Mixnets for Voting

Douglas Wikström
KTH Stockholm
dog@csc.kth.se

September 2, 2010

Outline

Goal of the Talk (and Disclaimer)

- Our goal is to give a conceptually simple description of mix-nets.
- We make no attempt to give fair historical account.
- We also ignore many variants and extensions of mix-nets. In particular, those that do not have applications in voting.

Simulation Paradigm

- Describe the normal behavior and expected security properties of your construction by an **ideal functionality**.

Simulation Paradigm

- Describe the normal behavior and expected security properties of your construction by an **ideal functionality**.
- The ideal functionality must be **secure by inspection**.

Simulation Paradigm

- Describe the normal behavior and expected security properties of your construction by an **ideal functionality**.
- The ideal functionality must be **secure by inspection**.
- Show that:
  for every **adversary** interacting with the real protocol,
  there is a **simulator** interacting with the ideal functionality,
  such that no outside **distinguisher** can distinguish:
  - the adversary and real protocol from
  - the simulator and ideal functionality.

General Security Model

- **Efficient Adversary.** The adversary must run in polynomial time.
- **Static Corruption.** The adversary decides before the execution starts which parties to corrupt.
- **Active Adversary.** The parties controlled by the adversary can execute any program.

General Security Model

- **Efficient Adversary.** The adversary must run in polynomial time.
- **Static Corruption.** The adversary decides before the execution starts which parties to corrupt.
- **Active Adversary.** The parties controlled by the adversary can execute any program.
- **Passive Adversary.** The adversary may look at the internals of corrupted parties, but not control their behavior.

General Security Model

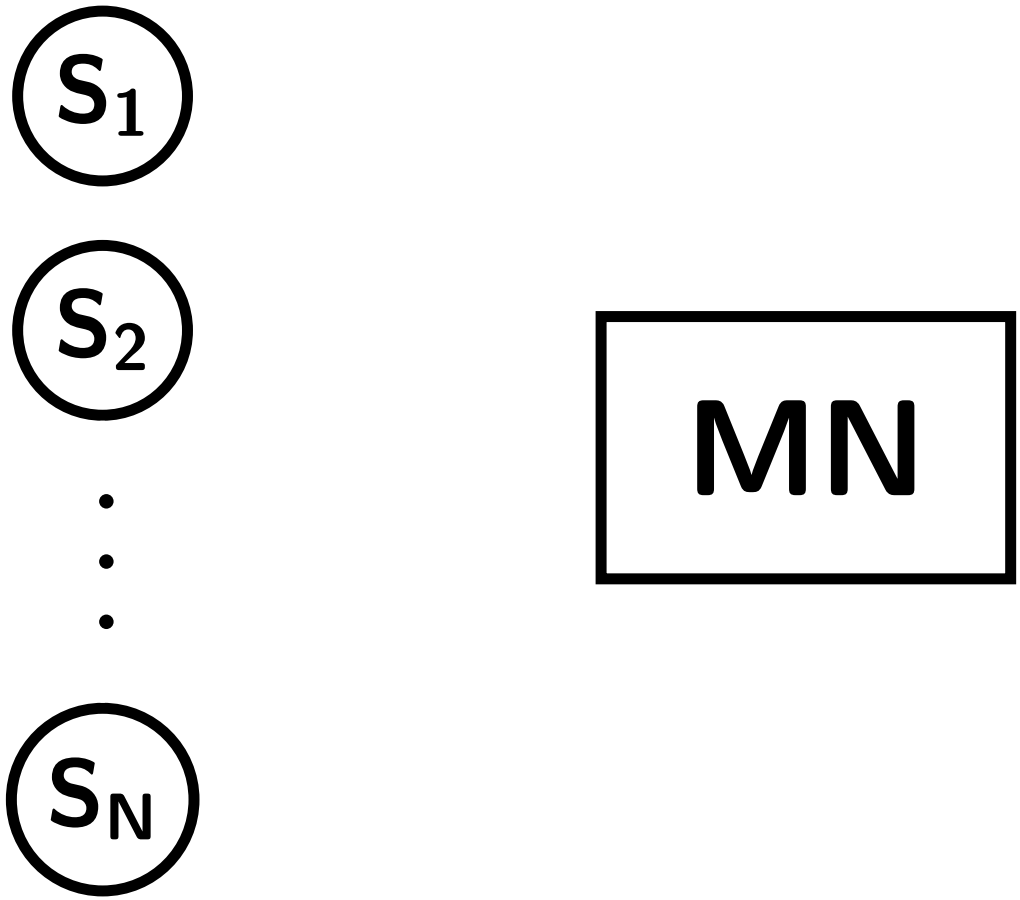
- **Efficient Adversary.** The adversary must run in polynomial time.
- **Static Corruption.** The adversary decides before the execution starts which parties to corrupt.
- **Active Adversary.** The parties controlled by the adversary can execute any program.
- **Passive Adversary.** The adversary may look at the internals of corrupted parties, but not control their behavior.
- **Scheduling.** In this talk we ignore the problem of corruption of the network. The results are mostly independent of this.

Mix-Net Basics

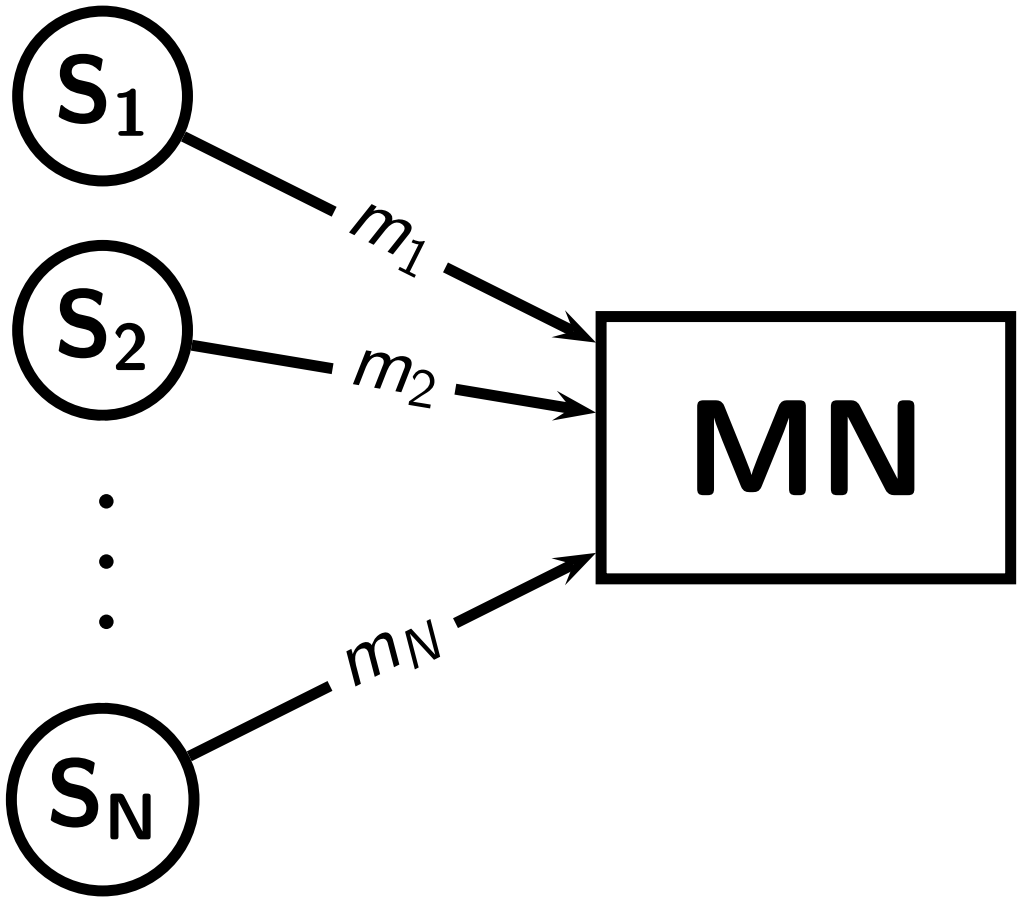
Ideal Mix-Net (1/5)

