Security and Cryptography 2022

Mirosław Kutyłowski

XI. PRIVACY

Protection of personal data and GDPR

– declared as fundamental right in EU, but technically fundamental for cybersecurity

- identity theft e.g. for financial criminality
- mobbing, discrimination, social abuse
- lack of protection is a threat for economy and national security

–

Frameworks

– GDPR – EU and European Economic Area, adopted by many countries, some recognized as equivalent by EU

– Safe Harbour approach , Privacy Shield: https://www.privacyshield.gov

– California Consumer Privacy Act

– Schrems II verdict of *Court of Justice of the European Union*

– after Schrems II: major companies get policies approved by EU

– but: EU Whistleblowers Directive: protection of whistleblowers is declarative only

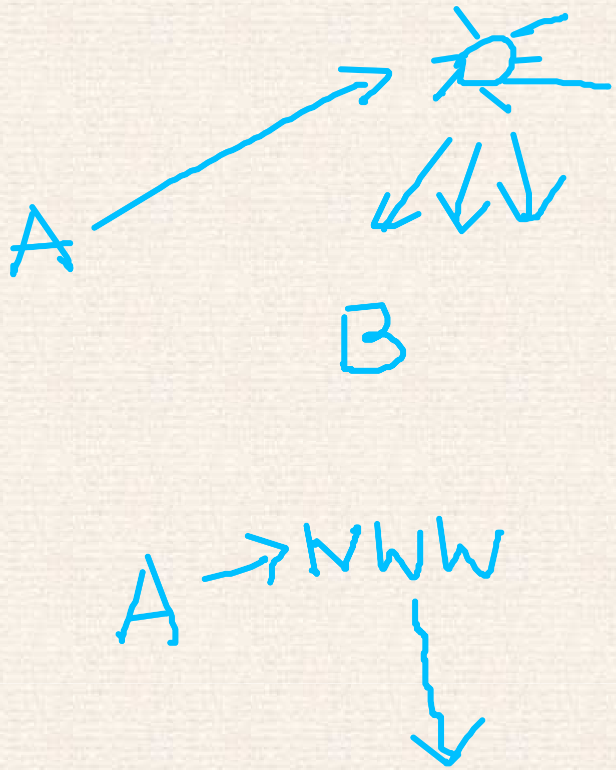
technology?

Privacy in communication

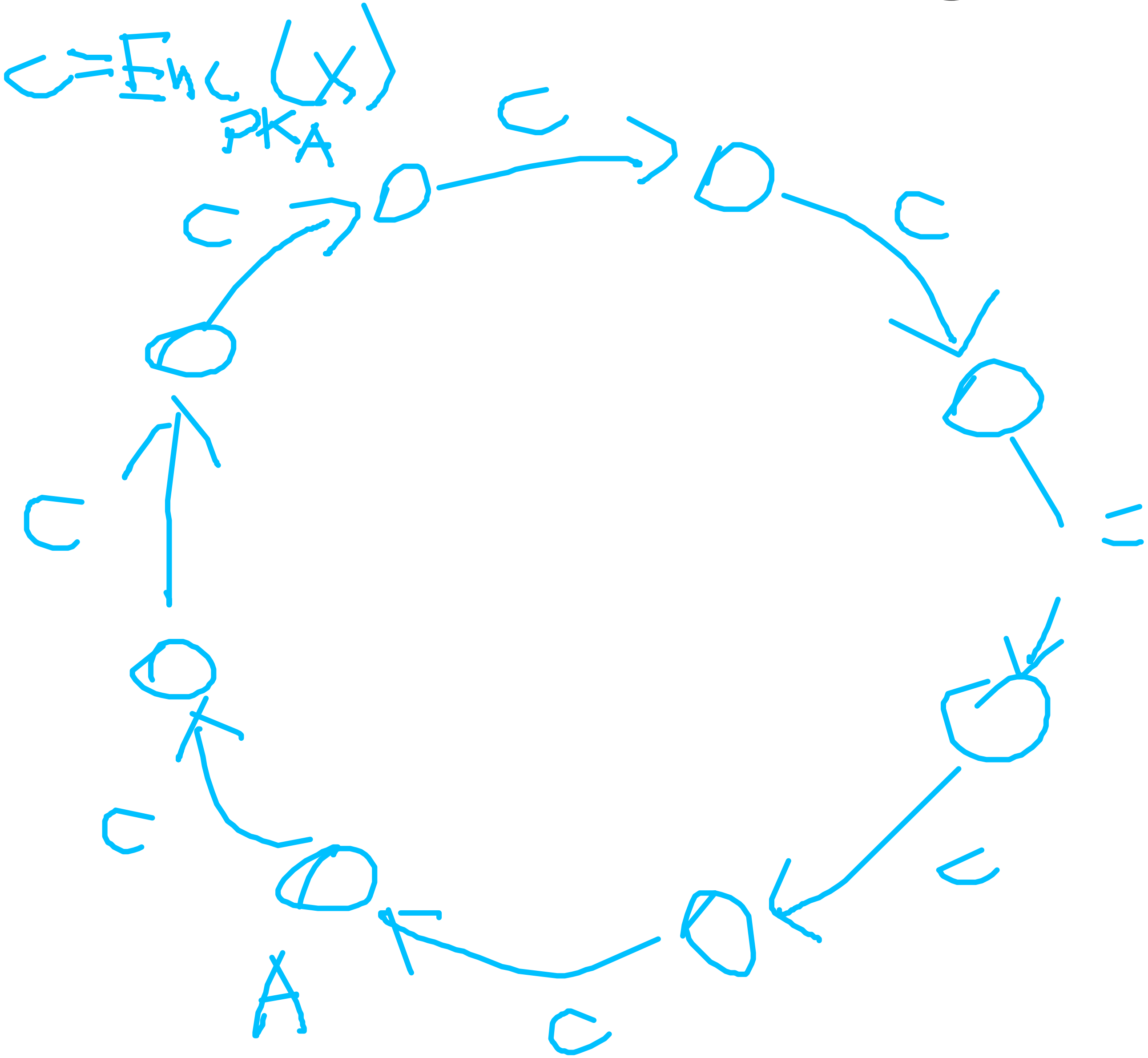
- one can hide payload communication
- it is not trivial how to hide **who is communicating with whom**
- this is a sensitive data

protection methods:

- broadcast channel
- token ring
- dining cryptographers -DC nets
- onion protocols and TOR



token ring mechanism



only A can understand c

with re-encryption, processing at a node

$c_2' = \text{Re-enc}(c_2)$

$c_3' = \text{Re-enc}(c_3)$

$c_1' = \text{Enc}(\text{new message})$

1) decrypt c_1, c_2, c_3 with own secret key

2) $c_1 \to T \to$ valid plaintext "it's for me"

$c_2, c_3 \to$ junk, ok, it is not for me

3) reencrypt and send something in c_1'

easy re-encryption: ElGamal

$$(PK^k \cdot m,\ g^k)$$

$$\downarrow$$

$$((PK^k \cdot m) \cdot PK^{\Delta},\ g^k \cdot g^{\Delta})$$

but: we can do it iff
we know recipients publickey PK

so it does not work this
way for anonymous communication

universal re-encryption based on ElGamal

ciphertext of m is a quadruple:

$$(A, B, C, D) = (PK^{k_1} \cdot m,\ g^{k_1},\ PK^{k_2},\ g^{k_2})$$

re-encrypted:

$$\left(A \cdot C^{\Delta},\ B \cdot D^{\Delta},\ C^{\Pi},\ D^{\Pi}\right)$$

Δ, Π – random

this is also a ciphertext of m:

$$\left(PK^{k_1 + \Delta \cdot k_2} \cdot m,\ g^{k_1 + \Delta \cdot k_2},\ PK^{\Pi \cdot k_2},\ g^{\Pi \cdot k_2}\right)$$

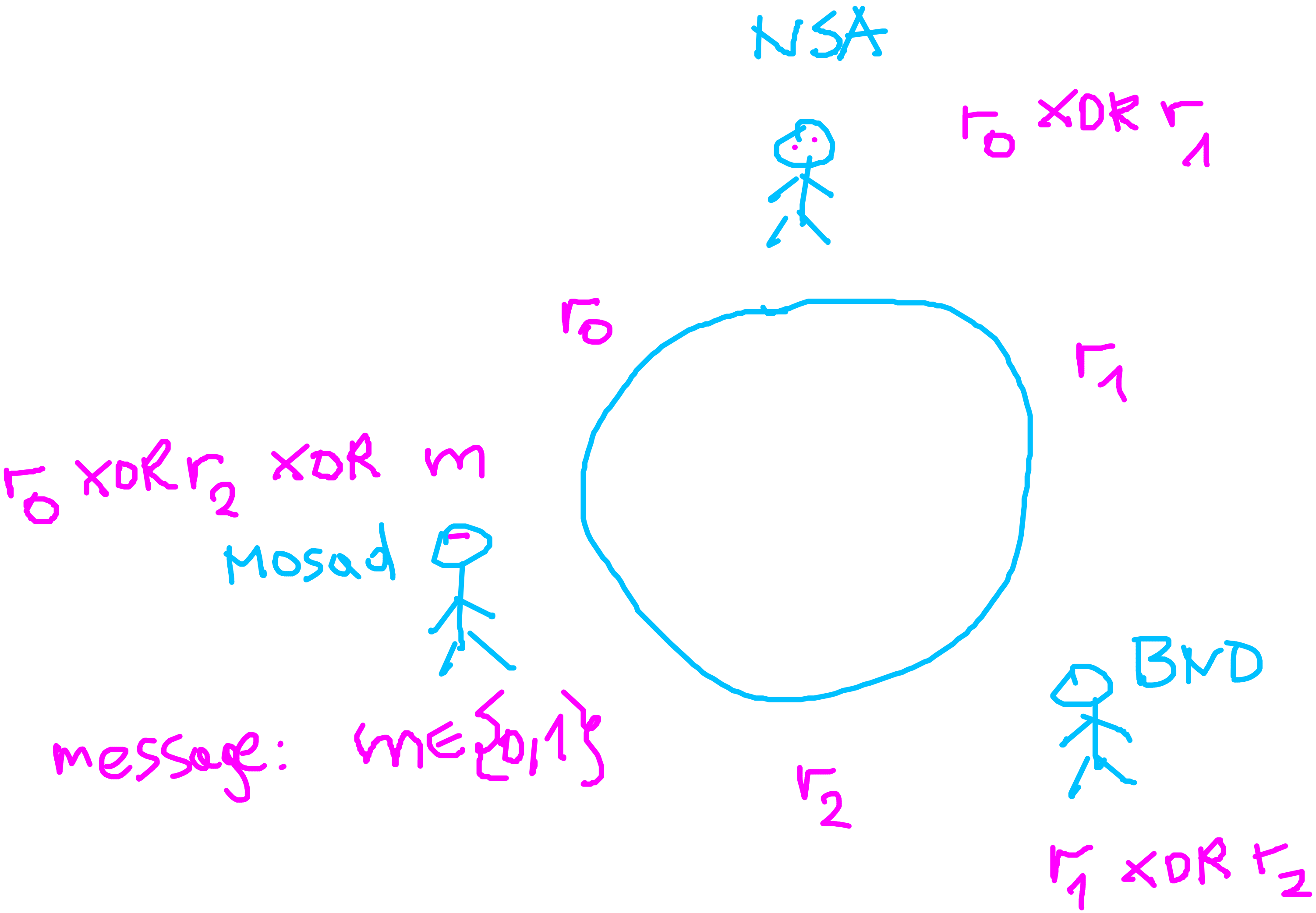
Dining Cryptographers Protocol

- protecting the source of a 1-bit message. An unknown user sends a 1-bit message.
- protocol for 3 cryptographers sitting at around table:
  1. each pair of neighbors establish a shared bit at random
  2. each cryptographer that is not transmitting computes XOR of the bits shared with the neighbors,
  3. the sender computes the same XOR but swaps it if the bit transmitted is 1
  4. each cryptographer reveals his result
  5. the message is the XOR of the bits published:
     - if the message is 0, then each shared bit occurs twice:

$$(b_{\mathrm{AB}} \oplus b_{\mathrm{BC}}) \oplus (b_{\mathrm{BC}} \oplus b_{\mathrm{CA}}) \oplus (b_{\mathrm{CA}} \oplus b_{\mathrm{AB}}) = 0$$

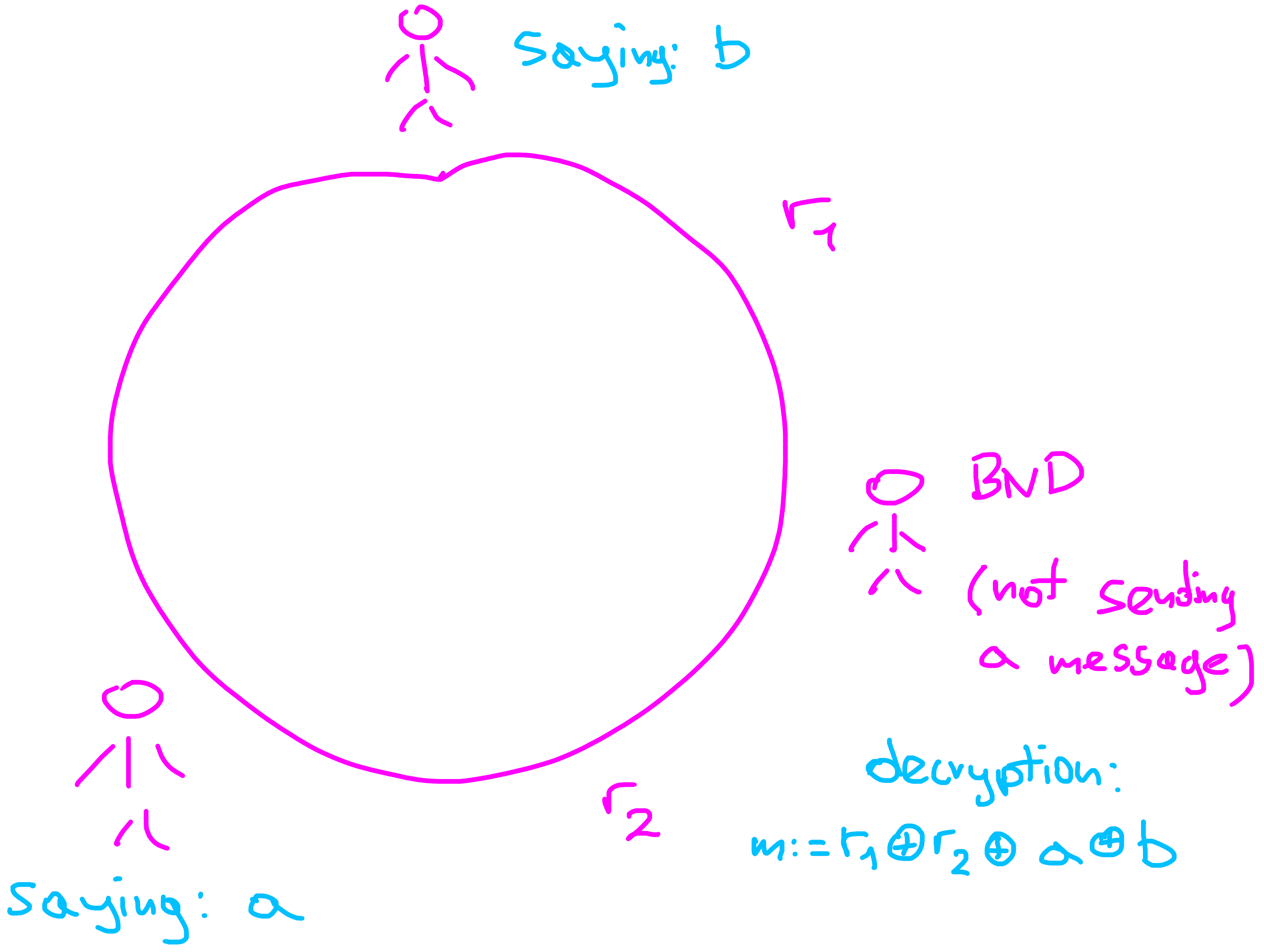
     - otherwise, one of the bits is swapped: e.g. we have

$$(b_{\mathrm{AB}} \oplus b_{\mathrm{BC}}) \oplus (b_{\mathrm{BC}} \oplus b_{\mathrm{CA}} \oplus 1) \oplus (b_{\mathrm{CA}} \oplus b_{\mathrm{AB}}) = 1$$



