Distributed Verification of Mixing - Local Forking Proofs Model

Jacek Cichoń, Marek Klonowski, **Mirek Kutyłowski**

Wrocław University of Technology
Institute of Mathematics and Computer Science

ACISP'2008, Woolongong, 7.07.2008

Reaching anonymity
typical scenario

Input

- a batch of encrypted messages/documents
- the authors for each message is (more of less) known

Output

- plaintexts
- no link between the authors and the plaintexts

MIX

David Chaum

Steps executed by a mix

1. get a set of ciphertexts
2. decrypt and/or re-encrypt them
3. permute the results at random
4. output them

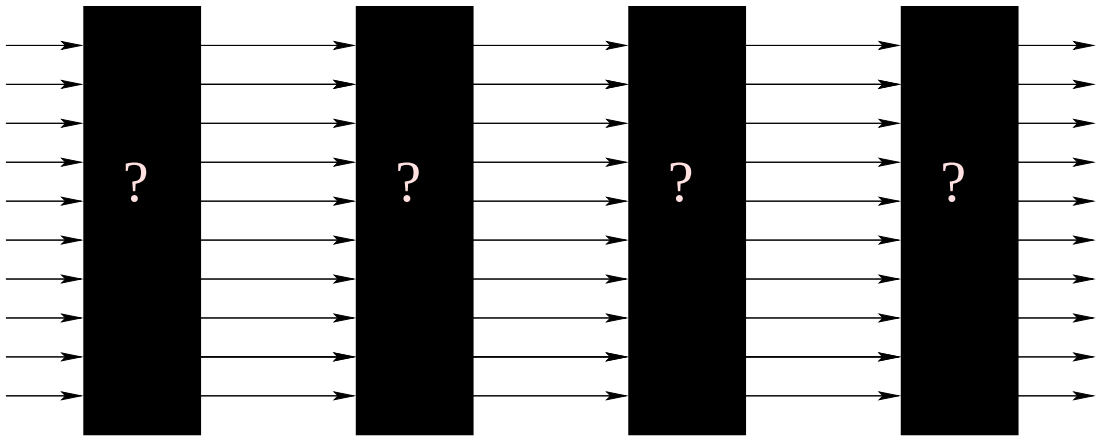
a perfect anonimizer as long as:

- cryptographic part does not leak information,
- the mix is honest.

Cascades of mixes
protocol

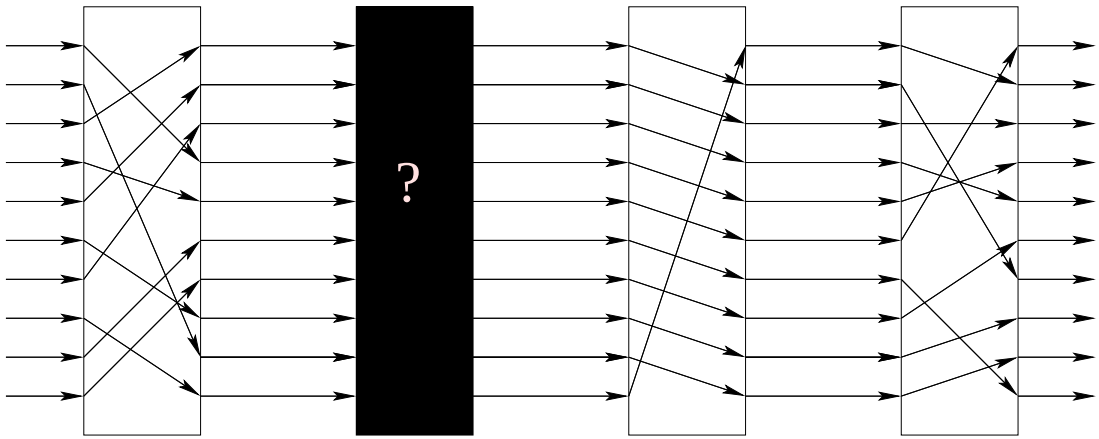
Anonymization process with k parties

- each party holds a mix,
- processing:
  1. the input goes to mix 1,
  2. mix i gets the input from mix $i-1$ (for $i > 1$) and sends its output to mix $i+1$ (for $i < k$),
  3. mix k gives the output of the cascade.



Cascades of mixes
anonymity

- perfect anonymity if at **at least one mix can be trusted**
- Alice may trust a different mix than Bob!



Correctness

Problem

How do we know that no mix

- modifies the messages?
- removes message?
- inserts own messages?

Correctness

Problem

How do we know that no mix

- modifies the messages?
- removes message?
- inserts own messages?