- There are N senders, $S_1, \ldots, S_N$.
- Each S_i wants to send/publish a message m_i.
- The senders wish to keep secret who sent/published which message.

Ideal Mix-Net (2/5)

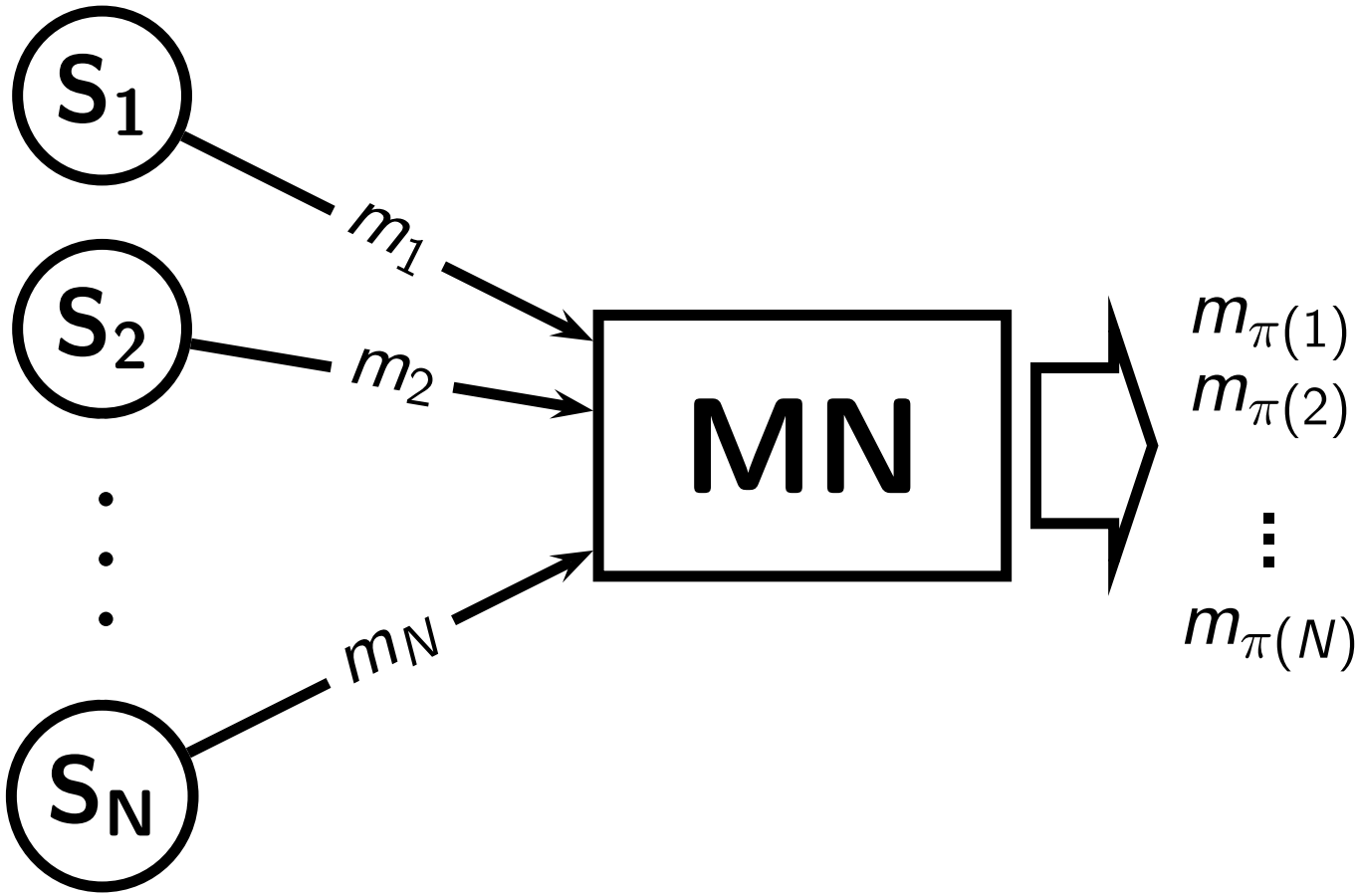


Ideal Mix-Net (2/5)



The messages are submitted using secret channels!

Ideal Mix-Net (2/5)



The messages are submitted using secret channels!

Ideal Mix-Net (3/5)

How many messages can be input by each sender?

- Typically, each sender may submit at most one message when a mix-net is used in an election.
- In this talk we assume that each sender submits at **exactly** one message.

Ideal Mix-Net (4/5)

Who decides when the mix-net produces the output?

- A group of authorities formally make the decision that an election is closed and this typically takes place at a pre-determined time.
- In this talk, we assume that the mix-net produces its output when it has received exactly one message from each sender.
- In a real mix-net the operators decide.

Ideal Mix-Net (5/5)

Which senders can be corrupted?

- We assume that the adversary can corrupt any sender it wants.
- We can obviously not provide any anonymity for corrupted senders. They **are** the adversary.
- Privacy must hold within the set of non-corrupted senders.