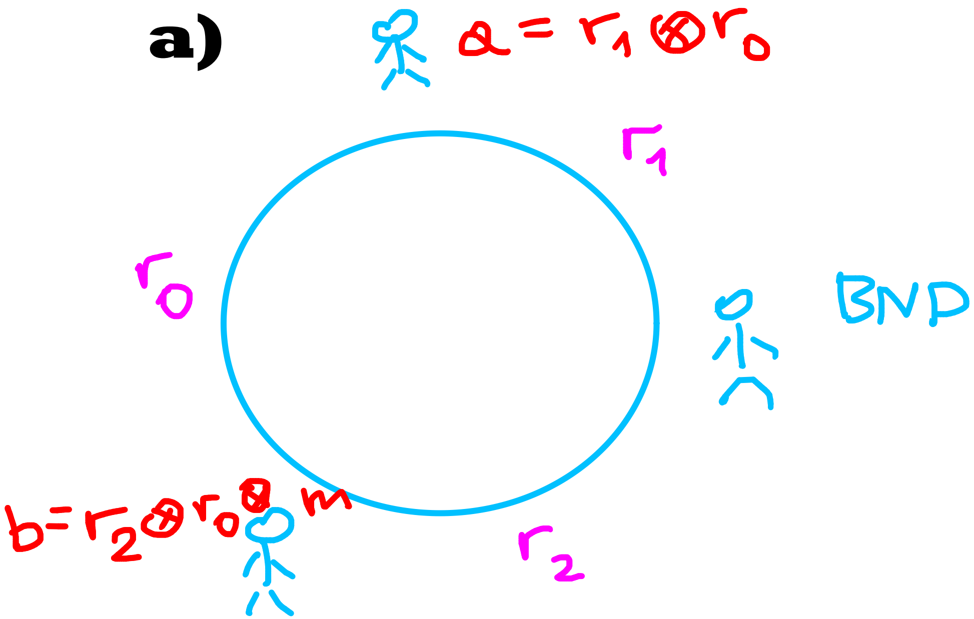
NSA
r0 XOR r1
r0
r1
r0 XOR r2 XOR m
Mosad
BND
message: m∈{0,1}
r2
r1 XOR r2

the point of view of a single participant:



possible options from the point of view of BND:

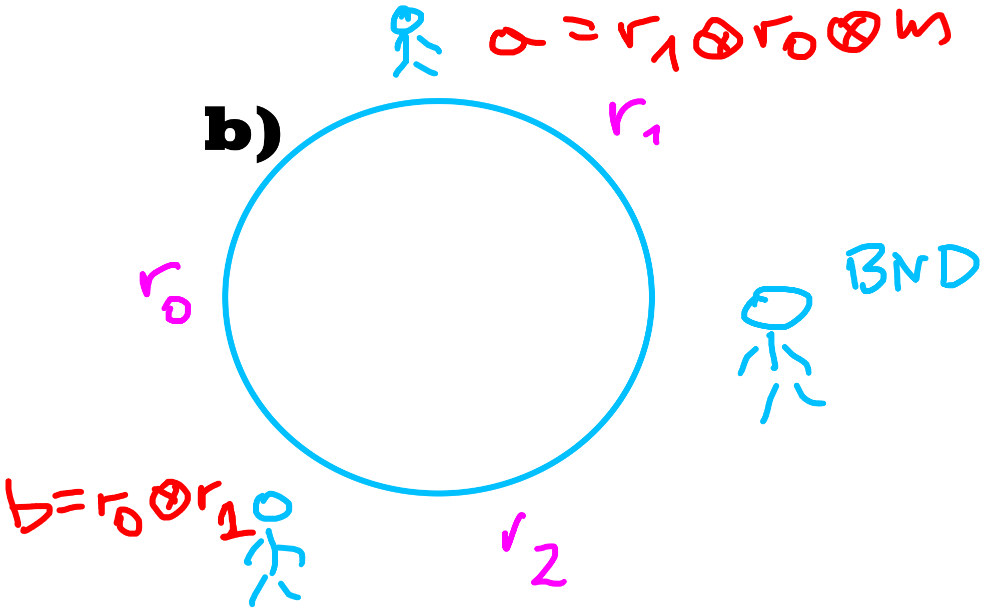
a)



can be solved for r_0

$r_0 := a \otimes r_1$

b)



can be solved for r_0

$r_0 = b \otimes r_2$

Communication steganography–

Hiding communication in innocent traffic

Idea: hiding data in innocent data transmitted (e.g. images, sound, protocol execution data)

steganography versus watermarking:

i. watermarking is visible, not annoying but hard to remove,

ii. steganography is invisible

typical applications: copyright protection, DRM, but also attacks against anonymity of communication

Steganography in images

1. original picture – name: stego image S

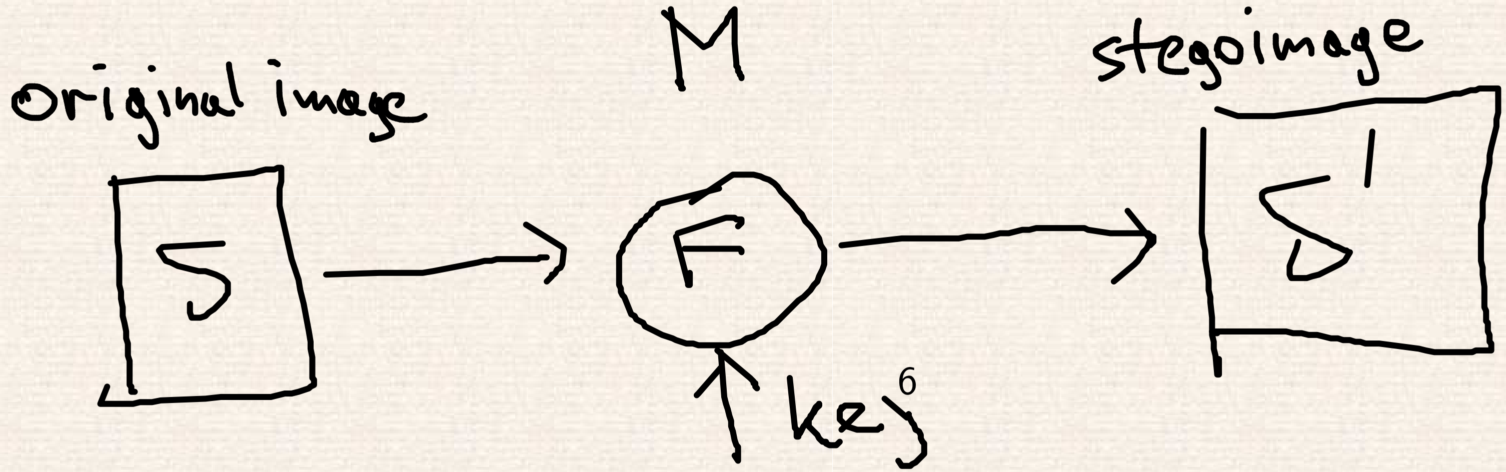
2. marking algorithm: applied to message M and the stego image S

$$S' := F(M, S, \mathrm{key})$$

3. outcome S' transmitted/published

4. retreiving the covered message $M' := G(S', \mathrm{key}, S)$ (S might be optional) where $M' \approx M$

invisibility: without key impossible to decide whether S' (or S) hides a message

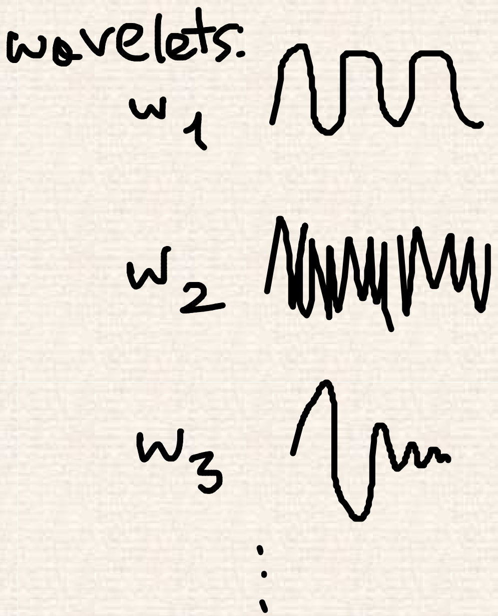


Concrete techniques for image/video/audio steganography:

- changing LSB bits of gray scale

LSB $\longleftrightarrow$ ciphertext

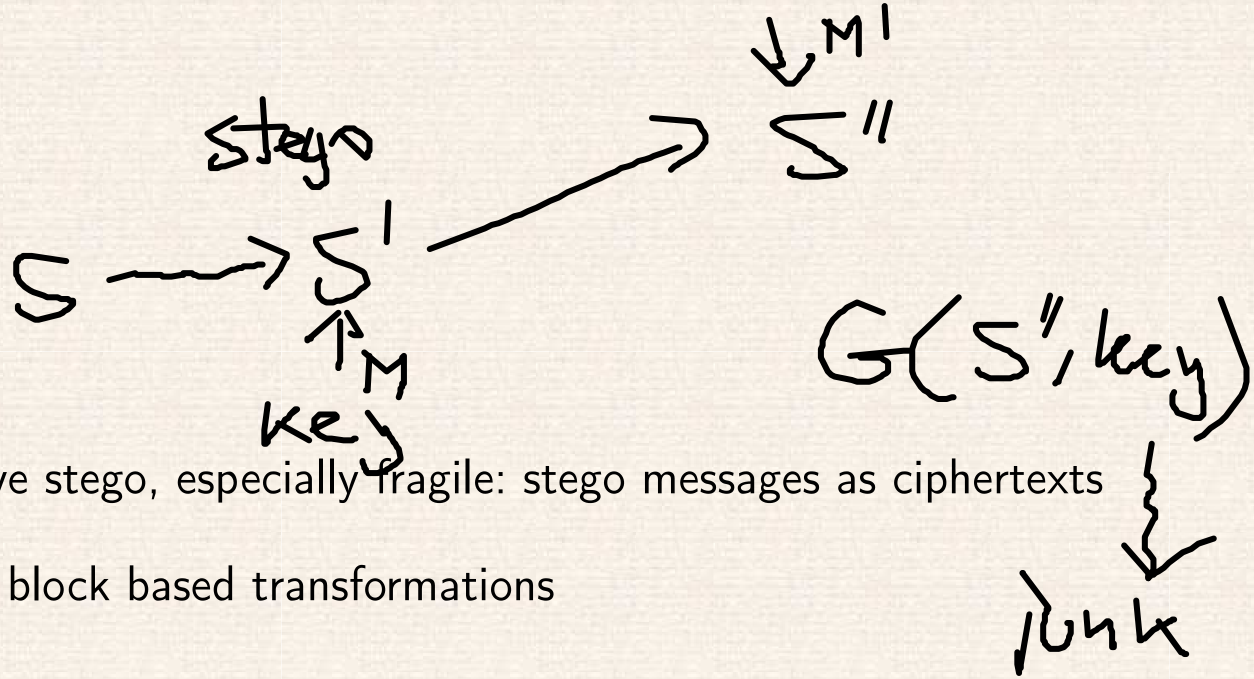
- JPEG encoding: cosinus transform, high frequency components are manipulated anyway for compression
- other digital transforms
- audio encoding: transformation and assigning coefficients to waves – manipulations of certain coefficients undetectable for humans



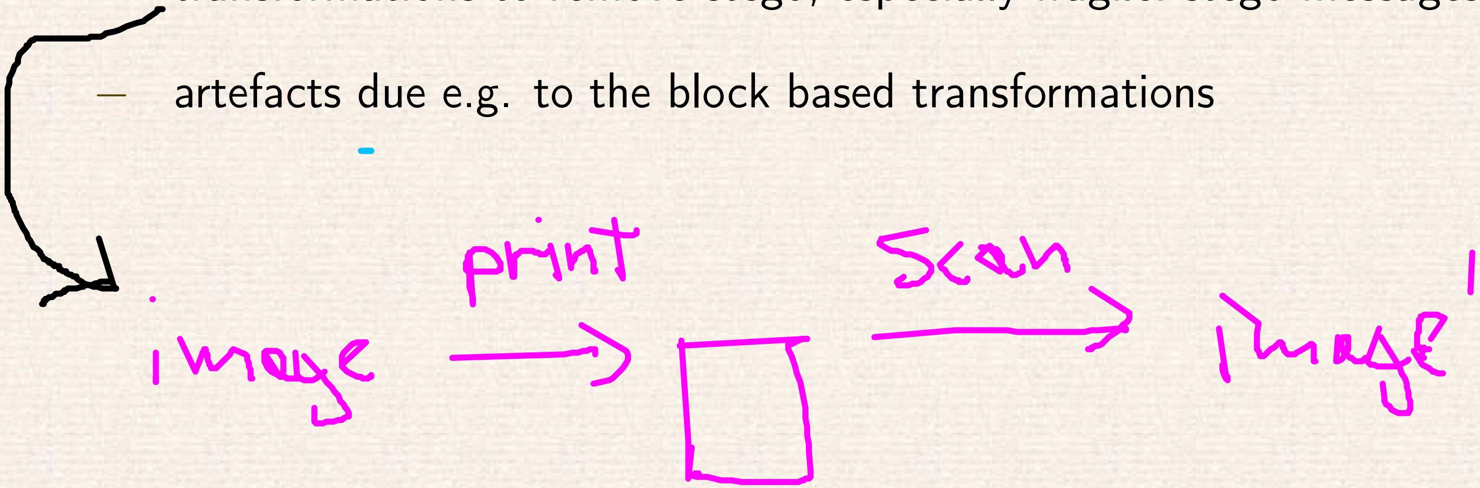
$$\sum \alpha_i \cdot w_i = \text{sound}$$

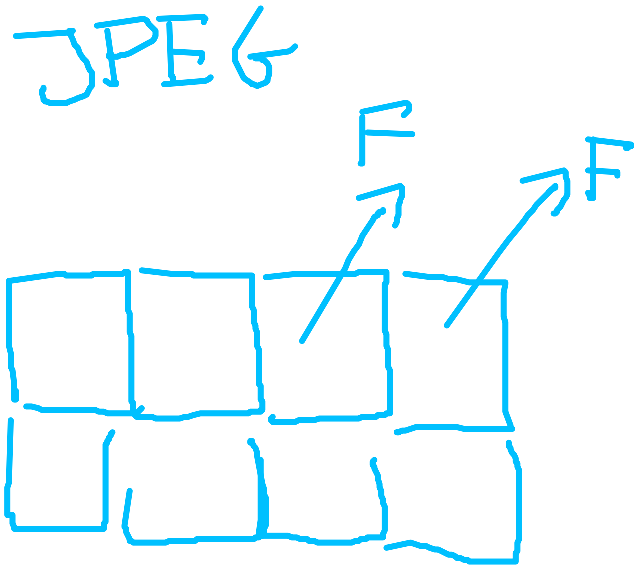
↑ manipulate some coefficients slightly

Problems



- multiple stego images
- transformations to remove stego, especially fragile: stego messages as ciphertexts
- artefacts due e.g. to the block based transformations