- It does not suffice that at least one mix can be trusted.
- **If at least one mix is cheating, then the plaintexts can be manipulated**

Applications

Anonymous communication in Internet

- messages sent to an Anonymizer encrypted with its public key,
- protocols for processing through many hops (e.g. TOR)

we admit that a message can be removed or modified, since it may occur anyway on the way to/from mixes

E-voting

requirements

the encrypted votes need to be mixed so that:

- anonymity is guaranteed
- a ballot cast must neither be modified nor replaced

achieving correctness is the critical issue

E-auctions

Requirements (for certain auctions)

offers come through anonymous communication channels:

- anonymity must be guaranteed: nobody should be able to say who is participating,
- an offer will neither be modified or replaced

achieving correctness is the critical issue

Re-encryption
with ElGamal

Modifying a ciphertext without decryption

- ciphertext $(a, b) = (m \cdot \beta^k, g^k)$
- re-encryption:

$$(a, b) := (a \cdot \beta^{k'}, b \cdot g^{k'})$$

for a random k'

(a, b) becomes $(m \cdot \beta^{k+k'}, g^{k+k'})$

Universal re-encryption
with ElGamal

Modifying a ciphertext without knowing public key

- ciphertext $(a, b, c, d) = (m \cdot \beta^k, g^k, \beta^m, g^m)$
- re-encryption:

$$(a, b, c, d) := (a \cdot c^{k'}, b \cdot d^{k'}, c^{k''}, d^{k''})$$

for random k', k''
(a, b, c, d) becomes $(m \cdot \beta^{k+mk'}, g^{k+mk'}, \beta^{mk''}, g^{mk''})$

Partial decryption
ElGamal

Forcing decryption by many parties

- ciphertext $(a, b) = (m \cdot (\beta_1 \beta_2 \ldots \beta_t)^k, g^k)$
- partial decryption:

$$(a, b) := (a / b^{x_1}, b)$$

where $g^{x_1} = \beta_1$

(a, b) becomes $(m \cdot (\beta_2 \ldots \beta_t)^k, g^k)$.

Proofs of knowledge

tools for showing correctness of re-encryption, decryption

ZKP of correct re-encryption

- given a, b and c, d, show that you know some k so that $a = c \cdot \beta^k$, $b = d \cdot g^k$
- or: $\log_\beta(a/c) = \log_g(b/d)$, i.e. equality of discrete logarithms

Proofs of knowledge

tools for showing correctness of re-encryption, decryption

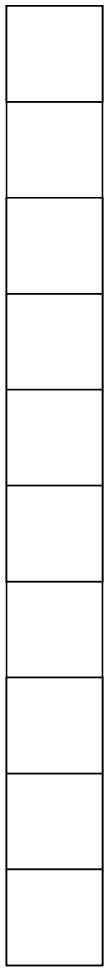
ZKP of correct re-encryption

- given a, b and c, d, show that you know some k so that $a = c \cdot \beta^k$, $b = d \cdot g^k$
- or: $\log_\beta(a/c) = \log_g(b/d)$, i.e. equality of discrete logarithms

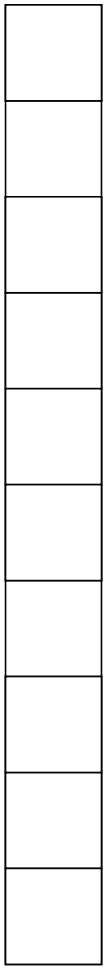
ZKP of correct re-encryption

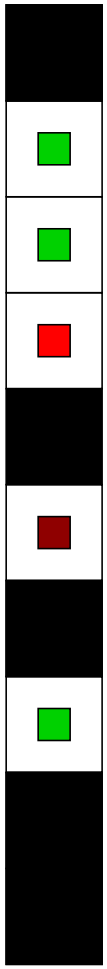
- given $(a_1, b_1), \ldots, (a_s, b_s)$ and c, d, show that you know some k so that for some (unrevealed) i: $a_i = c \cdot \beta^k$, $b_i = d \cdot g^k$
- or: $\log_\beta(a_i/c) = \log_g(b_i/d)$, i.e. equality of discrete logarithms with **some pair**