Real Mix-Nets

- An (almost) ideal mix-net is what we want in the end.
- A protocol that implements an ideal mixnet is called a **mix-net** (or **mix**, **anonymizer**, etc).
- The parties that execute the protocol (along with the senders) are called **mix-servers** (or **mix-centers**, **mixers**, etc).

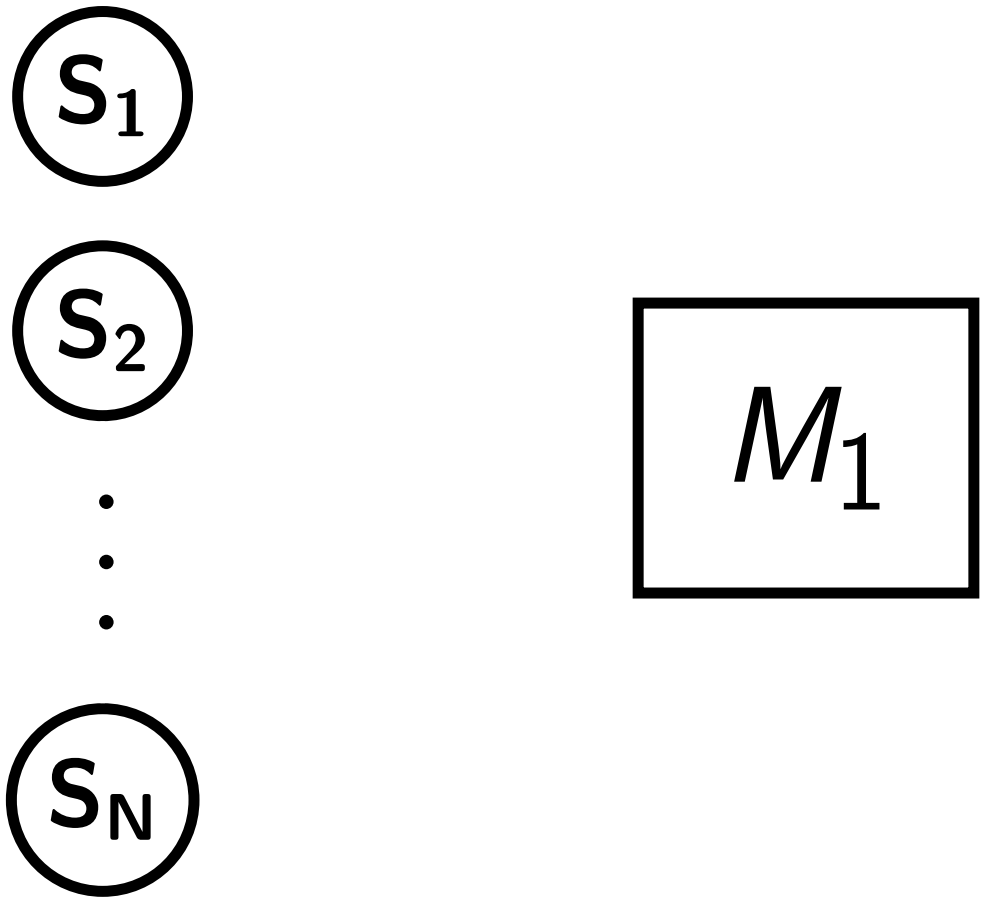
Notation

- $M_1, \ldots, M_k$ are mix-servers, **with** k **small**.
- $S_1, \ldots, S_N$ are senders, **with** N **large**.

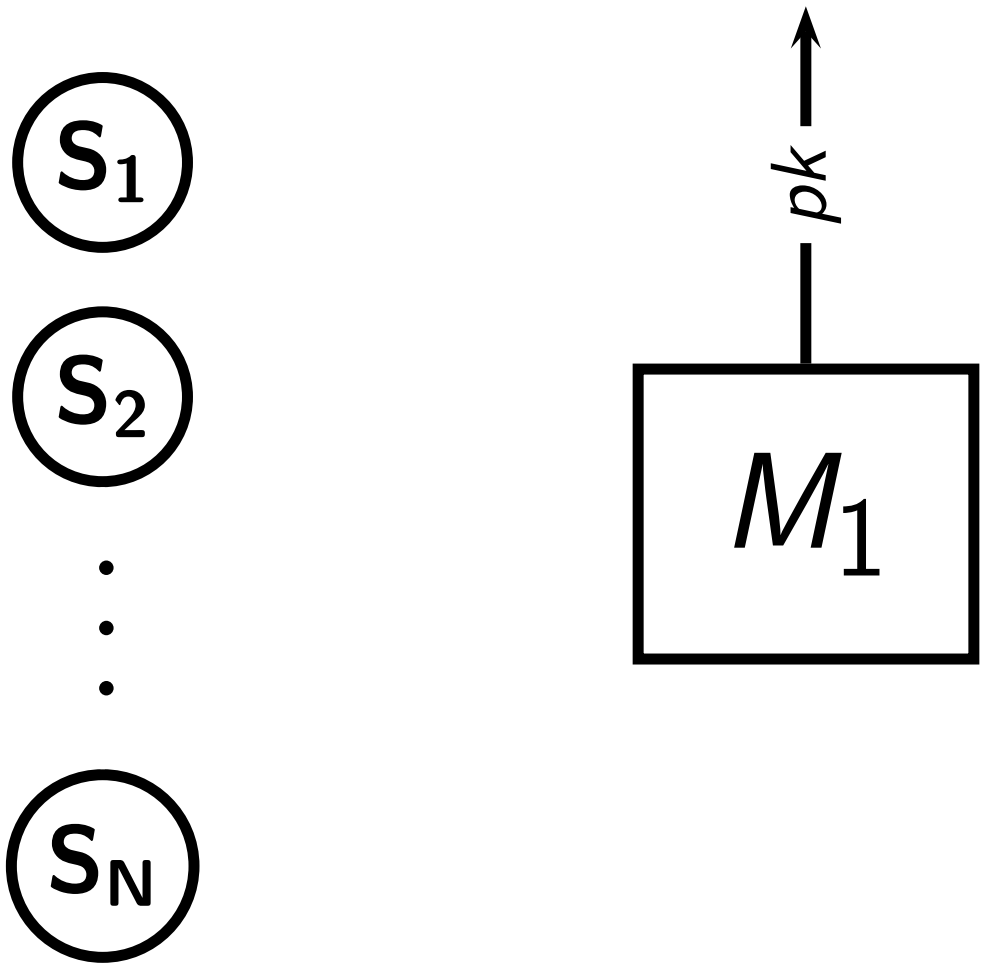
Question

How do we implement a mix-net using a single trusted mix-server in a provably secure way?