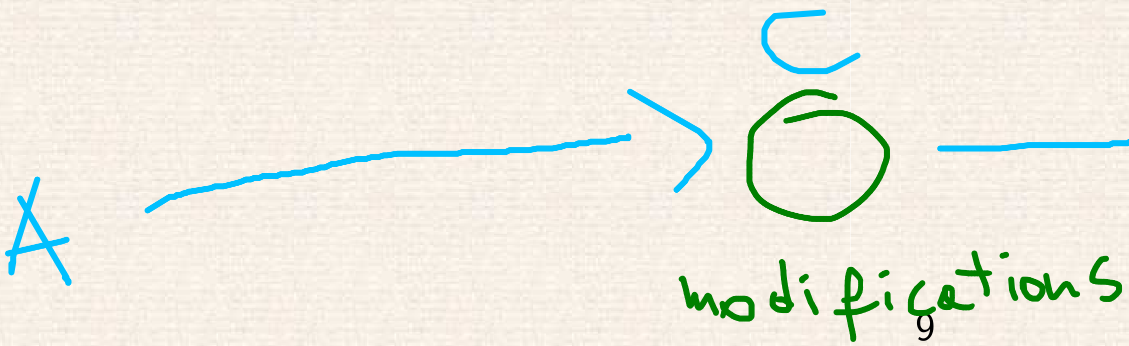


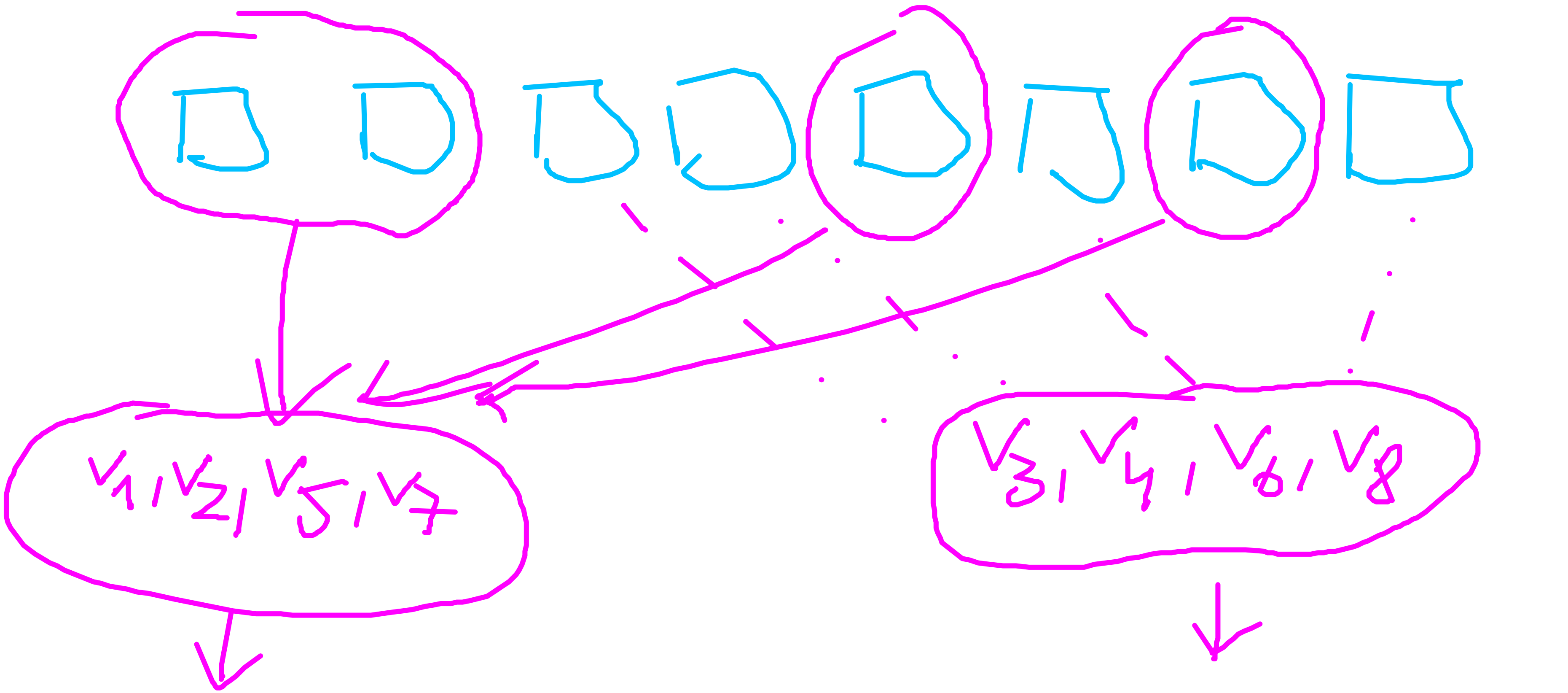


transformation is executed separately in each block, so if not careful, then the blocks of pixels become visible to a human eye

Watermarking/steganography in network flow:

- different ways of encoding (e.g.: a change encodes 1, no change encodes 0)
- all random parameters transmitted in clear may contain watermarks
- simple timing: interpacket delays, departure times may contain watermarks
- mean balancing:
  - $2d$ probes divided into two sets A and B.
  - the expected values of A and B are the same with no watermarking
  - changing the characteristics so that expected values differ in some direction (the direction is the watermarked value)





$v = E(v_i)$ $v = E(v_i)$

$v < \frac{\sum v_i}{4}$ $\frac{\sum v_i}{4} > v$

- sources of mean balancing watermarks:
  - interpacket delays
  - interval centroid: divide into $2d$ time intervals, in each compute mean arrival time
  - interval counting: divide into $2d$ time intervals, in each compute the number of packets

- size: packet size (harder if block encryption applied), object size (https, malicious Javascript generating watermarked size data)
- network rate:
  - one can influence it with dummy packets
  - burst traffic
- response times in transmission, packet order etc

Defense against steganography:

(sometimes problematic or illegal due to intelectual property rights)

i. compression

ii. transforms and random distortions

iii. stretching (Stirmark)

iv. printing and scanning, input to an analog device and digitalize again

effectiveness measured by relative entropy:

$$D(P_1 \| P_2) = \sum_{q \in \mathrm{space}} \mathrm{Pr}_1(q) \cdot \log \frac{\mathrm{Pr}_1(q)}{\mathrm{Pr}_2(q)}$$

relative entropy of plaintexts and hidden text should be smaller than some small ϵ

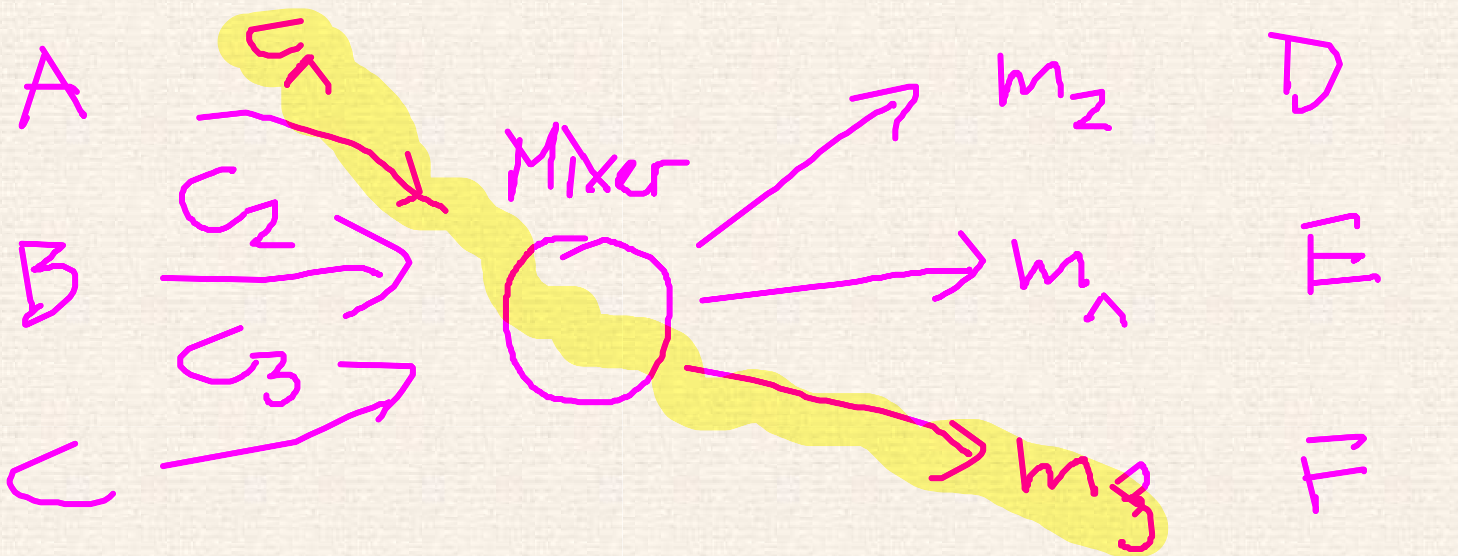
Communication Anonymity via Mixing: A mixer

messages $m_1, m_2, \ldots, m_k$ go through a mixer A:

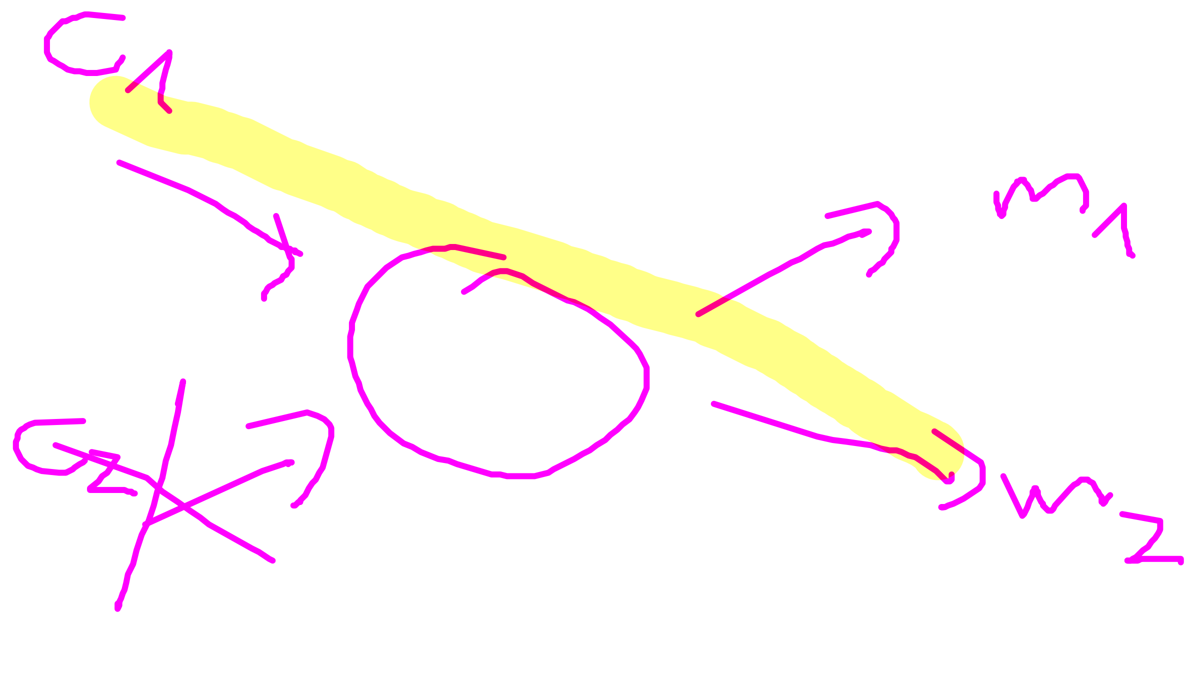
– message m_i sent encrypted with the public key of A

$$C_i := \mathrm{Enc}_{\mathrm{PK}_A}(m_i)$$

– A decrypts $C_1, \ldots, C_k$ and forwards them to their destinations



we assume semantic security of encryption



c_1
c_2
w_1
w_2

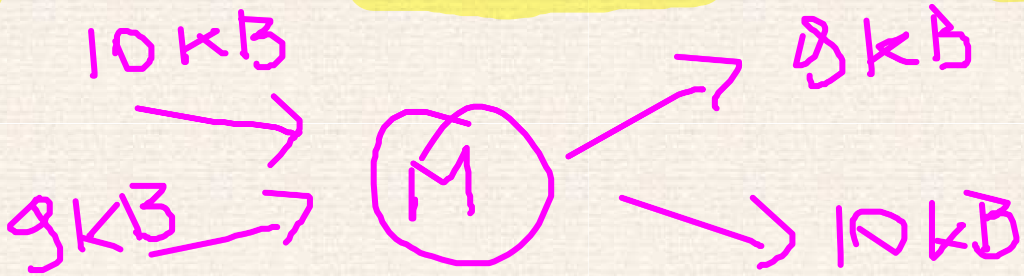
Conditions:

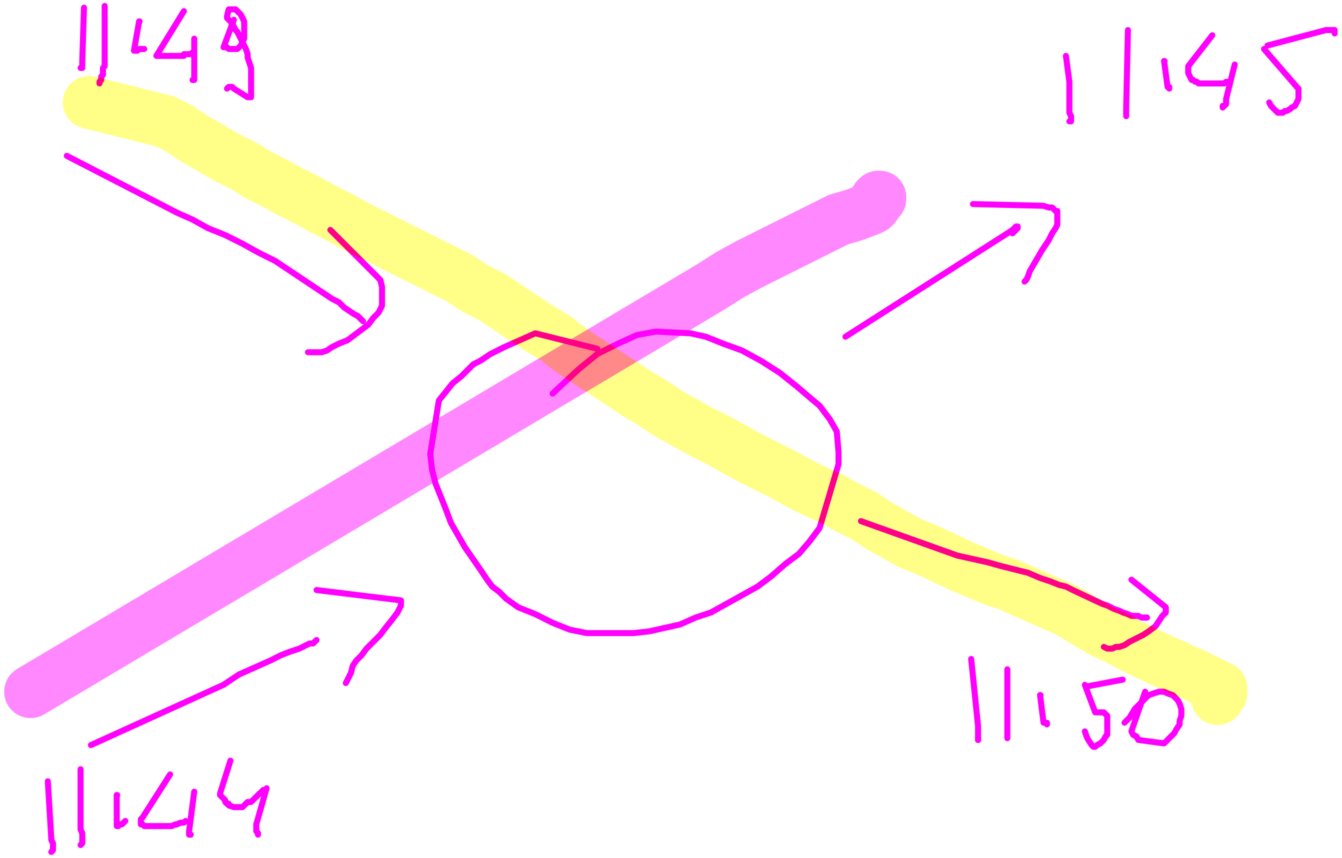
encryption might be secure but nevertheless one can link the ciphertexts with the decrypted texts due to:

- message size
- timing

So:

- all messages should have the uniform size
- A should collect them all, decrypt and send them in a random order