RPC
anonymization

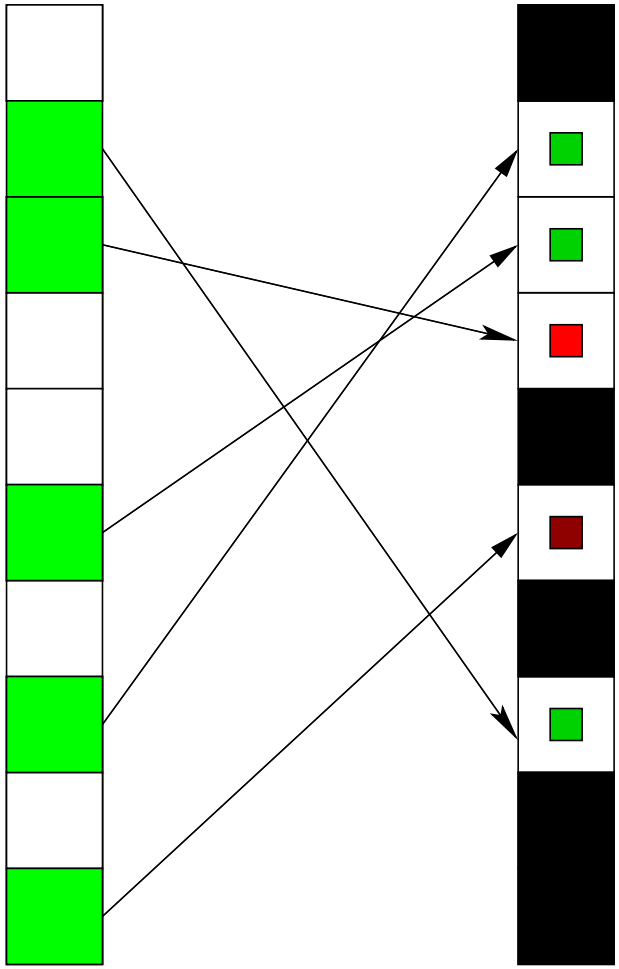


RPC
anonymization

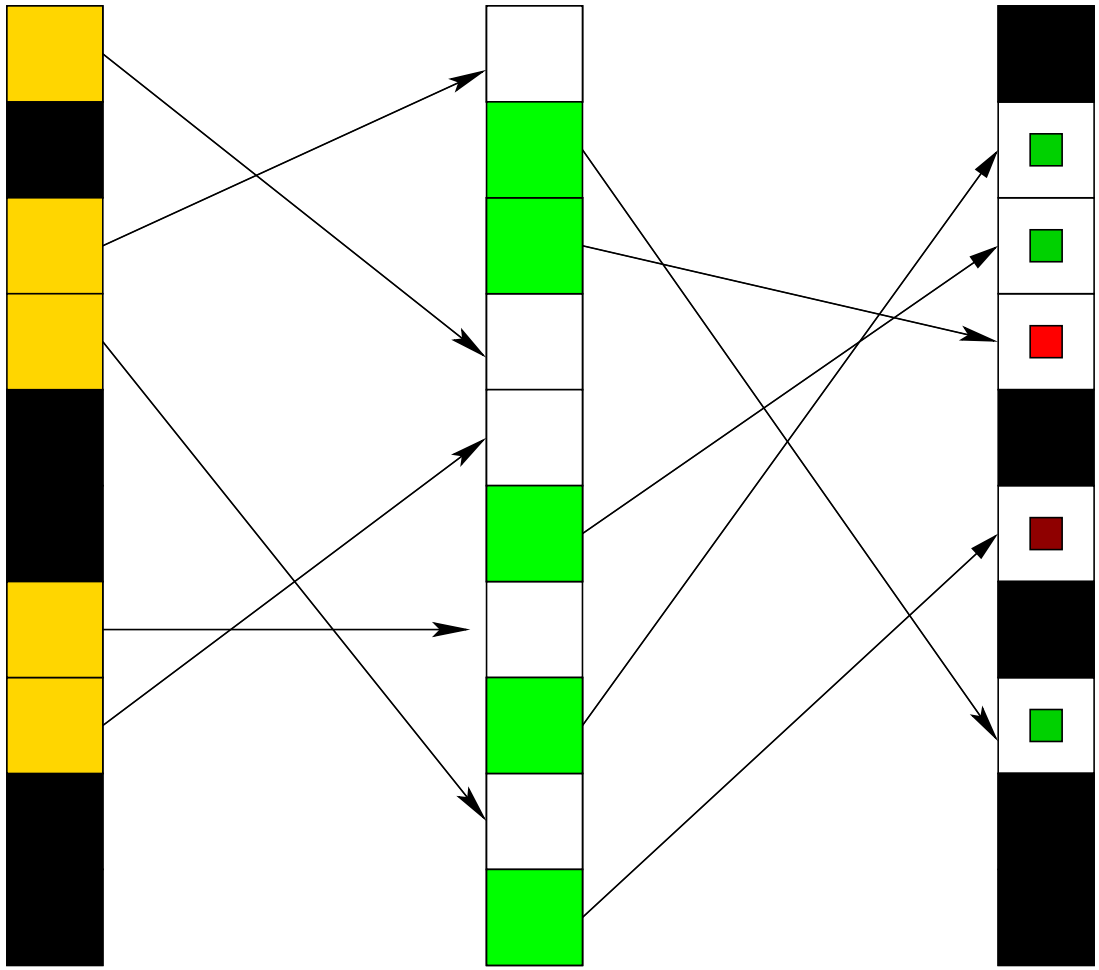




RPC
anonymization



RPC
anonymization



RPC

Properties

- 50% of links for each mix revealed
- no path of consecutive links revealed
- good properties in terms of probability distribution after $O(1)$ mixes

Verifiable mixing

Provable mixing

many very sophisticated techniques for specially designed mixing together with
a verification process:

Verification process

- input: mix input and output
- verification shows to **a third party** that mixing was correct

Verifiable mixing
complexity

Complexity issues

- one has to analyze the whole input and output of a mix
- the number of operations $c \cdot n$, where n is the number of elements in the input batch
- many sophisticated papers trying to reduce c, goal: go down towards $c = 1$

Main problem

- if Alice wants to check a mix, then she has to download the whole input and output.
- for applications like anonymizers in Internet or e-voting this is not a practical solution

Global versus local verifiability

Global verifiability

- a verification proof is performed on the whole input-output of a mix,
- everybody can check it himself, but it is necessary to download the data,
- ... or to trust an agent.