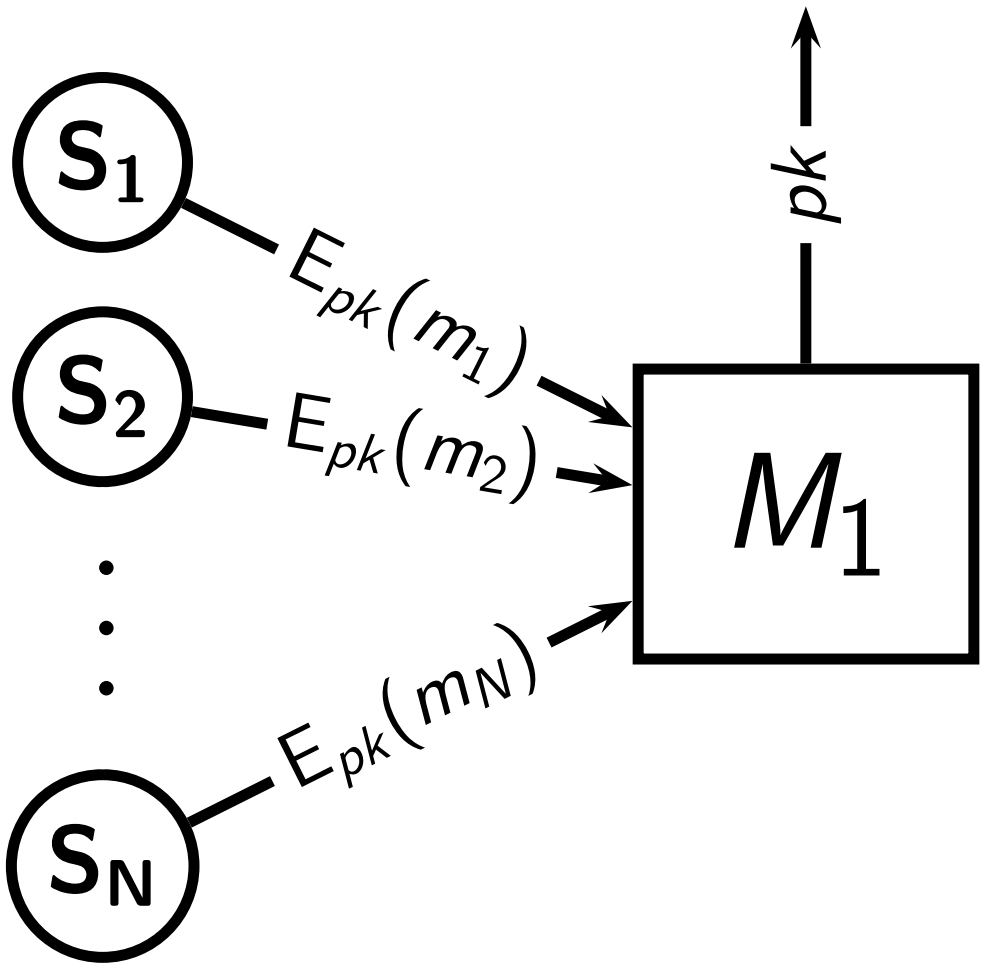
Single Trusted Mix-Server (1/3)



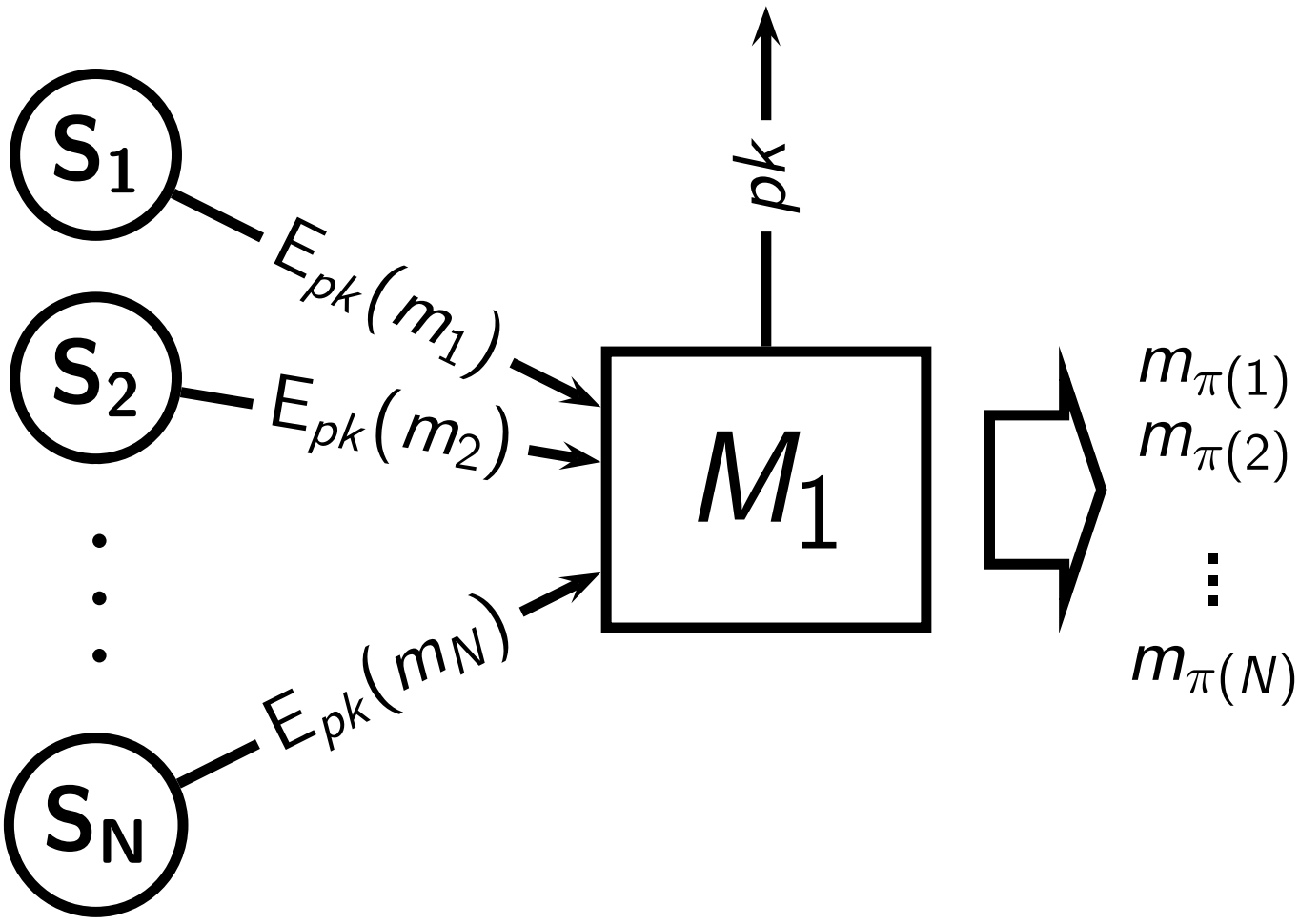
Single Trusted Mix-Server (1/3)



Single Trusted Mix-Server (1/3)



Single Trusted Mix-Server (1/3)



Single Trusted Mix-Server (2/3)

Theorem

If the cryptosystem is semantically/CCA2 secure, then the trusted mix-server is a secure implementation of a mix-net for passive/active adversaries.