11:49
11:45
11:44
11:50

Onion Routing

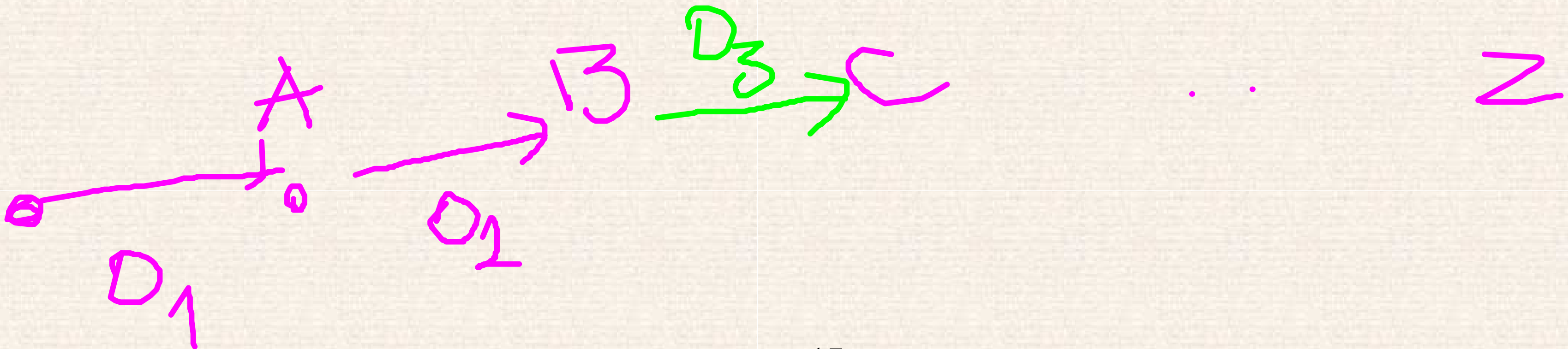
an attempt to hide the sender and the destination of a message – hiding in a crowd of messages

onion creation:

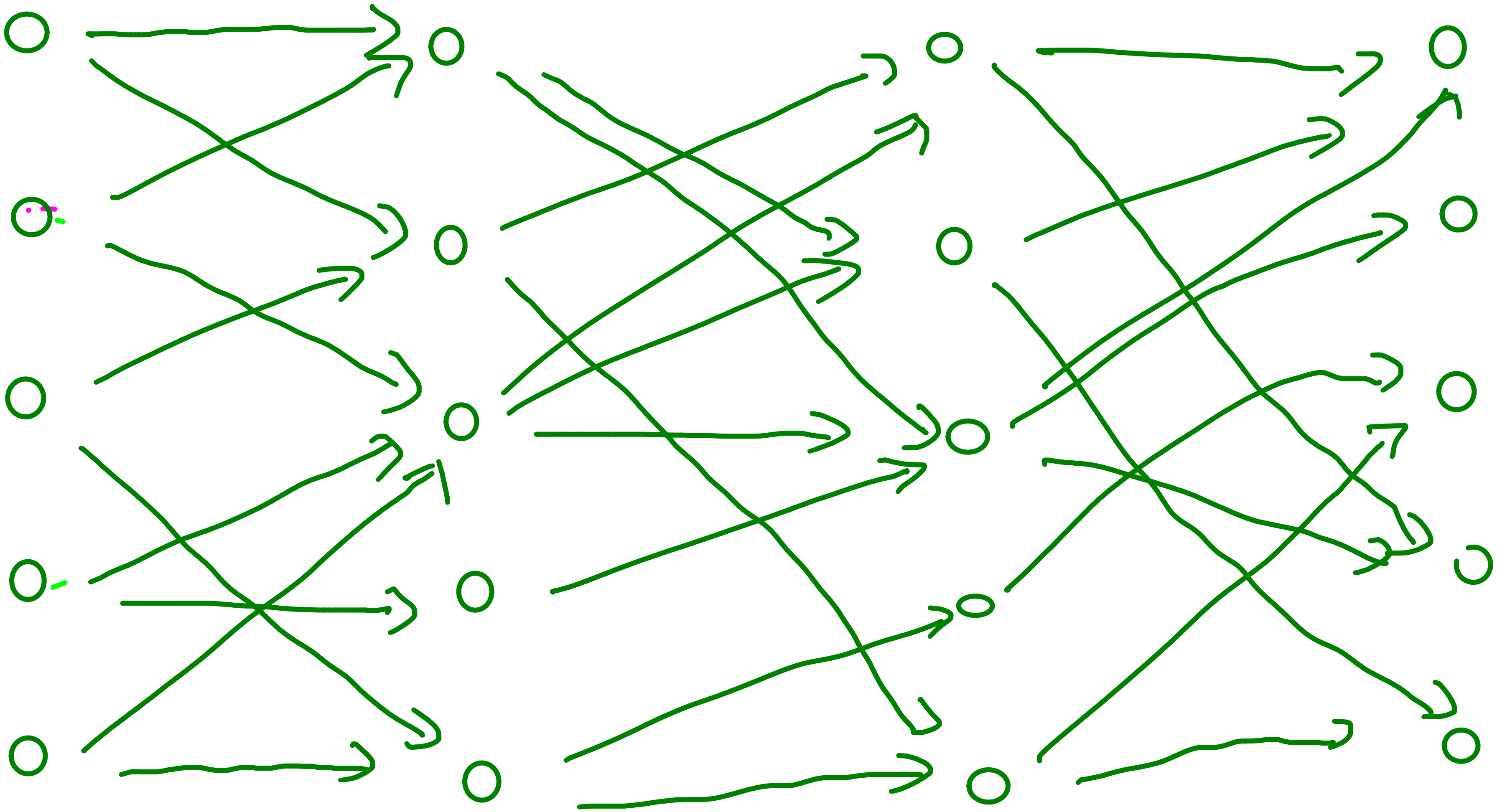
an onion created for a route going through servers $A, B, C, ..., Z$ to the final destination Γ

$$O_1 = \mathrm{Enc}_A(B, \mathrm{Enc}_B(C, \mathrm{Enc}_C(...(\mathrm{Enc}_Z(\Gamma, M))....)))$$

(by encryption Enc_X we mean encryption with the public key of X)



traffic observed



Onion processing:

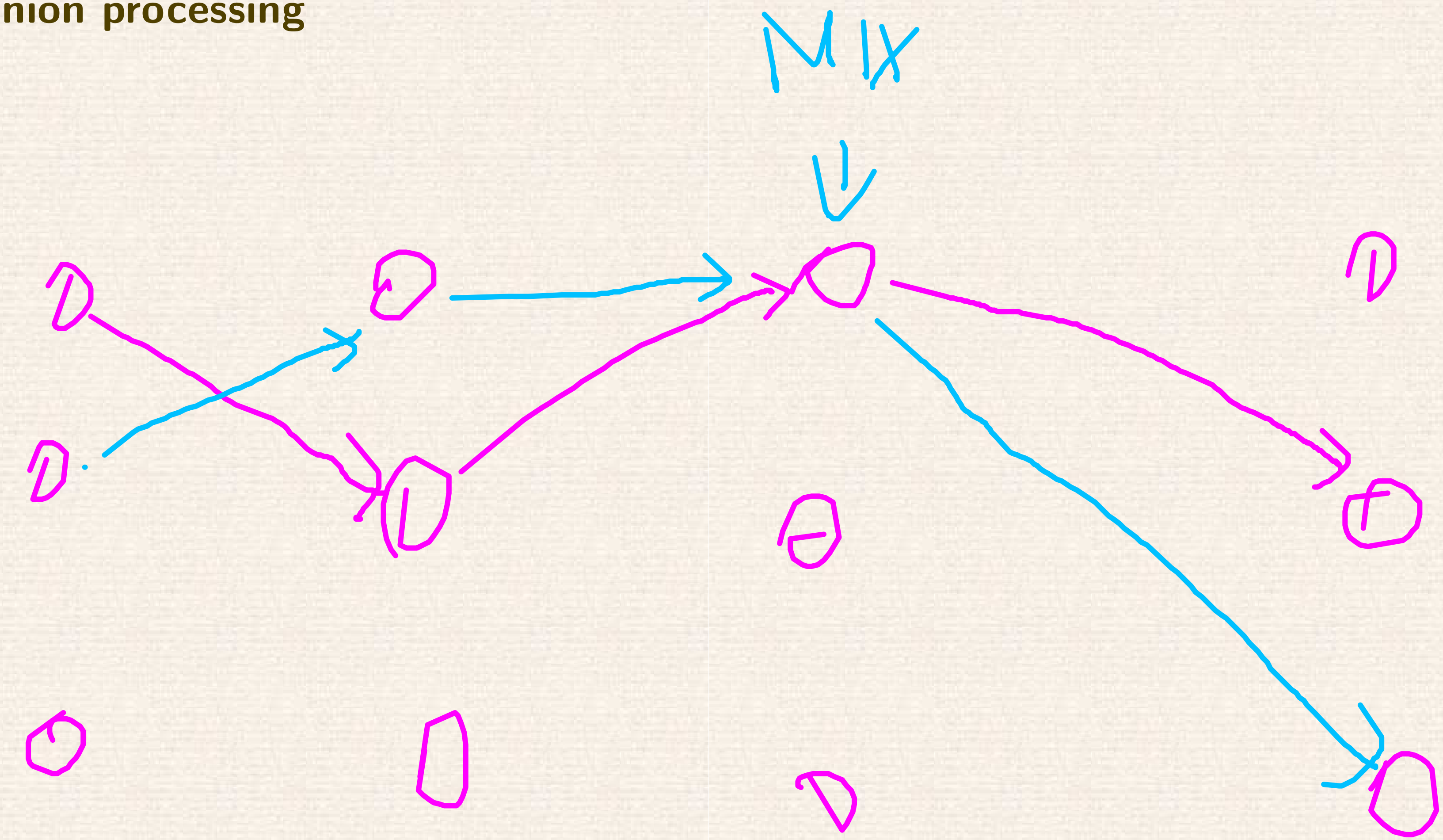
- O_1 sent from the origin machine to A,
- server A decrypts with its private key and gets B and

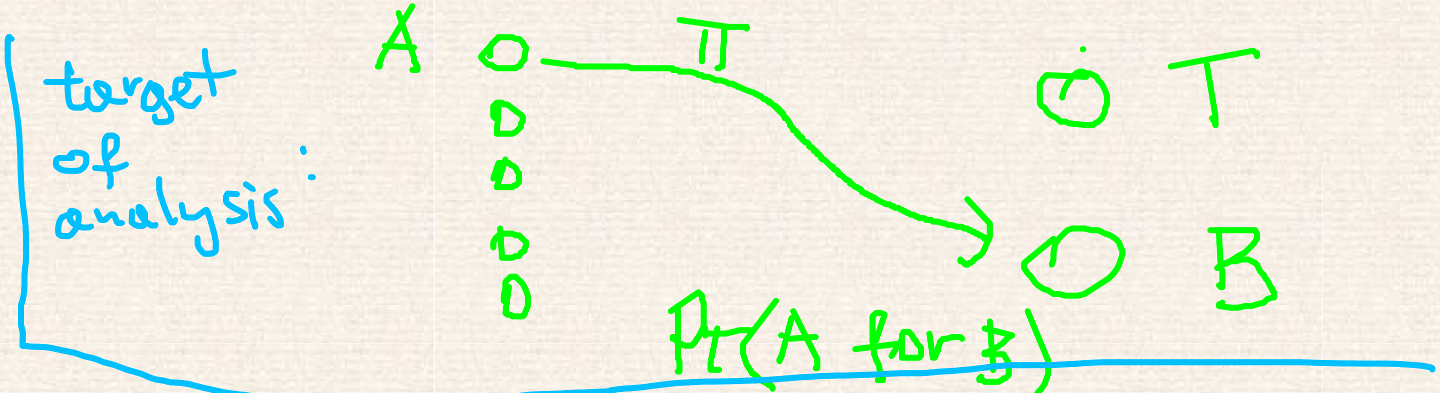
$$O_2 = \mathrm{Enc}_B(C, \mathrm{Enc}_C(\ldots(\mathrm{Enc}_Z(\Gamma, M))\ldots.))$$

- A sends O_2 to B
- server B decrypts O_2 with its private key and gets C and $O_3 = \mathrm{Enc}_C(\ldots(\mathrm{Enc}_Z(\Gamma, M))\ldots.)$
- the process is continued in the same way...
- ... until server Z finds Γ, M and forwards the message M to machine Γ

each processing steps is like peeling off one layer of an onion

Onion processing





Limitations

- **idea**: if two or more onions enter a server at the same time, get partially decrypted, then forwarded, then it is impossible to say which incomming onion corresponds to which outcoming onion - **node mixing**

- **traffic analysis:** assigning probabilities to permutations ($\pi(i) = j$ means that the ith sender has a message to the jth receiver)

- **it is not enough to say that $\pi(i) = j$ with ppb $\approx \frac{1}{n}$:**
  - let us assume that the adversary knows that π is a circular shift
  - assume that the adversary gets extra knowledge:
    - "if source i is talking to destination j, then $i+1$ is talking to destination $j+1$"

  however, still $\Pr(\pi(i) = j) = \frac{1}{n}$

Question: necessary length of the onions?

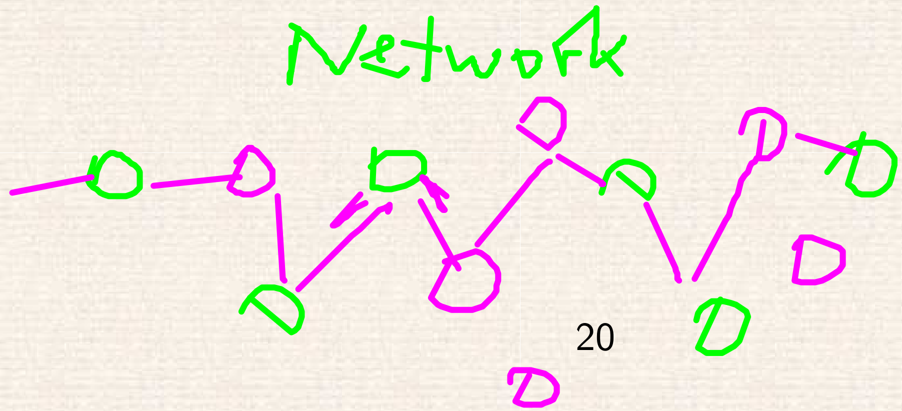
analytical results for restricted case of n senders and n receivers, messages sent simultaneously:

– $O(\log^2 n)$ if the adversary has a full knowledge of the system (not likely to have a better estimation unless ... big progres in math), **assumption:** uniform distribution for choosing destinations

$\log_2 2^{20} = 400$!