Global versus local verifiability

Global verifiability

- a verification proof is performed on the whole input-output of a mix,
- everybody can check it himself, but it is necessary to download the data,
- ... or to trust an agent.

Local verifiability

- everybody can check a chosen piece of the mixing process,
- any irregularity discovered by a single verifier shows that the mix was cheating,
- each verifier can download a small volume of data to perform local checking.

Local proofs for e-voting

Vote selling problem

- We cannot assume that the verifiers do not reveal the results of the proof – for the purpose of vote selling.
- The local proof should check the mix, but must not reveal the route of a message, even if the sender wants to reveal it.

Forking proof

Outline

- the method should be used for checking the mixes in a cascade,
- each mix works on big number of messages, (in cases where scalability problems make the classical solutions inefficient)
- it should work as a local verification procedure.

Forking proof

Setting

Assume that a mix processed ciphertexts

$$C_1, \ldots, C_n$$

and gave

$$C'_1, \ldots, C'_n$$

using a (hidden) permutation Π, that is C_i and $C'_{\Pi(i)}$ correspond to the same plaintext, for each i.

Forking proof

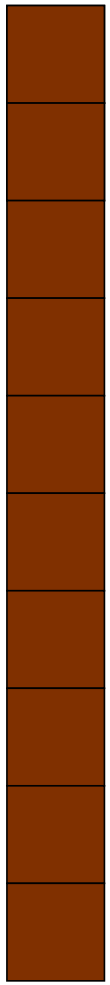
Verification protocol:

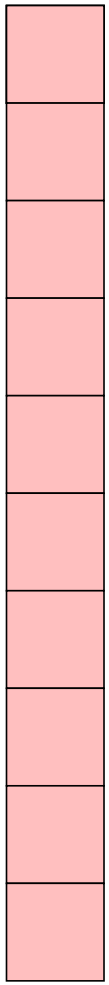
initialization: for each $i \leq n$, the mix determines a random set S_i of cardinality $k+1$ such that $\Pi(i) \in S_i$, (that is, $\Pi(i)$ is the only non-random element of S_i, the remaining k elements are chosen uniformly at random).

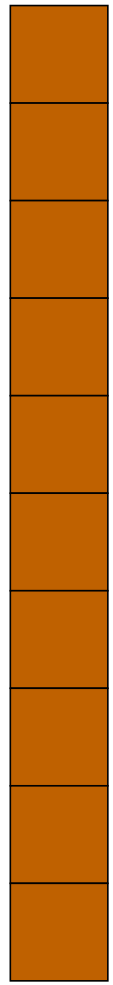
challenge: a verifier may challenge the mix with an arbitrary $i \leq n$,

response: the mix presents a proof that one of the ciphertexts C'_j for $j \in S_i$ corresponds to the same plaintext as C_i
(e.g. with ZKP, as mentioned before)