Single Trusted Mix-Server (2/3)

Theorem

If the cryptosystem is semantically/CCA2 secure, then the trusted mix-server is a secure implementation of a mix-net for passive/active adversaries.

Proof.

Consider a simulator that accepts pk as input and:

1. extracts the plaintext m_i from the ciphertext $\mathrm{E}_{pk}(m_i)$ of each corrupted sender by inspection/a decryption query,

Single Trusted Mix-Server (2/3)

Theorem

If the cryptosystem is semantically/CCA2 secure, then the trusted mix-server is a secure implementation of a mix-net for passive/active adversaries.

Proof.

Consider a simulator that accepts pk as input and:

1. extracts the plaintext m_i from the ciphertext $\mathrm{E}_{pk}(m_i)$ of each corrupted sender by inspection/a decryption query,
2. replaces the plaintext m_i of each honest sender by some "junk" message, i.e., $\mathrm{E}_{pk}(m_i)$ is replaced by $\mathrm{E}_{pk}(1)$, and

Single Trusted Mix-Server (2/3)

Theorem

If the cryptosystem is semantically/CCA2 secure, then the trusted mix-server is a secure implementation of a mix-net for passive/active adversaries.

Proof.

Consider a simulator that accepts pk as input and:

1. extracts the plaintext m_i from the ciphertext $\mathrm{E}_{pk}(m_i)$ of each corrupted sender by inspection/a decryption query,
2. replaces the plaintext m_i of each honest sender by some "junk" message, i.e., $\mathrm{E}_{pk}(m_i)$ is replaced by $\mathrm{E}_{pk}(1)$, and
3. outputs the full list of messages in random order as before.

Single Trusted Mix-Server (2/3)

Theorem

If the cryptosystem is semantically/CCA2 secure, then the trusted mix-server is a secure implementation of a mix-net for passive/active adversaries.

Proof.

Consider a simulator that accepts pk as input and:

1. extracts the plaintext m_i from the ciphertext $\mathrm{E}_{pk}(m_i)$ of each corrupted sender by inspection/a decryption query,
2. replaces the plaintext m_i of each honest sender by some "junk" message, i.e., $\mathrm{E}_{pk}(m_i)$ is replaced by $\mathrm{E}_{pk}(1)$, and
3. outputs the full list of messages in random order as before.

Indistinguishability of the simulator and a real execution + hybrid argument $\Longrightarrow$ reduction to the security of the cryptosystem.

Single Trusted Mix-Server (2/3)

Theorem

If the cryptosystem is semantically/CCA2 secure, then the trusted mix-server is a secure implementation of a mix-net for passive/active adversaries.

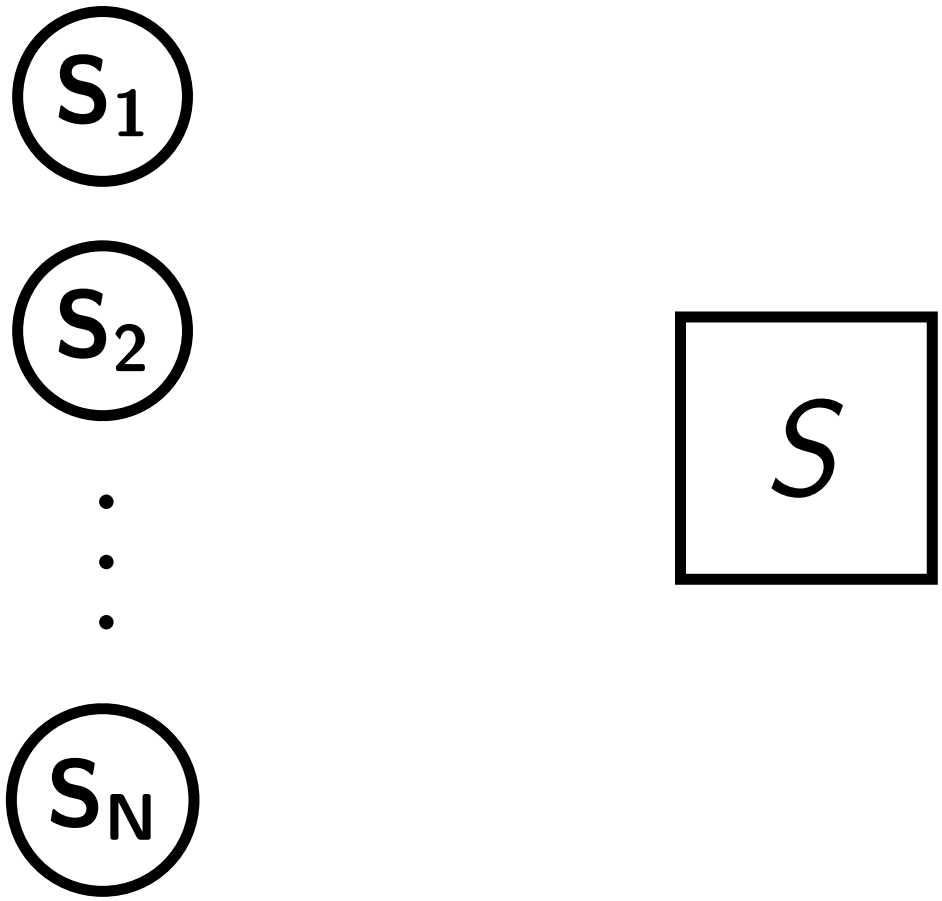
Proof.