– $O(\log n)$ if the adversary can see only a constant fraction of nodes, **assumption**: sender i may have non-uniform distribution of destination points

– it is easy to see that $\Omega(\log n)$ is necessary



maximum result from cryptanalysis:

Π - random variable describing solution

result: probability distribution:

$\Pi = \Pi_1$ with probability p_1

$\Pi = \Pi_2$ with probability p_2

$\vdots$

perfect protection if

$\pi = \pi_i$ with probability $\approx \frac{1}{n!}$

for each π_i

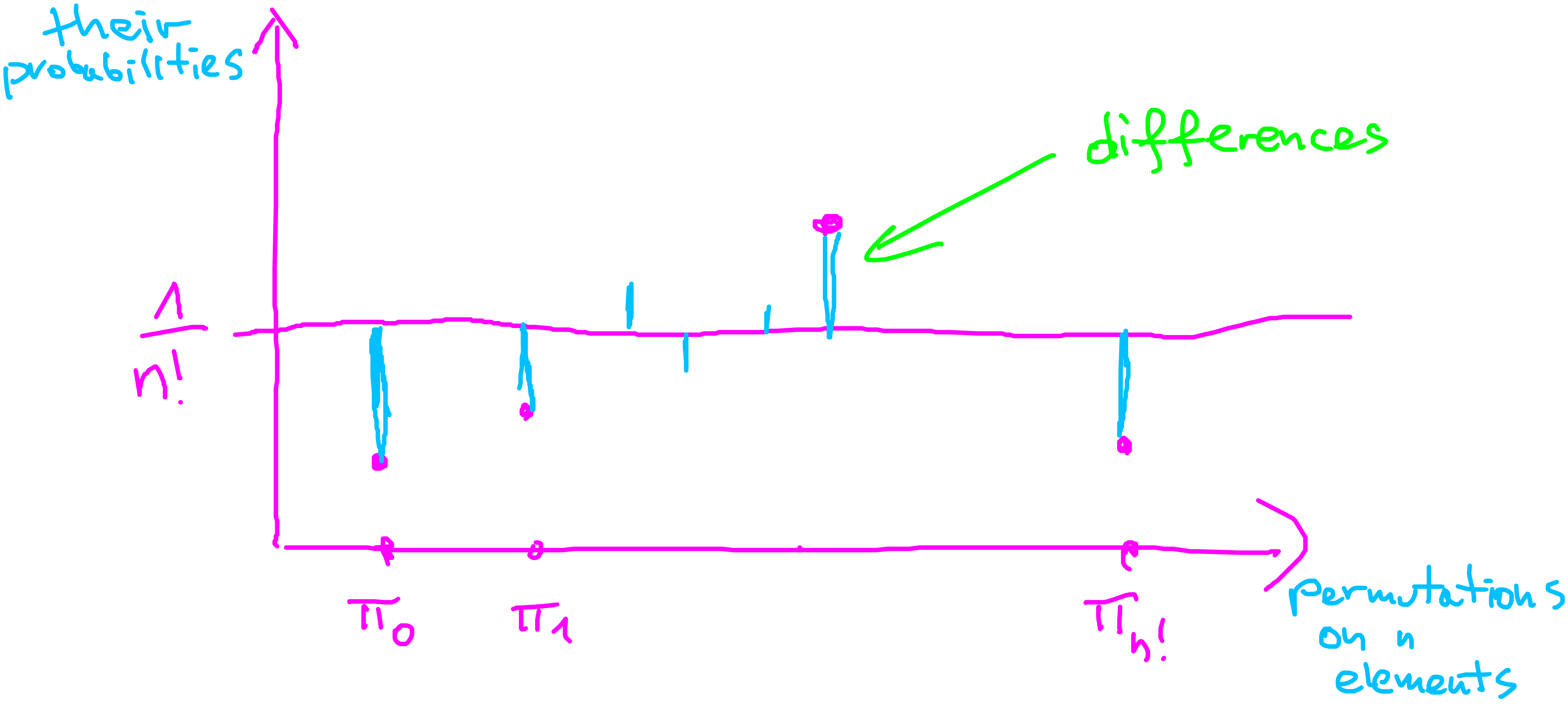
remark: getting $= \frac{1}{n!}$ might be impossible.

an example: p - prime, $p < n$, while no mix with

p inputs and outputs

no way to get $\frac{1}{p}$ for $\frac{1}{n!}$

measure of difference of probability distributions



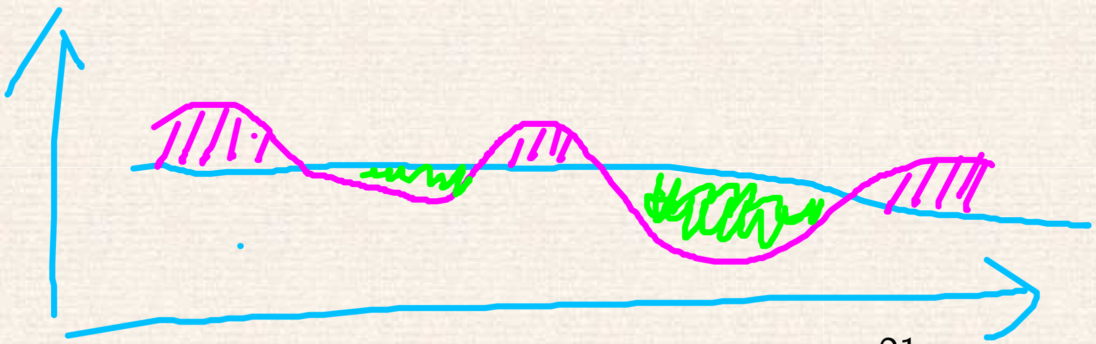
total variation distance to uniform distribution

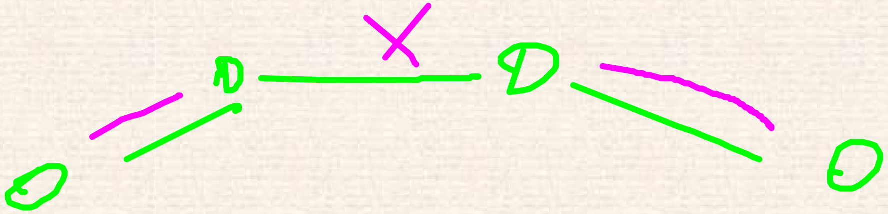
Meaning of the results:

traffic analysis does not improve our prior knowledge in a significant way (e.g. if we know in advance that source i always sends to destination j, then onions cannot hide this fact)

the guarantees are given in terms of **total variation distance** of two probability distributions:

$$\|\pi, \mu\| = \frac{1}{2}\sum_{\omega\in\Omega} |\pi(\omega) - \mu(\omega)|, \qquad \text{where } \Omega \text{ is the set of all events}$$





Problems with onions

– **replay attacks**: just send the same onion (or partially decrypted onion) for the 2nd time: the same subonions will appear along the forwarding path

defense: universal reencryption, example based on ElGamal encryption:

– ciphertext of m:

$(\beta^r, m \cdot g^r, \beta^s, g^s)$ for r, s chosen at random

– renecryption:

1. choose α, β at random
2. replace (y_1, y_2, y_3, y_4) by $\left(y_1 \cdot {y_3}^{\alpha}, y_2 \cdot y_4^{\alpha}, y_3^{\beta}, y_4^{\beta}\right)$

   thereby we get $(\beta^{r+\alpha s}, m \cdot g^{r+\alpha s}, \beta^{s\beta}, g^{s\beta})$

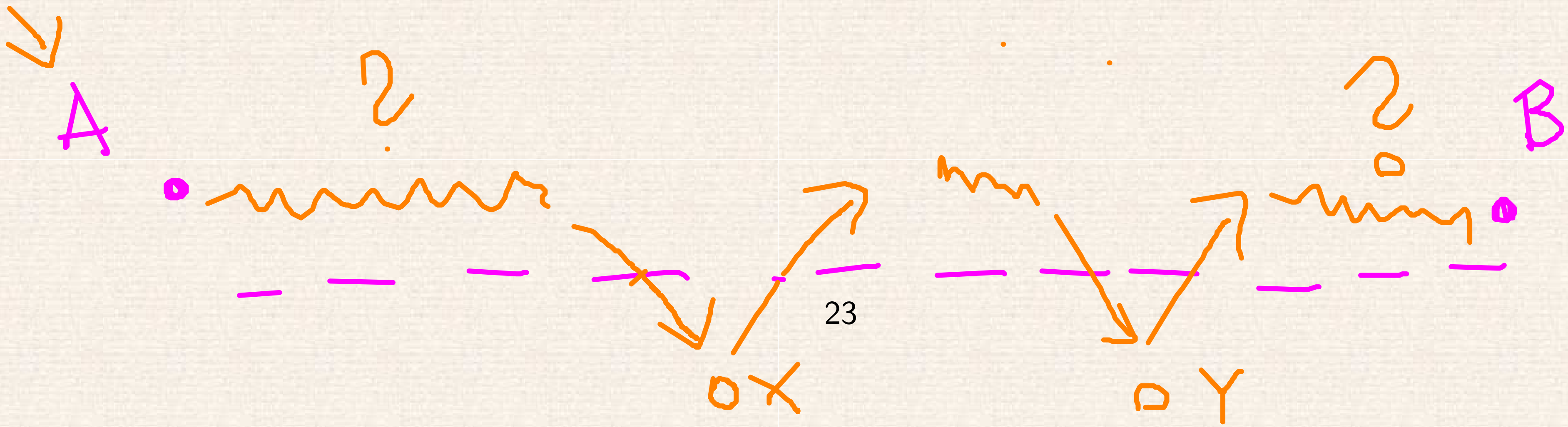
   if order of the group is a prime number, then this is equivalent with choosing $(\beta^{r'}, m \cdot g^{r'}, \beta^{s'}, g^{s'})$ for random r', s'

Local view:

not all users have the same list of servers

then: **long routes do not improve anonimity**. Toy example:

- give user A a list of servers with 50% of servers used by nobody else
- no matter how long is the routing path designed by A, it is likely that close to destination the path goes through a rogue server
- a few destinations available from this rogue server (50% of cases the rogue server sends directly to the destination)
- an onion going through the rogue server originates from the attacked source



Network information

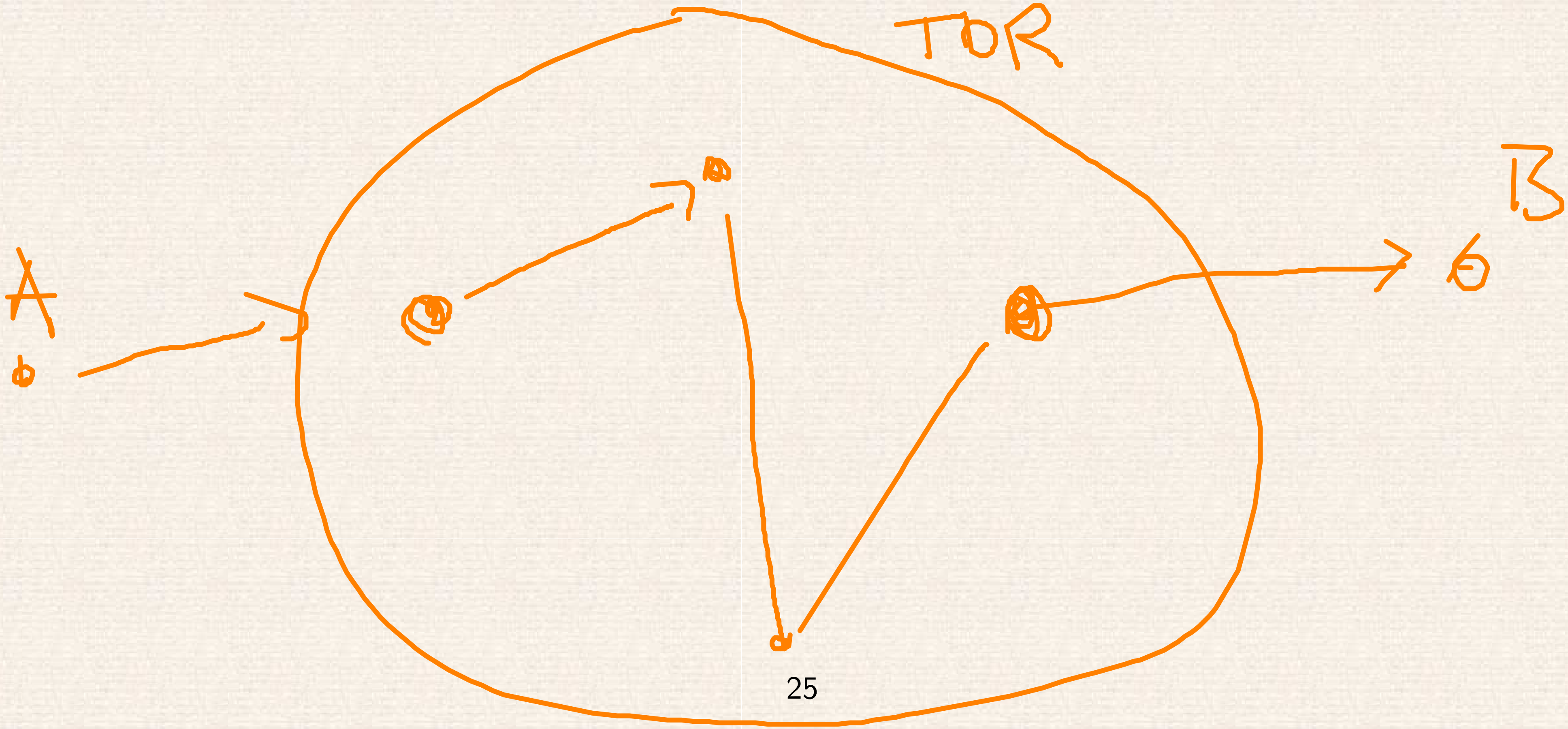
- **timing at nodes**: delays necessary

  defense: collecting enough onions and flashing them at once. (slowdown!!!)

- **sparse traffic** means no protection

TOR

- free BSD licence
- connection based protocol, new connection established periodically ("10 minutes or so")
- routes limited to 3 TOR nodes



TOR: onion based forwarding the symmetric keys

i. each node on the path learns only the predecessor and the successor

ii. the path established step by step:

- after establishing a subpath $X_0, X_1, ..., X_k$ the subpath is used to send an encrypted message over the channel to X_k stating that the next node is X_{k+1}.
- the sender and X_{k+1} negotiate a new connection key via DH key exchange

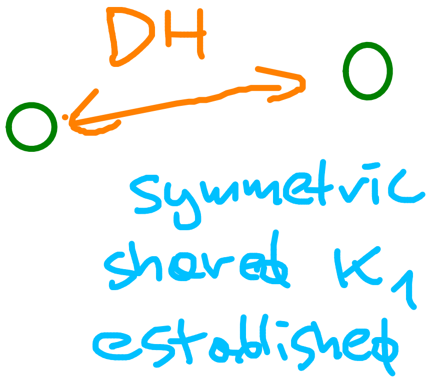
iii. after making a connection the message is encrypted symmetrically with the keys:

$$\text{AES}_{\text{relay1}}(\text{AES}_{\text{relay2}}(\text{AES}_{\text{relay3}}(m)))$$

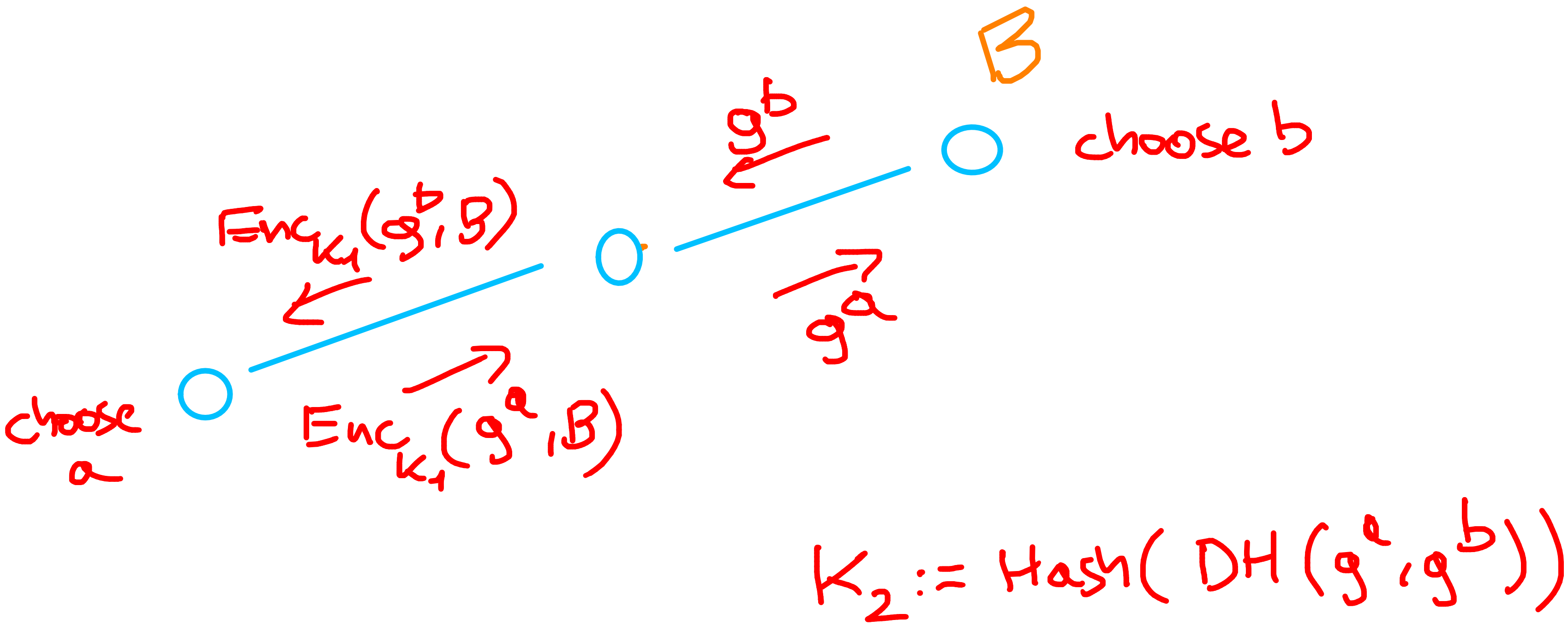
each relay node removes one layerof encryption when forwarding a message

iv. **response to the sender:** instead of decryption: encryption with keys shared with the sender. The sender has to decrypt the onion