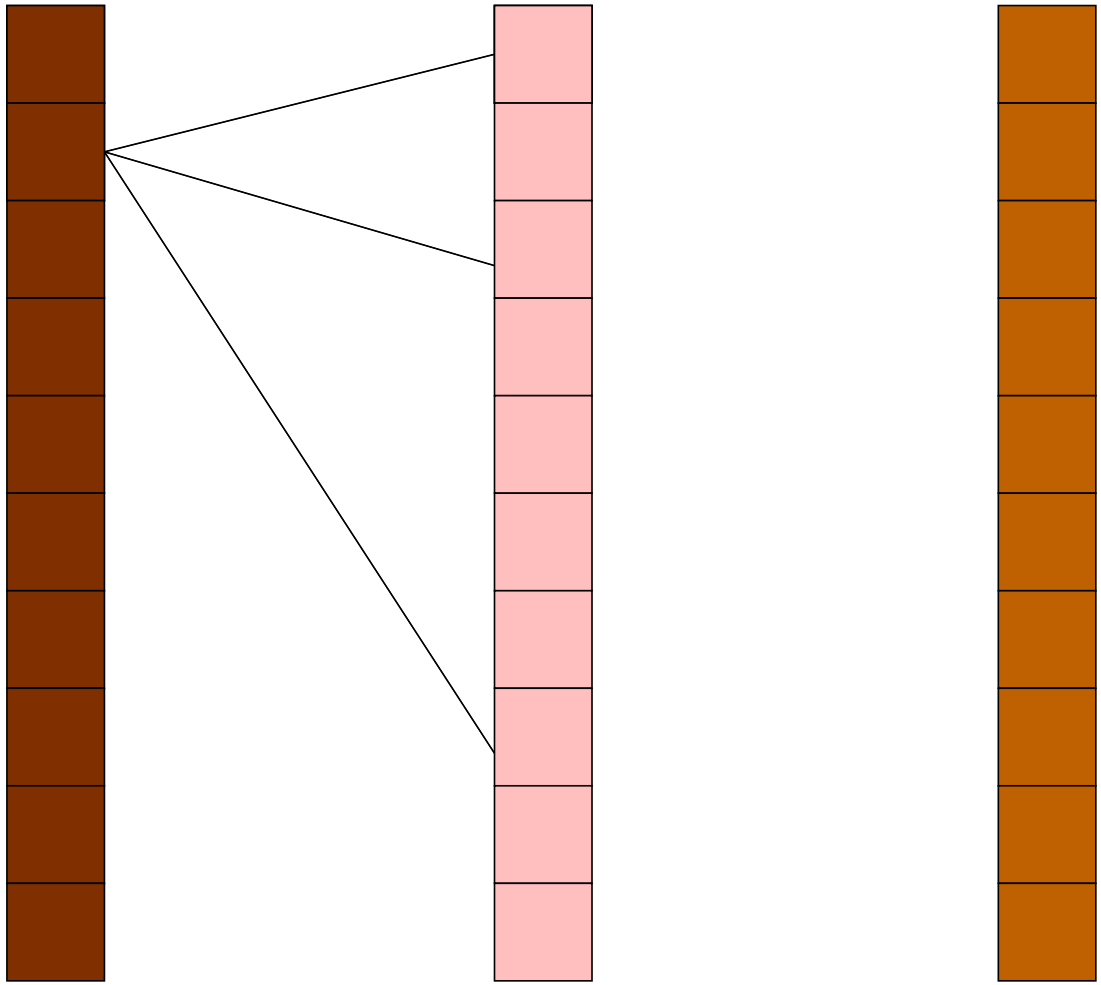
Forking proof



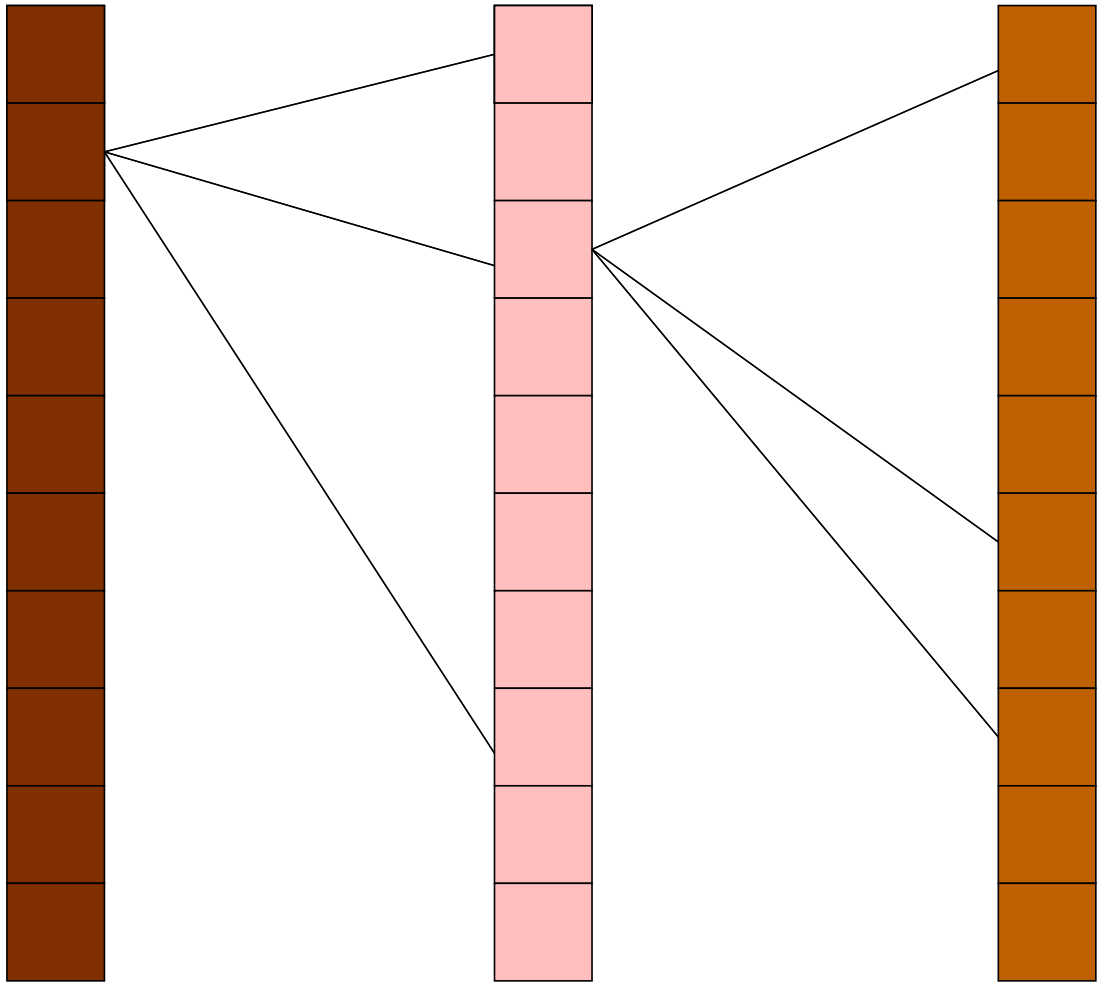




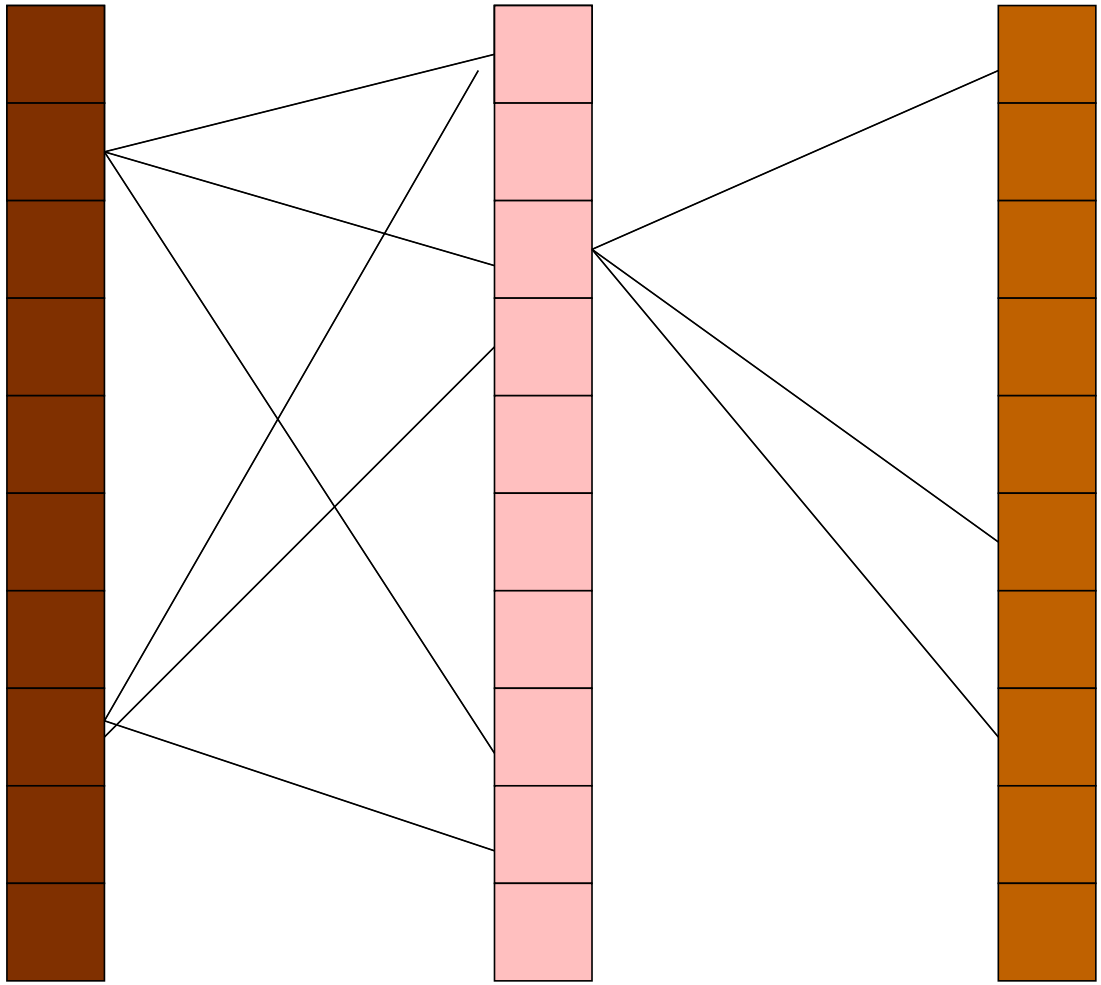
Forking proof



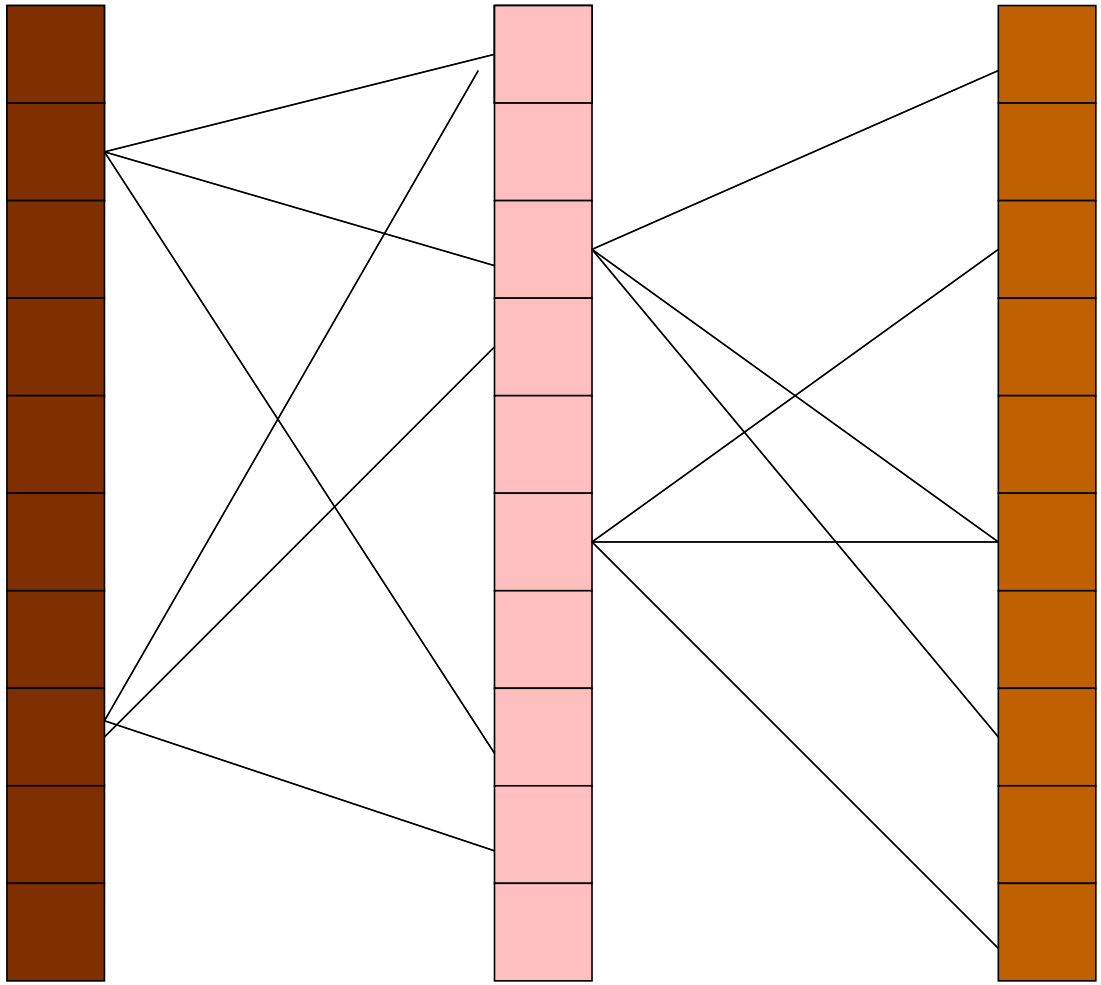
Forking proof



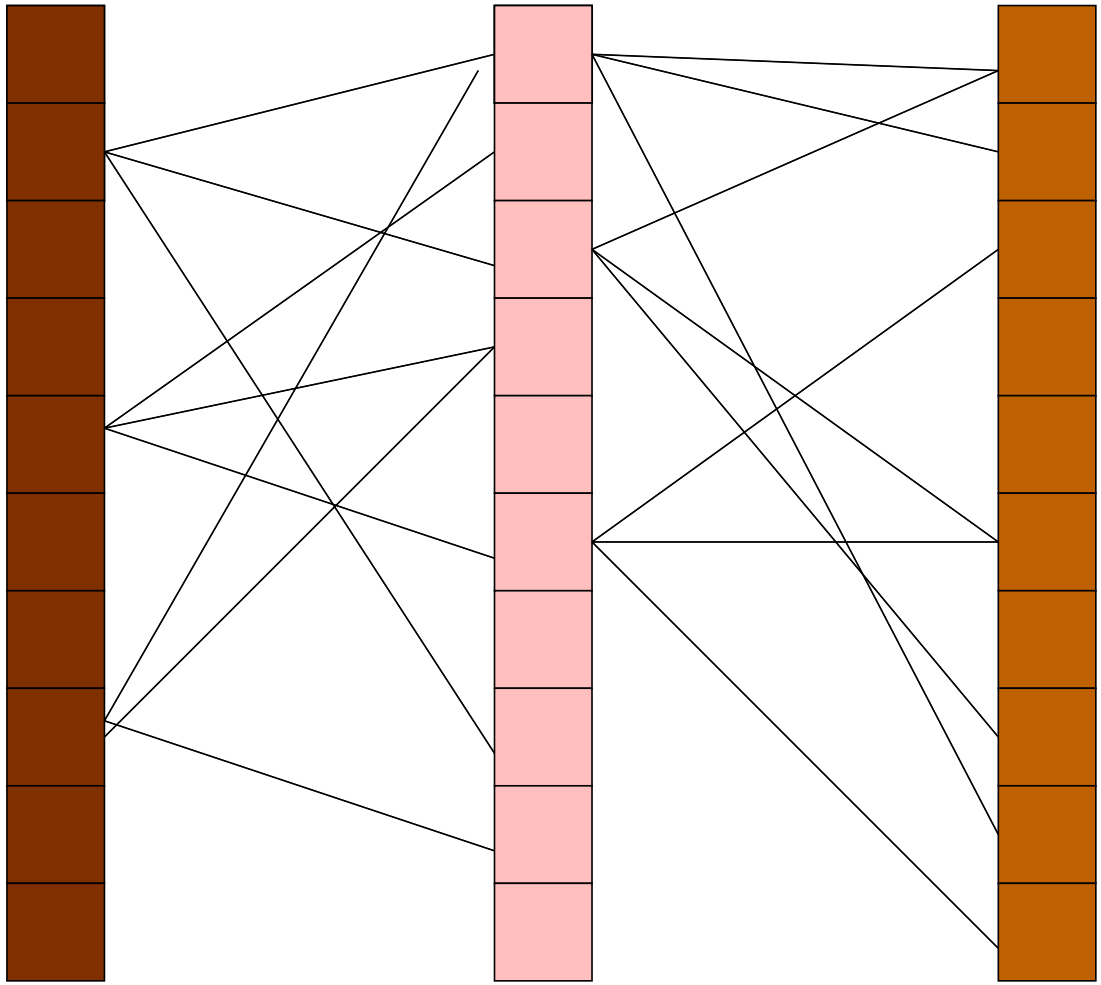
Forking proof



Forking proof



Forking proof



Forking proof

Details

- k might be a parameter - with bigger k we achieve more anonymity, at a cost of increasing communication volume,
- the verifiers can work independently

Application
e-voting

Properties

- a voter can check for sure that his vote has not been eliminated
- (with RPC this was only guaranteed with a certain probability)
- the voters that distrust the mixes can check more points

Problem
e-voting

Main problem

- the forking proof (just like RPC) reveals some information about mixing,
- can a voter use it to prove how he has voted?