Consider a simulator that accepts pk as input and:

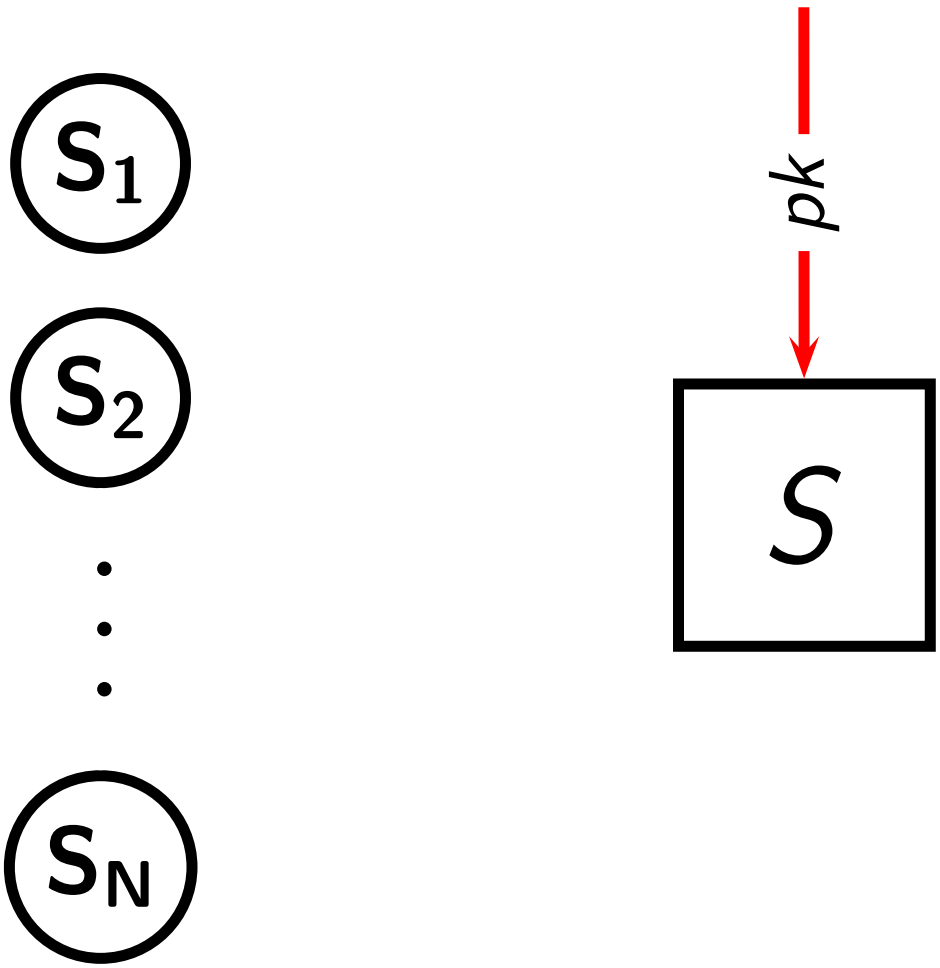
1. **extracts the plaintext** m_i from the ciphertext $\mathsf{E}_{pk}(m_i)$ of each **corrupted sender** by inspection/a decryption query,
2. replaces the **plaintext** m_i of each **honest sender** by some **junk** message, i.e., $\mathsf{E}_{pk}(m_i)$ is replaced by $\mathsf{E}_{pk}(1)$, and
3. outputs the full list of messages in random order as before.

Indistinguishability of the simulator and a real execution + hybrid argument $\Longrightarrow$ reduction to the security of the cryptosystem.

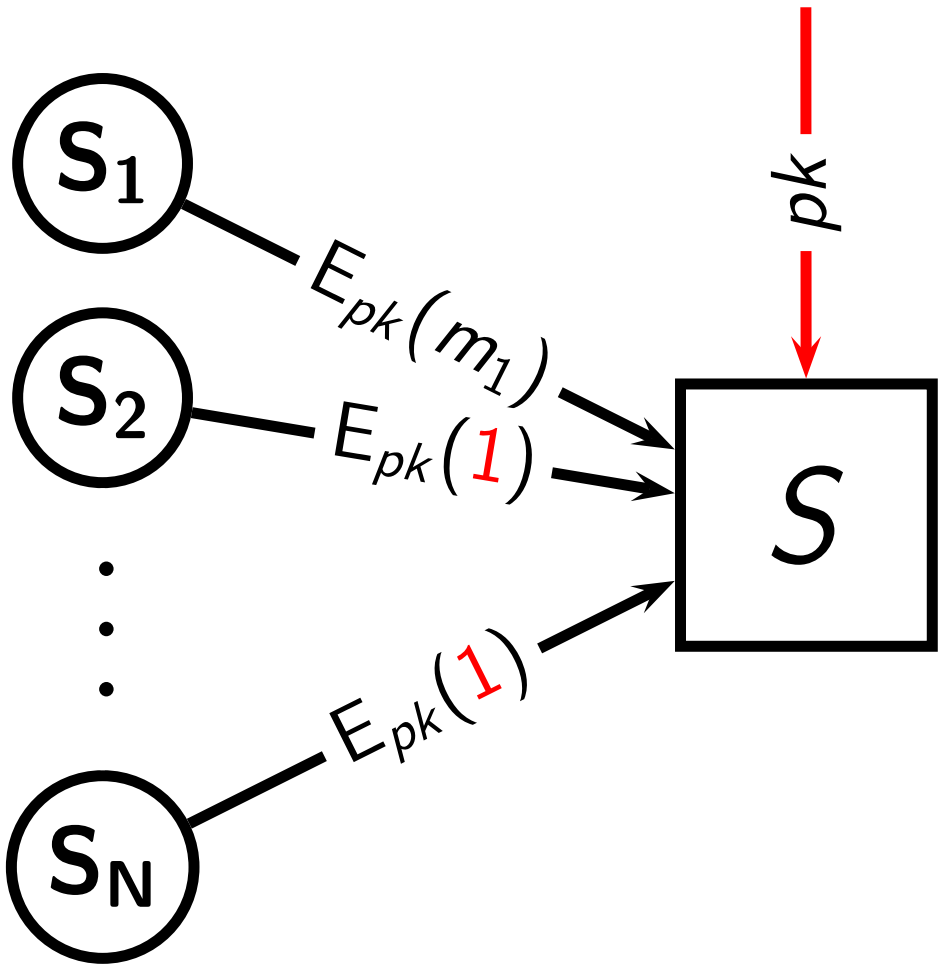
Single Trusted Mix-Server (3/3)