Step 1: connecting to the 1st node on the path

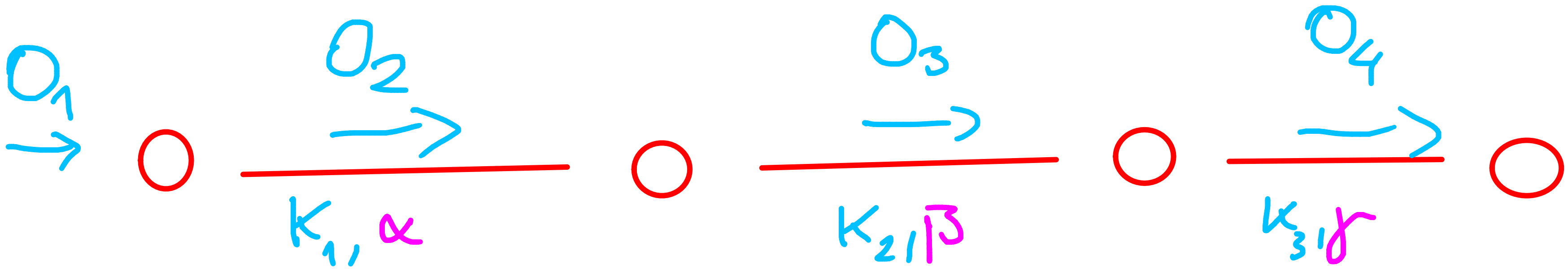


Step 2: connecting to the 2nd node on the path



DH protocol via the 1st node
(only forwarding the messages)

Sending onion and peeling it off



$$O_1 = (\alpha, Enc_{K_1}\underbrace{(\beta, Enc_{K_2}\underbrace{(\gamma, \underbrace{Enc_{K_3}(message)}_{O_4})}_{O_3})}_{O_2})$$

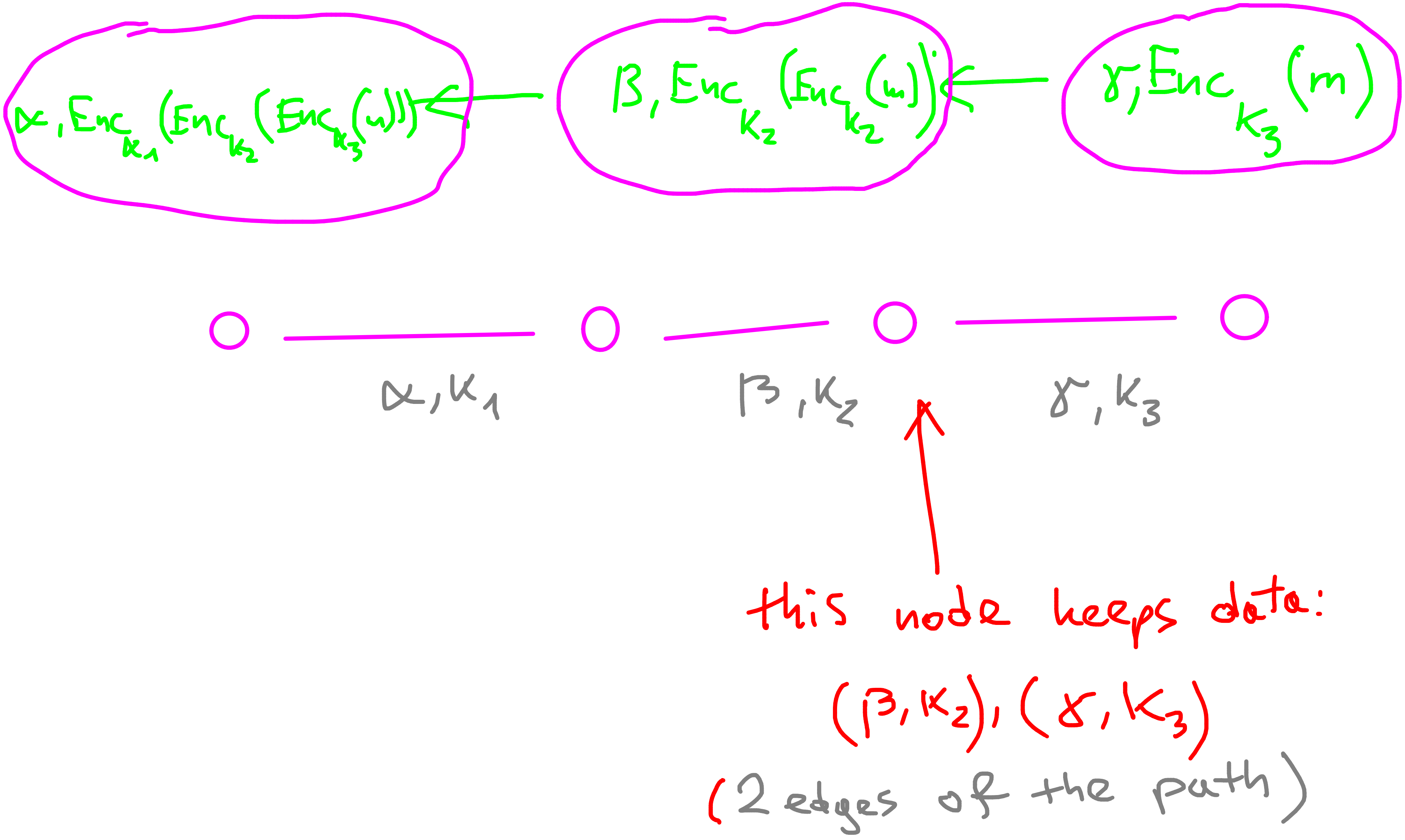
Problems:

- the exit node knows the plaintext
- traffic correlation
- application level attacks
- Heartbleed - change of public keys, some clients use old keys, ...

Other issues:

- many authorities fight against TOR as it helps to escape the control

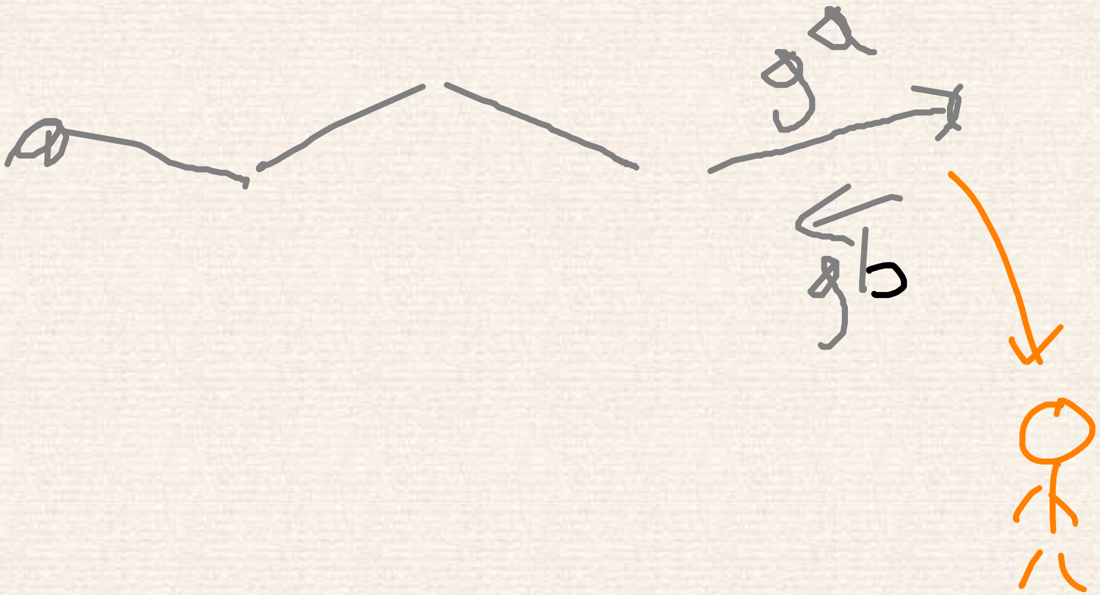
Sending onion back



Onion Routing - Warning: Rogue Encryption

example: weak DH so that the relay key is leaked

the sender has a weak PRNG, so each g^a proposed in DH can be broken, adversary learns the symmetric keys $K_1, K_2, \ldots$



$a = F(g^a)$

solving special case of DL

$K_4 := (g')^a$

recommendations of ENISA

PSEUDONYMIZATION

Symmetric methods:

SALT

– **hashing the identifier:** $\text{pseudonym} = \text{Hash}(\text{identifier})$

problem: it is impossible to compute the identifier from the pseudonym, however hashing all possible identifiers and brute force reveals the link between the pseudonym and identifier

– **encryption with a (secret) symmetric key:** unlinkability, however the user cannot compute the pseudonym himself and the owner of the secret key can link all pseudonyms

– **hashing with a key:** as above, the party holding the secret key has to perform brute force to link back the pseudonym to the identifier

F: Enc_K(Jan Kowalski) = Ax76--

F^{-1}: DEC_K(Ax76...) = Jan Kowalski

Guessing real ID:

$$p := \text{Hash}(K, \text{Jan Nowak})$$

find x such that:

$$p = \text{Hash}(*, x)$$

↑ anything

pseudonymous ID

$$ID \longrightarrow F(ID)$$

there is a trapdoor function

$H = F^{-1}$

anonymous ID

$$ID \longrightarrow G(ID)$$

no trapdoor function to compute G^{-1}

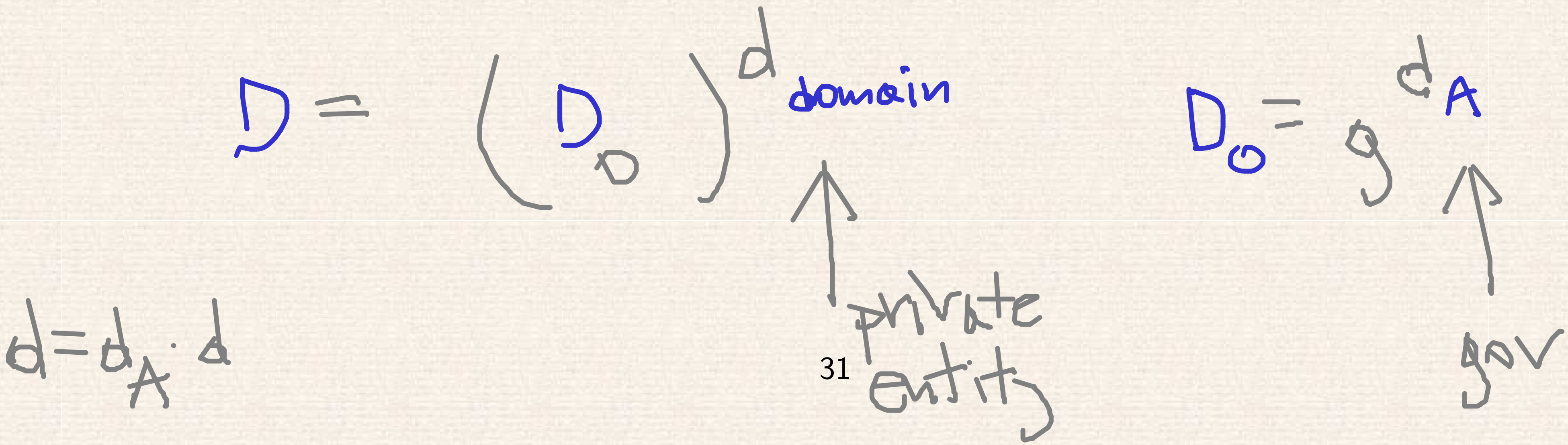
example: $G(ID) := HMAC(K, ID)$

Asymmetric pseudonymization methods:

- **based on Diffie Hellman Problem**:
  - a **domain** (service provider, database, etc) holds a pair of keys $(d, D = g^d)$
  - a user Alice holds a pair $(x, X = g^x)$
  - the **pseudonym** of Alice corresponding to D is $g^{x \cdot d}$, which is computed as X^d by the domain manager, and as D^x by Alice
  - nobody but the user and the domain manager can compute the pseudonym:

    for a 3rd person deciding whether X^d corresponds to X in domain D means solving DDH Problem

- **a variant based on domain and a central Authority:**
  - the key d is not known to the domain authority
  - $d = d_A + d_{\text{domain}}$, where d_A is known by Authority and d_{domain} is known by the domain manager
  - steps of generating the pseudonym:
    1. Authority computes $X' := X^{d_A}$ and presents X' to the domain manager
    2. the domain manager computes pseudonym as $(X')^{d_{\text{domain}}}$
  - linking a pseudonym with the starting public key is a reverse process but **both the domain manager and the Authority must participate** in it



deanonymization

pseudonym $p := D^x = g^{dx}$

$$(p)^{1/d_{domain}} = p'$$

$$(p')^{1/d_A}$$

$$= (g^{dx})^{\frac{1}{d_{domain}} \cdot \frac{1}{d_A}} = g^{dx/d} = g^x$$

g^x is the user's main public key (non-anonymous)

- **a variant from German personal identity cards (Restricted Identification):**
  - **pseudonym** of a user with public key $X = g^x$ is $\mathrm{Hash}(D^x)$
  - **pseudonym presentation:** by the ID card over a secure channel,
    - no proof that the pseudonym is correct
    - but a smart card can create only one pseudonym per domain
  - **revocation:** by computing $\mathrm{Hash}((X^{d_A})^{d_{\mathrm{domain}}})$ jointly by the Authority and the domain manager and putting the result on the **blacklist**
  - **blacklisting based on the domain pseudonym:** requires brute force and recomputing all pseudonyms
- more flexibility, if pairing groups are available but be careful: DDH might be easy and so the above methods do not work

$\mathrm{Hash}(D^x)$ $\mathrm{Hash}(D'^x)$

D, D'. D^x D'^x

Id card stolen, public key X

1) ID issuer knows X

2) computing pseudonyms:

$$X \longrightarrow X^{d_A}$$

3) X^{d_A} sent to a domain A

4) A calculates

$$z := \mathrm{Hash}\left(\left(X^{d_A}\right)^{d_{domainA}}\right)$$

and puts z on the blacklist

Serious problems

database in hospital A with pseudonyms

database in hospital B with pseudonyms

data from A to be included in B

the following computation is infeasible:

$$\mathrm{Hash}(D^{x}) \longrightarrow \mathrm{Hash}(D'^{x})$$

Advantages and disadvantages of Restricted Identification:

- different pseudonyms generated automatically is
  - user friendly
  - makes re-identification based solely on data related to the pseudonym much harder
- problems:
  - converting a pseudonym in domain D_1 to a pseudonym in domain D_2 might be hard or infeasible, and require cooperation with the user and/or an authority

    (problem area: moving pseudonymized medical records)

DATABASES and PRIVACY for QUERIES

the main problem is answering queries: does a query result disclose personal data?

Approach 1: anonymity set

– a query accepted if the number of record used to answer the query is at least k (and each concerns a different person)

– the method is naive: the attack is to ask for two sets of records: one including Alice and one excluding Alice to know the value for Alice

Approach 2: differential privacy

classify the algorithms (queries)

algorithm A satisfies ϵ-differential privacy, if for any two databases D and D' that differ by elimination of one record:

– for any subset S of the image of A:

$$\Pr\left(A(D) \in S\right) \leq e^{\epsilon} \cdot \Pr\left(A(D') \in S\right)$$

where the probability is over the random choices within the algorithm A

Then:

– $\epsilon = 0$ is the ideal for privacy: as $e^0 = 1$ and the probabilities are exactly the same, but the result does not depend on the database contents (noise)

– so it is necessary to find balance between privacy (ϵ as small as possible) and information in the response (ϵ as big as possible)

Typical approach for achieving differential privacy

if the output of a query if z, then A creates a noise δ and outputs $z + \delta$

goal: if noise stronger than the effect of eliminating one record, then it should work to some extent

Problem with outliers

if there are records that have very different values it is hard to keep promise of *differential privacy*

solution: disregard them (as private data leak anyway) and concentrate on the rest

e.g.:

1. disregard a few entries that are outliers

2. for differential privacy take only those elements that have at least k neighbors in some sense

PSEUDONYMOUS SIGNATURES

Application areas:

- while having the pseudonyms, how to authenticate digital data? Digital signatures would solve the problem
- implementing GDPR rights in practice:

  a data subject can authenticate the request (e.g. for data rectification) in a database with pseudonyms by sending a request with a signature corresponding to the pseudonym

BSI Pseudonymous Signature:

- **keys:**
  - domain parameters D_M and a pair of global keys $(\mathrm{PK}_M, \mathrm{SK}_M)$
  - public key $\mathrm{PK}_{\mathrm{ICC}}$ for a group of eIDAS tokens, the private key $\mathrm{SK}_{\mathrm{ICC}}$ known to the

issuer of eIDAS tokens

– assigning the private keys for a user:

the issuer chooses $\mathrm{SK}_{\mathrm{ICC},2}$ at random, then computes $\mathrm{SK}_{\mathrm{ICC},1}$ such that

$$\mathrm{SK}_{\mathrm{ICC}} = \mathrm{SK}_{\mathrm{ICC},1} + \mathrm{SK}_{M} \cdot \mathrm{SK}_{\mathrm{ICC},2}$$

– a sector (domain) holds private key $\mathrm{SK}_{\mathrm{sector}}$ and public key $\mathrm{PK}_{\mathrm{sector}}$.

