyone can use this proof to calculate c and check whether $t \equiv g^r y^c$.

If the hash value used below does not depend on the (public) value of *y*, the security of the scheme is weakened, as a malicious prover can then select a certain value *t* so that the product *cx* is known.[8]

Fiat–Shamir heuristic allows to replace the interactive step 3 with a **non-interactive** [random oracle](#) access. In practice, we can use a [cryptographic hash function](#) instead.[7]

1. Lena wants to prove that she knows
 such that
 without revealing
.
2. Lena picks a random
 and computes
.
3. Lena computes
, where
 is a cryptographic hash function.
4. Lena computes
. The resulting proof is the pair
.
5. Anyone can use this proof to calculate
 and check whether
.

If the hash value used below does not depend on the (public) value of *y*, the security of the scheme is weakened, as a malicious prover can then select a certain value *t* so that the product *cx* is known.[8]

From <https://en.wikipedia.org/wiki/Fiat%E2%80%93Shamir_heuristic>

Medziaga

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

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

```
>> 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
```

$\mathcal{A}$: **ZKP of knowledge** x:

$PrK_A = x =$ **randi(*p*-1)**

$PuK_A = a = g^x$ **mod** p

1.Computes **commitment**

t for random number i:

i=randi(p-1)

$t=g^i$ mod p

3.Computes **response** ***res***:

$res=i+xh$ mod (p-1)

$\xrightarrow{t,\ a}$ $\qquad$ $\xrightarrow{t,\ a}$

$\xleftarrow{h}$ $\qquad$ $\xleftarrow{h}$

$\xrightarrow{res}$ $\qquad$ $\xrightarrow{res}$

$\mathcal{B}$: $PuK_A = a$

2.Generates **challenge** h:

h=**randi(*p*-1)**

Verifies:

$g^{res}=ta^h$ **mod** p

t

h

res

Time

Alice: 'Hello Bob'

>> M='Hello Bob'

M; σ=(r,s); a

$\longrightarrow$

M'; σ=(r,s); a $\longrightarrow$

Bob: let $M'=M$.

1.Computes h=**H(*M*||*r*)**.

>> h=concat(M,r)

2.Verifies signature on h.