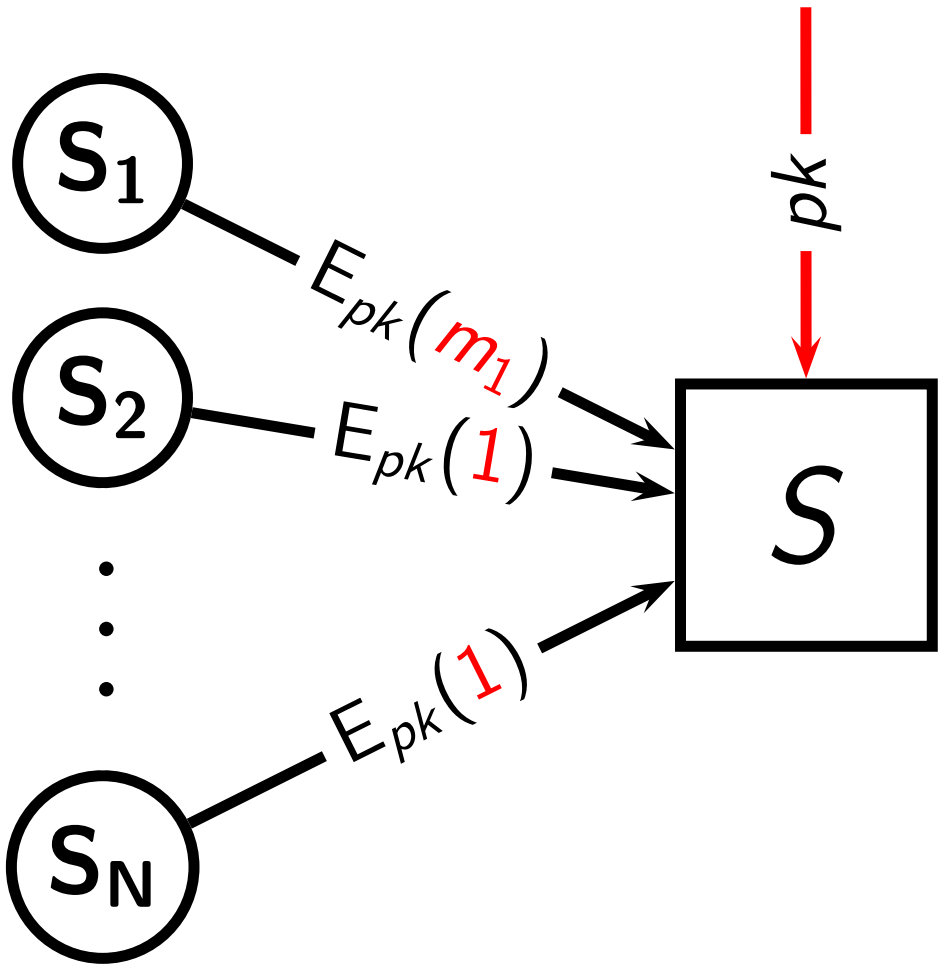
Single Trusted Mix-Server (3/3)



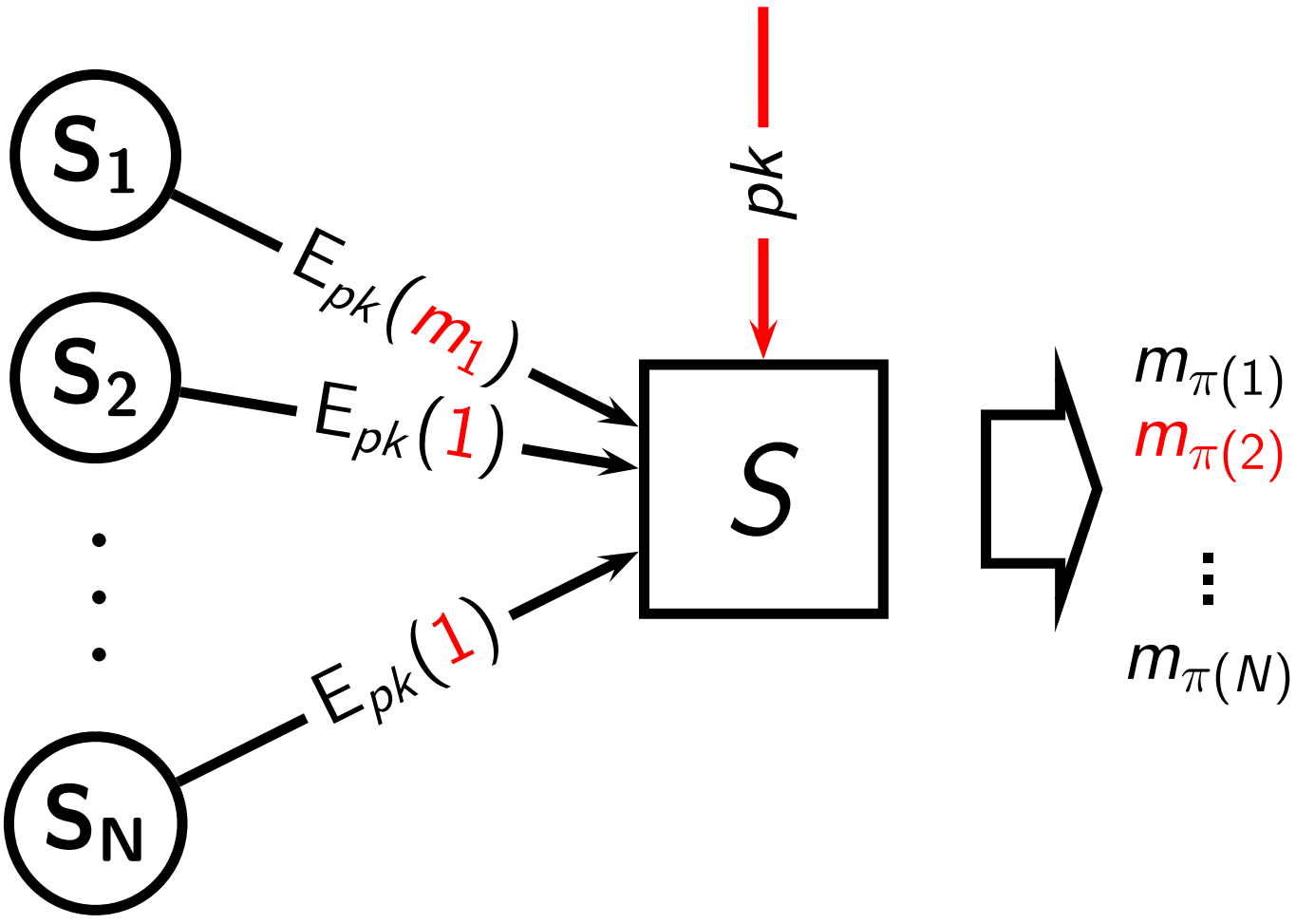
Single Trusted Mix-Server (3/3)