– a sector has revocation private key $\mathrm{SK}_{\mathrm{revocation}}$ and public key $\mathrm{PK}_{\mathrm{revocation}}$

– sector specific identifiers $I^{\mathrm{sector}}_{\mathrm{ICC},1}$ and $I^{\mathrm{sector}}_{\mathrm{ICC},2}$ for the user:

$$I^{\mathrm{sector}}_{\mathrm{ICC},1} = (\mathrm{PK}_{\mathrm{sector}})^{\mathrm{SK}_{\mathrm{ICC},1}}$$

$$I^{\mathrm{sector}}_{\mathrm{ICC},2} = (\mathrm{PK}_{\mathrm{sector}})^{\mathrm{SK}_{\mathrm{ICC},2}}$$

- **signing:** with keys $\mathrm{SK}_{\mathrm{ICC},1}$, $\mathrm{SK}_{\mathrm{ICC},2}$ and $I^{\mathrm{sector}}_{\mathrm{ICC},1}$ and $I^{\mathrm{sector}}_{\mathrm{ICC},2}$ for $\mathrm{PK}_{\mathrm{sector}}$ and message m

   i. choose K_1, K_2 at random

   ii. compute

   – $Q_1 = g^{K_1} \cdot (\mathrm{PK}_M)^{K_2}$

   – $A_1 = (\mathrm{PK}_{\mathrm{sector}})^{K_1}$

   – $A_2 = (\mathrm{PK}_{\mathrm{sector}})^{K_2}$

   iii. $c = \mathrm{Hash}(Q_1, I^{\mathrm{sector}}_{\mathrm{ICC},1}, A_1, I^{\mathrm{sector}}_{\mathrm{ICC},2}, A_2, \mathrm{PK}_{\mathrm{sector}}, m)$

   (variant parameters omitted here)

   iv. compute

   – $s_1 = K_1 - c \cdot \mathrm{SK}_{\mathrm{ICC},1}$

   – $s_1 = K_2 - c \cdot \mathrm{SK}_{\mathrm{ICC},2}$

   v. output (c, s_1, s_2)

- **verification:**

  compute

  – $Q_1 = (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2}$

  – $A_1 = (I^{\mathrm{sector}}_{\mathrm{ICC},1})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_1}$

  – $A_2 = (I^{\mathrm{sector}}_{\mathrm{ICC},2})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_2}$

  – recompute c and check against the c from the signature

- why it works?

$$(\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2} = (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot \mathrm{SK}_{\mathrm{ICC},1}} \cdot (\mathrm{PK}_M)^{c \cdot \mathrm{SK}_{\mathrm{ICC},2}}$$

$$= (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot \mathrm{SK}_{\mathrm{ICC},1}} \cdot (g)^{-c \cdot \mathrm{SK}_M \cdot \mathrm{SK}_{\mathrm{ICC},2}}$$

$$= (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot \mathrm{SK}_{\mathrm{ICC}}} = g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} = Q_1$$

- there is a version without A_1, A_2 and the pseudonyms $I^{\mathrm{sector}}_{\mathrm{ICC},1}, I^{\mathrm{sector}}_{\mathrm{ICC},2}$

Problems:

- **the issuing authority knows the private keys**

  but: there is a way to solve it when the user gets two pairs of keys on the device and takes their linear combination)

- **breaking into just 2 devices reveals the system keys**

- possible to create a trapdoor for enabling to link pseudonyms

  - apart from $\mathrm{SK}_{\mathrm{ICC}} = \mathrm{SK}_{\mathrm{ICC},1} + \mathrm{SK}_M \cdot \mathrm{SK}_{\mathrm{ICC},2}$ there is a another relationship for the user u

$$x_u = \mathrm{SK}_{\mathrm{ICC},1} + s_u \cdot \mathrm{SK}_{\mathrm{ICC},2}$$

  - x_u and s_u are dedicated for user u - maybe not in the database but derived from a secret key, say Z
  - domain trapdoor: $T_{\mathrm{domain},u} = \mathrm{PK}_{\mathrm{domain}}^{x_u}$ and s_u (it can be derived from Z alone)
  - then one can conclude that nym_1 and nym_2 correspond to user u, iff:

$$T_{\mathrm{domain},u} = \mathrm{nym}_1 \cdot \mathrm{nym}_2^{s_u}$$

Anonymous credentials

two commercial products (libraries): Idemix (IBM) and UProve (Microsoft)

some details concerning Idemix

components:

– **actors:** issuer, recipient, verifier, trusted party

– **attributes**: for each attribute there is: `name, value and type`. The types are `int, string, date, enum` (enumeration). The attributes concern the recipient.

– **credentials**: given by the issuer to the recipient

i. known (A_k): the issuer knows the value of an attribute

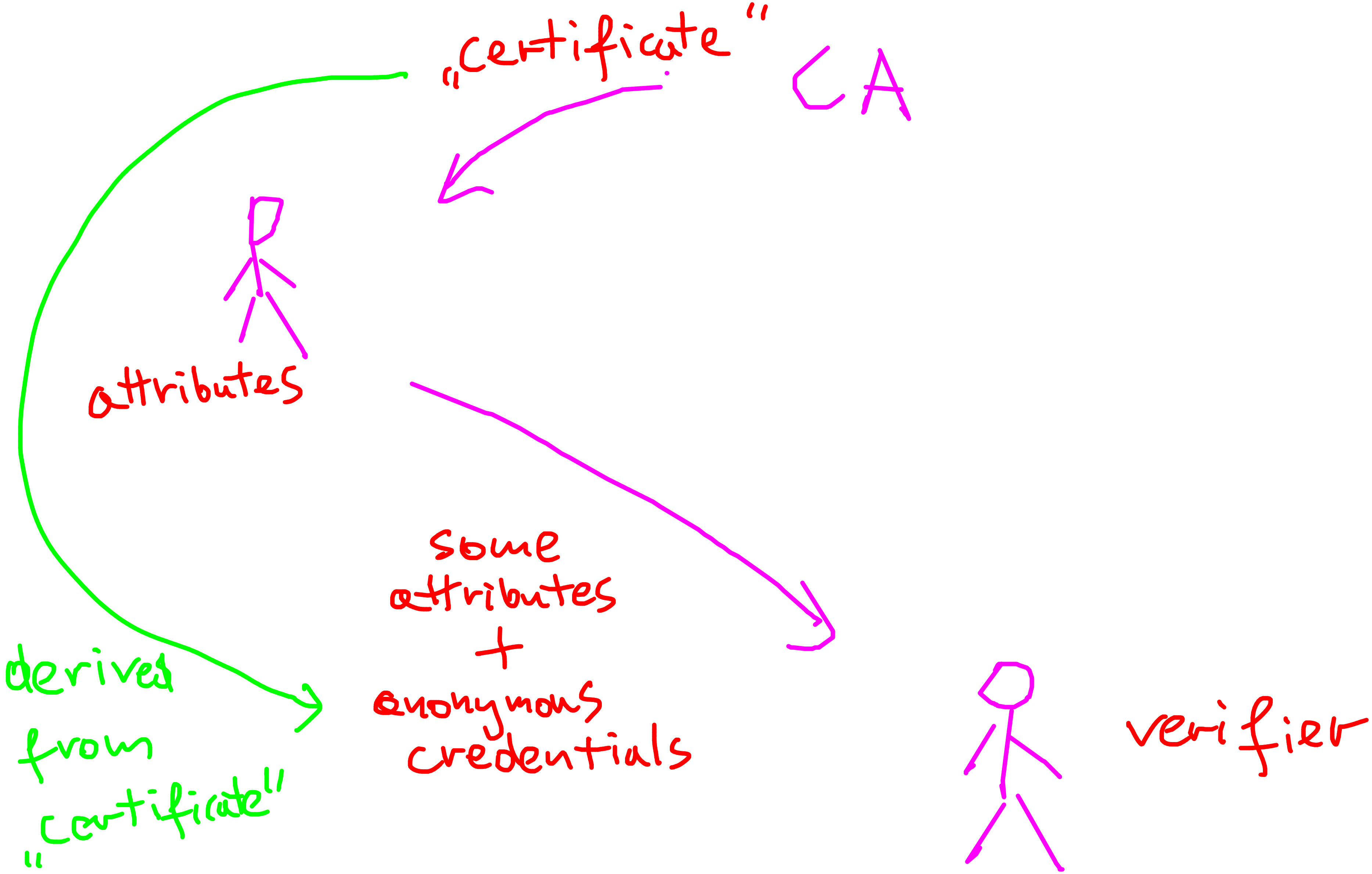
ii. commited (A_c): the issuer knows a commitment to the attribute but not the commitment itself

iii. hidden (A_h): the attribute is completely hidden to the issuer

- **keys:**
  - single master key for each user (m_1)
  - single master key for the Issuer – for creation of CL signatures
- **pseudonyms:**
  - a single domain pseudonym for a user per domain: generated as as

    $$\mathrm{dom}^{m_1}$$

    where dom is the public key of a domain, and m_1 is the user's master key
  - pseudonyms are unlinkable


„certificate"
CA
attributes
some attributes
+
anonymous credentials
verifier
derives from „certificate"

Cryptographic schemes used by Idemix

CL signatures:

- RSA group, special choice of primes: $p = 2p' + 1$, $q = 2q' + 1$, where p' and q' are primes
- choose at random quadratic residues: $R_1, ..., R_l, Z, S$
- public key: $(n, R_1, ..., R_l, Z, S)$, private key: p, q (enabling computation of roots $\bmod n$)
- security based on **Strong RSA assumption**: it is infeasibile to compute e-roots for $e > 2$
- signature for messages $m_1, ..., m_l$:
  - choose v at random and a prime $e > 2$ of length higher than each $m_1, ..., m_l$
  - $A := ((Z/(S^v \cdot \prod R_i^{m_i}))^{1/e}$
  - the signature is (A, e, v)
- verification: check if

$$Z = A^e \cdot S^v \cdot \prod R_i^{m_i} \quad ?$$

Manipulation attempt

- $m_i \to m_i'$
- $R_i^{m_i} \to R_i^{m_i'}$
- $A := A \cdot (R_i^{m_i' - m_i})^{1/e}$

exponentiation is the problem!

computing $\sqrt[e]{r} \mod n$

is infeasible without p, q

---

computing $r^{1/e}$ as well

each time use a different e since otherwise from: $a^{1/e}$ and $b^{1/e}$ the adversary could compute $(ab)^{1/e}$

$$r^{1/e} \bmod n$$

$$\equiv$$

$$r^{1/e} \bmod p \,, \quad r^{1/e} \bmod q$$

$$r^{1/e} \bmod p = r^{z} \bmod p$$

$$z = \frac{1}{e} \bmod p-1$$

$$ze = 1 \bmod p-1$$

Issuing a certificate for values $m_1, \ldots, m_l$

- somewhat complicated since the Issuer can learn only some attributes to be signed
- **method**: a two-party protocol to compute CL signature of the Issuer, algorithm draft:
  - the user chooses v' at random and computes $U := S^{v'} \cdot \prod R_i^{m_i}$ apart from known attributes that are not included in the product $\prod R_i^{m_i}$

$$U = S^{v'} \cdot \prod R_i^{m_i} \longleftrightarrow C(m_i)$$

m_i unknown

$U = S^{v'} \cdot \prod R_i^{m_i}$
unknown m_i

- the user creates a ZKP that U computed in this way, in particular that
  - the user knows hidden attributes
  - the user uses the same attributes as commited
- the issuer checks the ZKP proofs
- the issuer chooses at random: v'' and a prime e
- the issuer computes

$$Q := Z/(U \cdot S^{v''} \cdot \prod_{\text{known}\, m_i} R^{m_i}) \text{ and } A := Q^{1/e}$$

- (A, e, v'') is sent to the user together with a ZKP proof of corectness
- the user computes $v := v' + v''$, checks the proof and validity of signature (A, e, v)

Presenting a credential

complicated: also involves proofs over encrypted values and the range of attributes. Some attributes may be revealed, but some must stay hidden.

Moreover, **the certificate must not be revealed** (to ensure unlinkability) .

some details for verification of certificate without revealing it:

- value $\widetilde{m_i}$ is chosen for each hidden attribute m_i, that is, $i \in A_{\bar{r}}$
- the user chooses r_A at random and randomizes (A, e, v):
  - $A' := A \cdot S^{r_A}$, $v' := v - e \cdot r_A$

$$S^{r_A} \cdot A = \left(\frac{Z}{S^{v} \cdot \Pi} \right)^{\frac{1}{e}} \cdot S^{r_A}$$

$$v \to -r_A \cdot e$$

- so called t-values computed:
  - chosen at random: $\tilde{e}$,$\tilde{v}'$
  - $\tilde{Z} := (A')^{\tilde{e}} \cdot S^{\tilde{v}'} \cdot \prod R^{\widetilde{m_i}}$
- these t-values $\tilde{Z}$ and t values from other proofs plus some other data are hashed to get challenge c
- signatures components (s-values) are derived:
  - $\hat{e} := \tilde{e} + c \cdot e$
  - $\hat{v}' := \tilde{v}' + c \cdot v'$
  - $\hat{m}_i := \widetilde{m_i} + c \cdot m_i$

Credential verification - based on recomputation of t-values and recomputing c.

$\tilde{Z}$ recomputed as:

$$(A')^{\hat{e}} \cdot \prod_{i \in A_{\bar{r}}} R_i^{\widehat{m_i}} \cdot S^{\hat{v}'} / \left(\frac{Z}{\prod_{i \notin A_{\bar{r}}} R^{m_i}} \right)^c$$

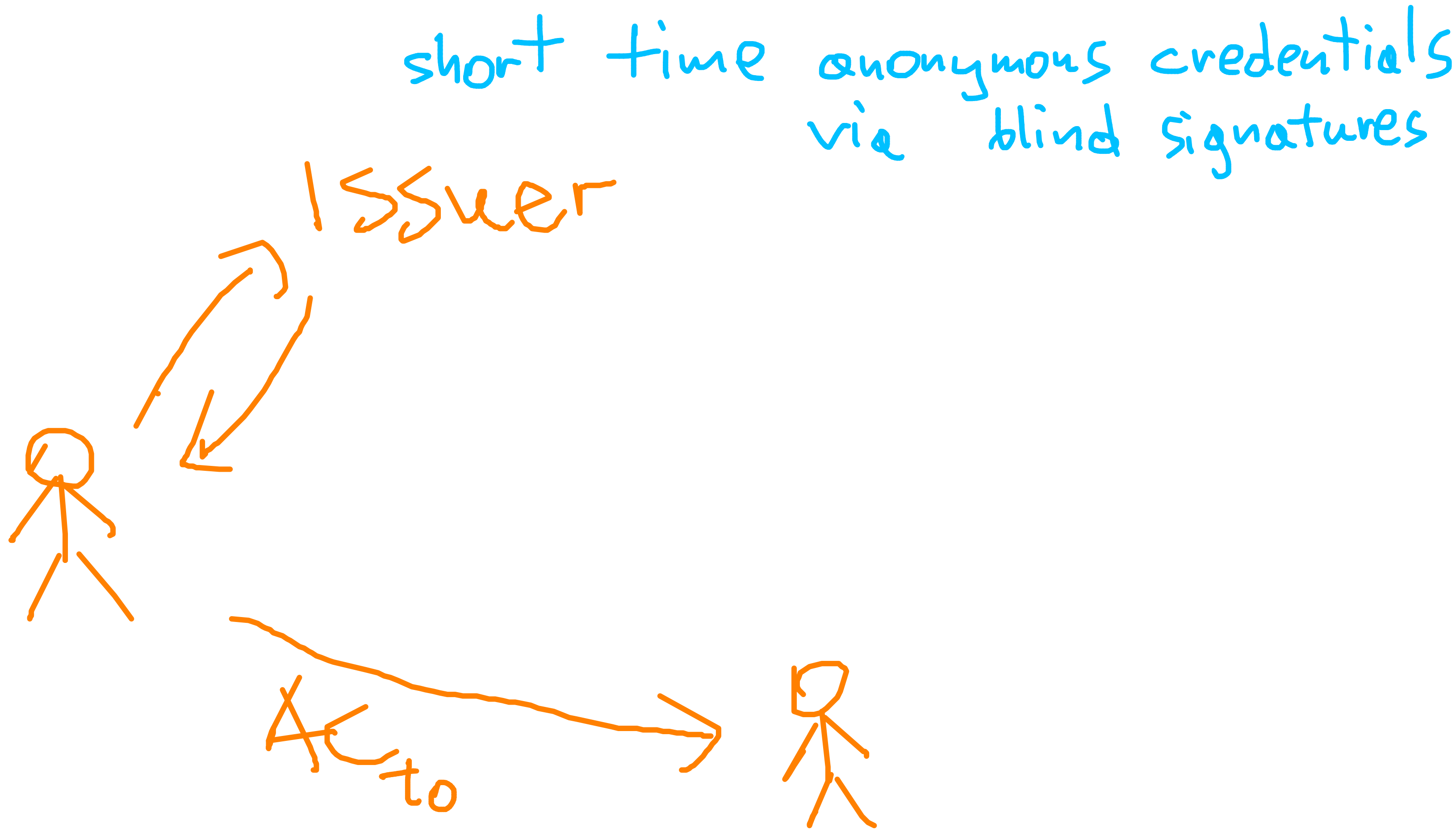
– we remove from Z the expressions R^m that correspond to the known attributes

– what is left will cancel the $c \cdot e$, $c \cdot v'$, $c \cdot m_i$ when using the exponents $\hat{e}$, $\hat{v}'$, $\hat{m}_i$

ranges have to be checked, etc

...

pragmatic future?



not only user2machine

IDENTIFICATION

running wireless communication protocol may enable tracing a user.