- **can he show at least that has not cast a particular vote?**

Infection process

Anonymity

Which encrypted messages processed by the chain could hide the vote sent by Alice:

- after the first mix: exactly k ciphertexts,
- after the second mix each of k ciphertexts leads to k suspects on the output of the second mix,
- ...
- after each mix an additional number of ciphertexts may become candidates for the vote of Alice.

Infection process

Problem

- **how many mixes are necessary until all ciphertexts become infected?**
- obviously, $\log_k n$ mixes are necessary
- **does $c \cdot \log_k n$ suffice? For which c?**

The constants are important since they determine the number of mixes that have to be used in the system, and thereby the costs and speed of computing election results.

Random process

Description

- n different nodes,
- initially exactly one node is infected,
- for each step a regular directed graph with outdegree k is chosen at random,
- if a directed edge (a, b) is in the graph and a is infected, then b becomes infected as well.

Random process

Problem

- the speed of infection depends very much on the graphs chosen:
  - the edge from infected nodes may lead to the same node (infecting it twice),
  - the edges may lead to nodes already infected,
- **the time point of infecting all nodes is a random variable depending on the choice of the digraphs.**

Properties of the process

Phases

phase 1 :

- initially almost no conflicts, k nodes infected by an infected node at each step with high probability,
- gradually the number of nodes infected comes down,

Properties of the process

Phases

phase 1 :

- initially almost no conflicts, k nodes infected by an infected node at each step with high probability,
- gradually the number of nodes infected comes down,

phase 2 :

- it is hard to infect somebody new, but
- it is becoming harder to remain uninfected.

In fact, in the analysis we distinguish 3 phases.

Results

Time to infect all n nodes with probability $> 1 - \frac{1}{n}$

$$T \leq \left(0.8 + \frac{4.4}{k}\right) \log n + 1.7 \frac{\log\left(\frac{16}{k} \log n\right)}{\log\left(1 + \frac{k}{3}\right)} + \frac{\log(n/2)}{\log(1 + \frac{k}{4})}$$
$$+ \sqrt{2.7 \frac{\log\left(\frac{16}{k} \log n\right)}{\log\left(1 + \frac{k}{3}\right)} \log n + 0.65 \log^2 n}.$$

Future work

The work ahead of us:

- compute the moment, when probability distribution is more or less uniform on a large set of nodes,
- once it is done, delayed path coupling method can be applied to get the time when overall probability distribution is close to the uniform distribution with high probability.

thank you for your attention!

kutylowski.im.pwr.wroc.pl