Threats:

- explicit exchange of identifiers: an eavsdropper learns who is communicating with whom
- strong cryptographic proofs created during identification: can be misused for proving presence to the third parties

elimination of explicit identifiers:

- at each communication round Alice and Bob create random nonce (nonces) for the next round
- even more secure: if n is such a nonce, then Alice uses n' where n' is the same as n except for a limited number of bits at random positions

  (so the adversary has to follow Alice and Bob without long interruptions)

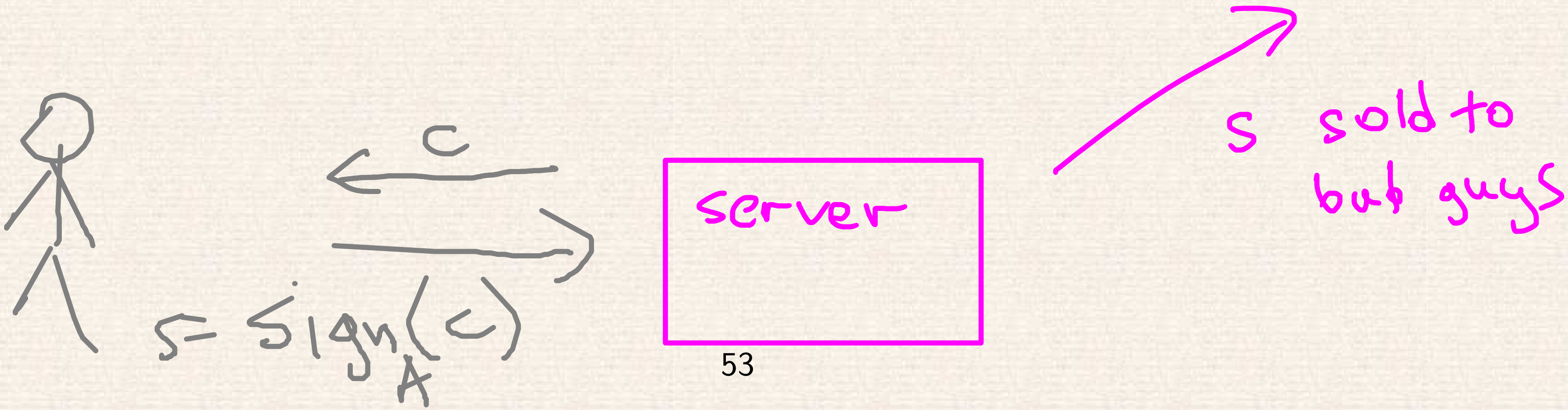
deniability:

- the idea is that a transcript of a communication (including the answer from the Prover created with his private key) can be simulated

  consequence: a third party has no grounds to believe the communication transcript presented to him

- **wrong example:** challenge-response algorithm with digital signature:

  1. the Verifier selects x at random and sends to the Prover
  2. the Prover returns his signature s over x



simple deniable protocol based on DH

Alice

key: $X = g^x$

server

Alice: X

random $W = g^w$

$\xleftarrow{\;W,\ R\;}$

$K := W^x$

$K := X^w$

$C = Enc_K(R)$

$\xrightarrow{\;C\;}$

$Dec_K(C) \stackrel{?}{=} R$

Deniability:

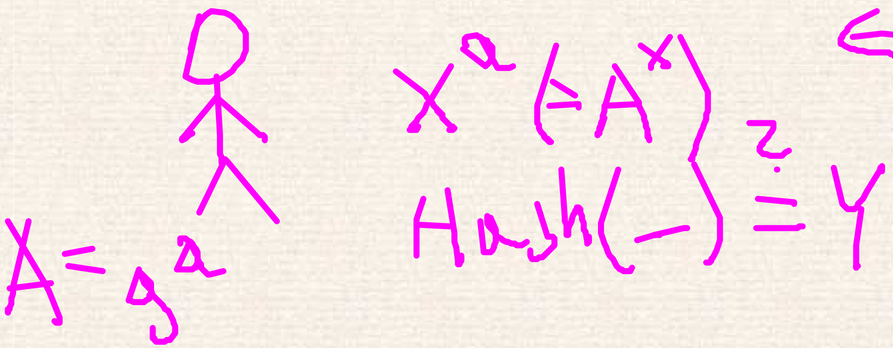
server can choose R and compute $C = Enc_K(R)$

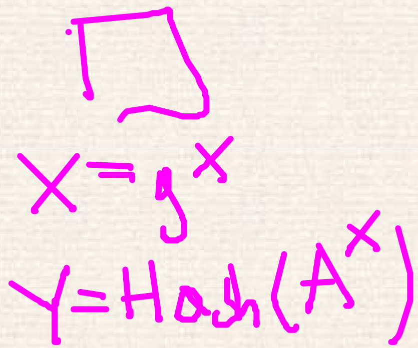
the server knows the response of Alice in advance unfortunately: s can serve as a proof of the claim of the Verifier: "I have talked to Prover" if x is a signature of the Verifier or somthing that only could be created by the Verifier

– **good example:** static Diffie-Hellman protocol

– **good example:** Stinson-Wu for Prover with the key pair $(a, A = g^a)$

1. Verifier chooses x at random, computes $X := g^x$ and $Y := \mathrm{Hash}(A^x)$
2. Verifier sends X, Y to Prover
3. Prover computes $Z := X^a$ and aborts if $Y \neq \mathrm{Hash}(Z)$
4. Prover sends Z
5. Verifier accepts iff $Z = A^x$





Stinson-Wu protocol

- Stinson-Wu does not create an oracle for DH Problem, Verifier must send a challege for which *somebody* knows x
- it is untrue that Verifier must know x:

```
Preparation:
```

- Eve creates correct X, Y as well as $\mathrm{Enc}_{\mathrm{Hash}(Z)}(x)$
- Eve sends these data to Verifier

Identification:

- Verifier sends X, Y to Prover
- Prover computes $Z := X^a$ and aborts if $Y \neq \mathrm{Hash}(Z)$
- Prover sends Z
- Verifier computes $\mathrm{Hash}(Z)$ and uses it as a key to decrypt and derive x
- Verifier accepts iff $Z = A^x$

Proof of Interaction: **Verifier returns x to Eve as a proof of interaction with Prover**

Anonymous Transactions

idea:

- transactions records publicly available in a distributed ledger (DLT) ⇒ undeniability, no backdating, possibility to detect double spending ([illegible]) , anti Money Laundering ([illegible])

- however, we must not create a public Big Brother

core mechanism for digital currencies:

cash hides money flow, this should be the key property of digital money as well

examples below will be taken from Monero

User keys and hidden recipient

user keys (EC notation):

- private keys a, b
- public keys: $A = a \cdot G$, $B = b \cdot G$
- sometimes (a, B) revealed (tracking key) – if the transactions have to be deanonymized

Creating transaction with a hidden recipient: (Alice sends to Bob)

- Alice fetches the public key (A, B)
- Alice chooses r at random, $R := r \cdot G$
- Alice generates one-time public key $P := \mathrm{Hash}(r \cdot A) \cdot G + B$
- Alice uses P as a one-time destination key for the transaction containing metadata R

Receiving a transaction by Bob

– Bob tries each transaction posted:

→ compute $P' := \mathrm{Hash}(a \cdot R) \cdot G + B$

→ if this is the right transaction, then $P = P'$ and Bob knows it is for him

– Bob calculates the one-time private key:

$$x = \mathrm{Hash}(a \cdot R) + b$$

– Bob can spend the money obtained in the transaction by signing with x

Remarks:

1: Receiving a transaction possible with (a, B), while (a, B) does not enable to compute x

2: Still only a partial anonymity: using x and the public key P would indicate who has got transaction with P from Alice

One time ring signatures

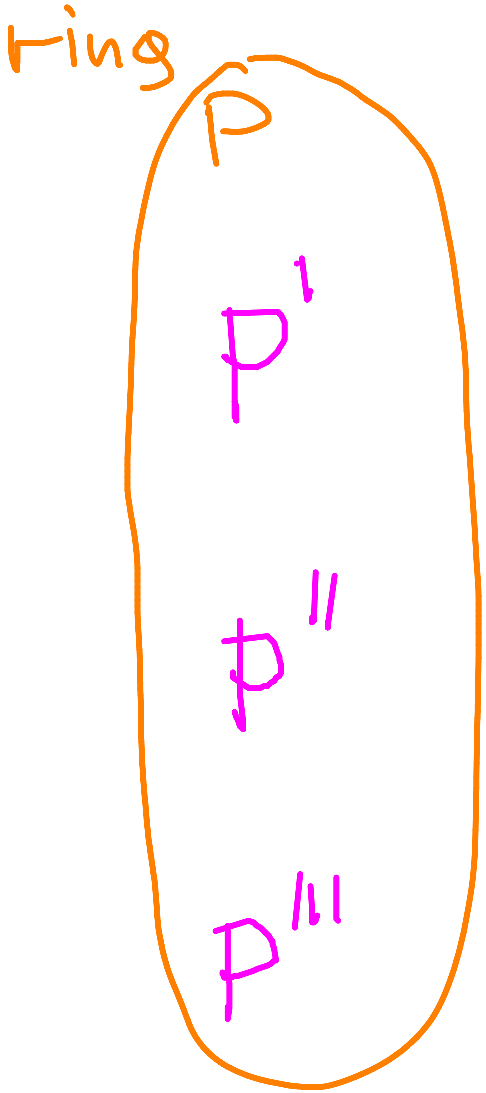
idea:

- instead of signing with x and showing P, a ring signature created:
  - a set of public keys $P_1, P_2, ..., P_m$ from transactions chosen at random (transaction value must be the same)
  - x used for signing
- any two ring signature of this kind created with x will be linked immediately

Goals achieved:

- double spending exposed
- m-anonymity concerning where the e-coin comes from

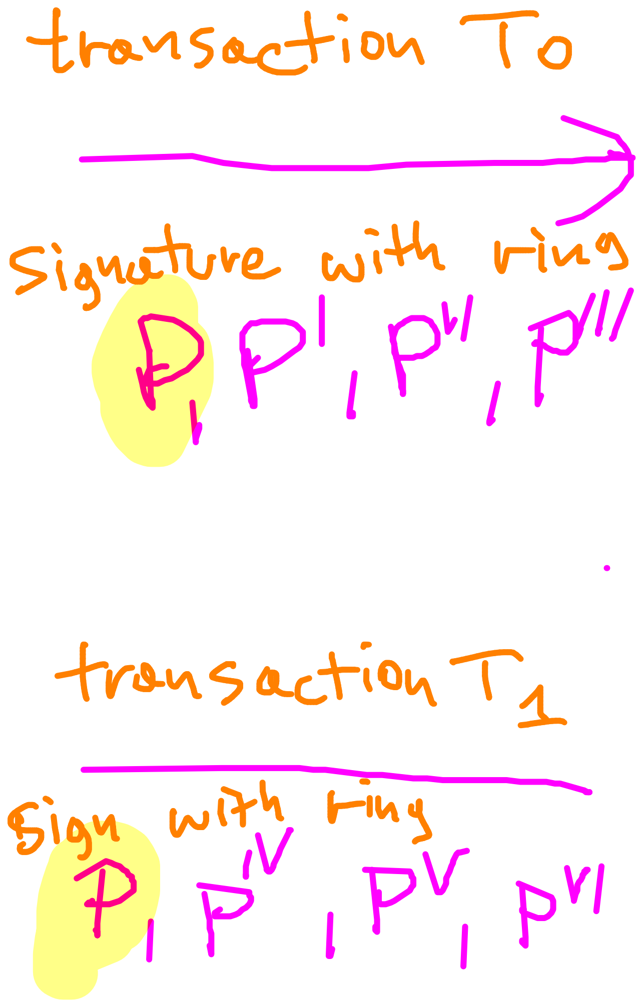
signing a transaction with P and private x for P



$sign_{P, P', P'', P''' ; x}(transaction)$

infeasible to say which private key has been used

double spending problem and detection



if the private key of P used twice, the both signatures contain the same "key image" I

Creating one-time ring signature

for key pair (x, P)

1. compute image key

$$I := x \cdot \mathrm{Hash}(P)$$

2. choose a ring of keys $P(P_{0,\ldots.}, P_n)$ where $P_s = P$ for some s
3. choose $q_0, \ldots., q_n$ at random
4. choose $w_0, \ldots., w_n$ at random, except for w_s
5. calculate for $i \neq s$

$$L_i := q_i \cdot G + w_i \cdot P_i$$

6. calculate $L_s := q_s \cdot G$

7. calculate for $i \neq s$

$$R_i := q_i \cdot \mathrm{Hash}(P_i) + w_i \cdot I$$

8. calculate $R_s := q_s \cdot \mathrm{Hash}(P_s)$

9. calculate the non-interactive challenge:

$$c := \mathrm{Hash}(\mathrm{message}, L_0, ..., L_n, R_0, ..., R_n)$$

10. calculate individual components:

   – for $i \neq s$: $c_i = w_i$, and $r_i = q_i$

   – $c_s := c - \sum_{i \neq s} c_i$

   – $r_s := q_s - c_s \cdot x$

11. output signature $(I, c_0, ..., c_n, r_0, ..., r_n)$

Verification

L_i recomputed as $L_i' := r_i \cdot G + c_i \cdot P_i$

R_i recomputed as $R_i' := r_i \cdot \mathrm{Hash}(P_i) + c_i \cdot I$

test:

$$\sum c_i = \mathrm{Hash}(\mathrm{message}, L_0', \ldots., L_n', R_1', \ldots., R_n')$$

Linking:

via the same I

Concept used:

to close the ring somewhere a schnorr signature must be created that applies to two generators simultaneously:

- P_s (which is hidden)
- I (which is explicit)

Many extensions possible (e.g. a transaction signed with multiple keys)

do not trust too much to anonymity of ring signatures

- if Bob holds private key for P' and Alice chooses P' for a ring, then for Bob the ring is smaller
- attack: flooding with own transactions and keys
- the rings have been small (size=5) now with size=10 somewhat better
- which keys to use for the ring?
  - uniformly from the past?
  - prefere the fresh ones as old keys are likely to be already used?