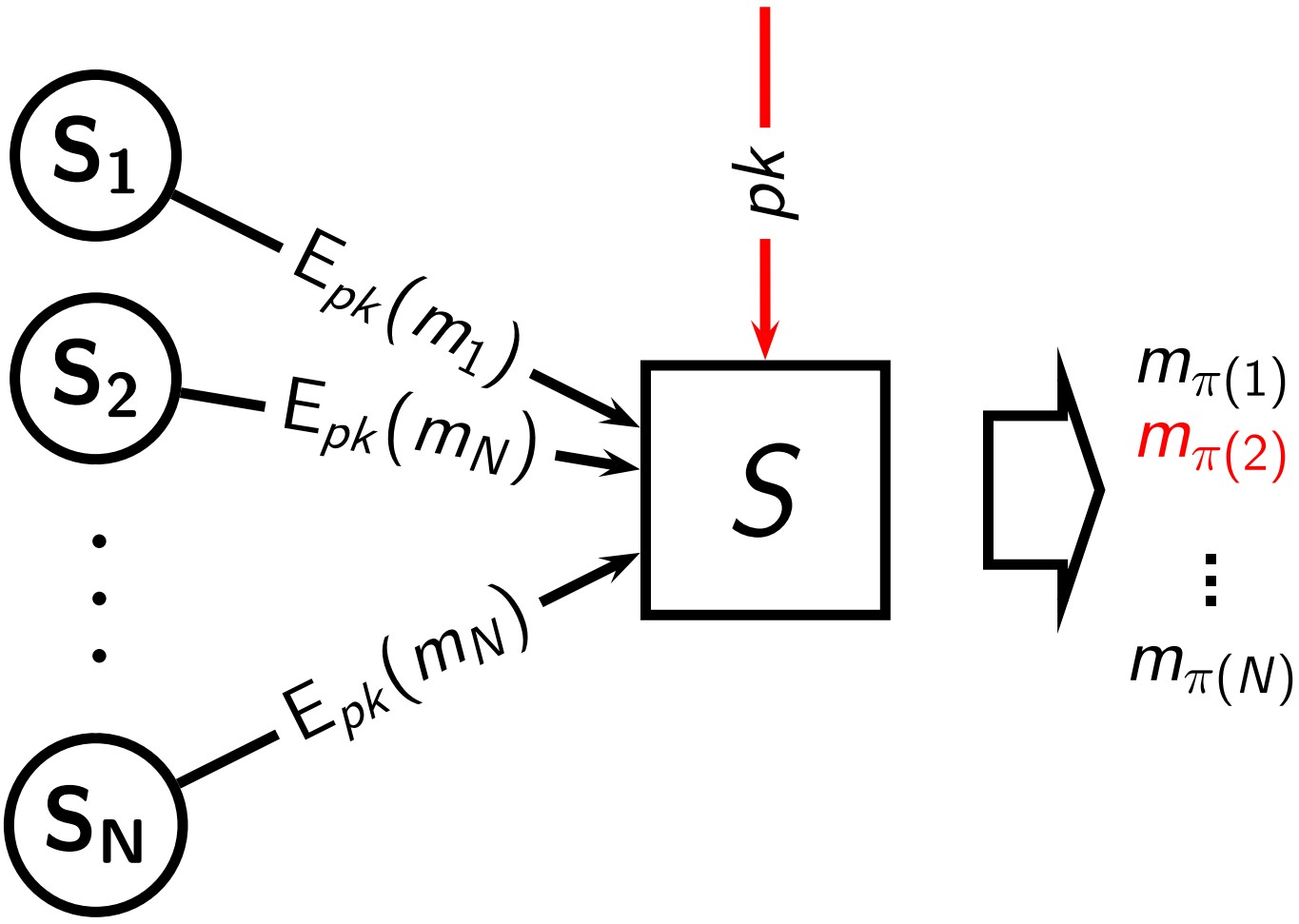
Single Trusted Mix-Server (3/3)



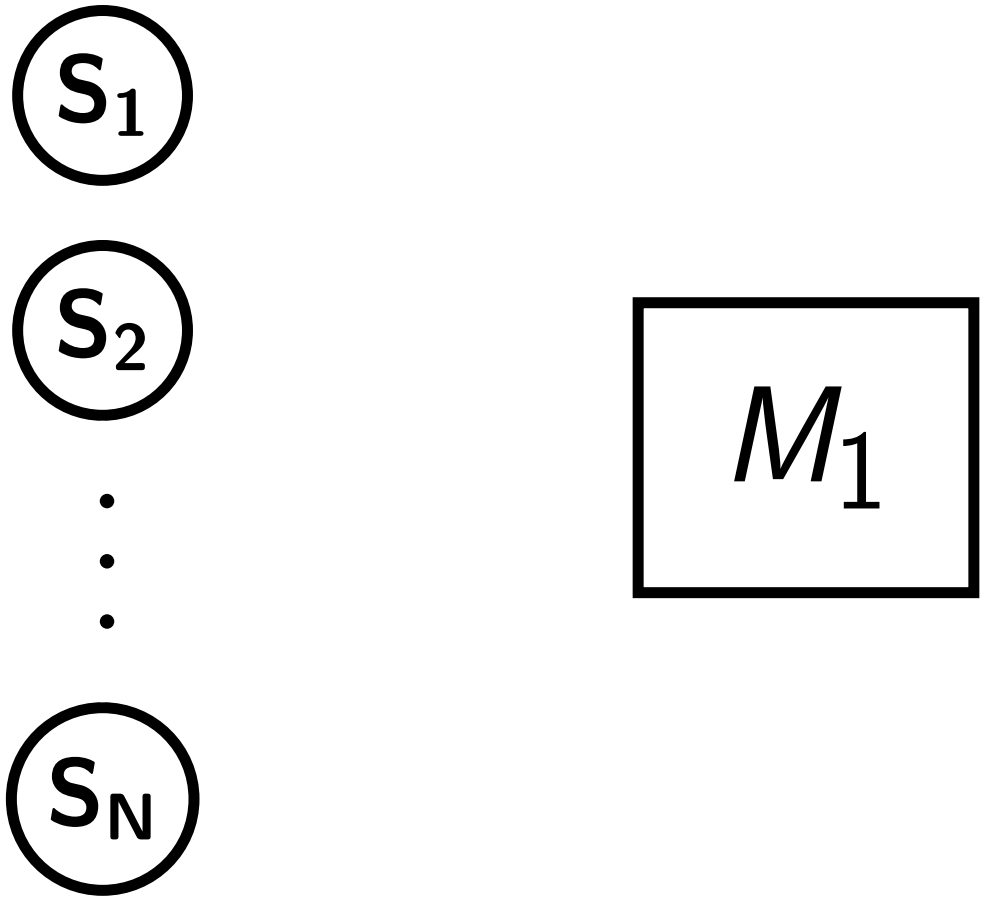
Single Trusted Mix-Server (3/3)



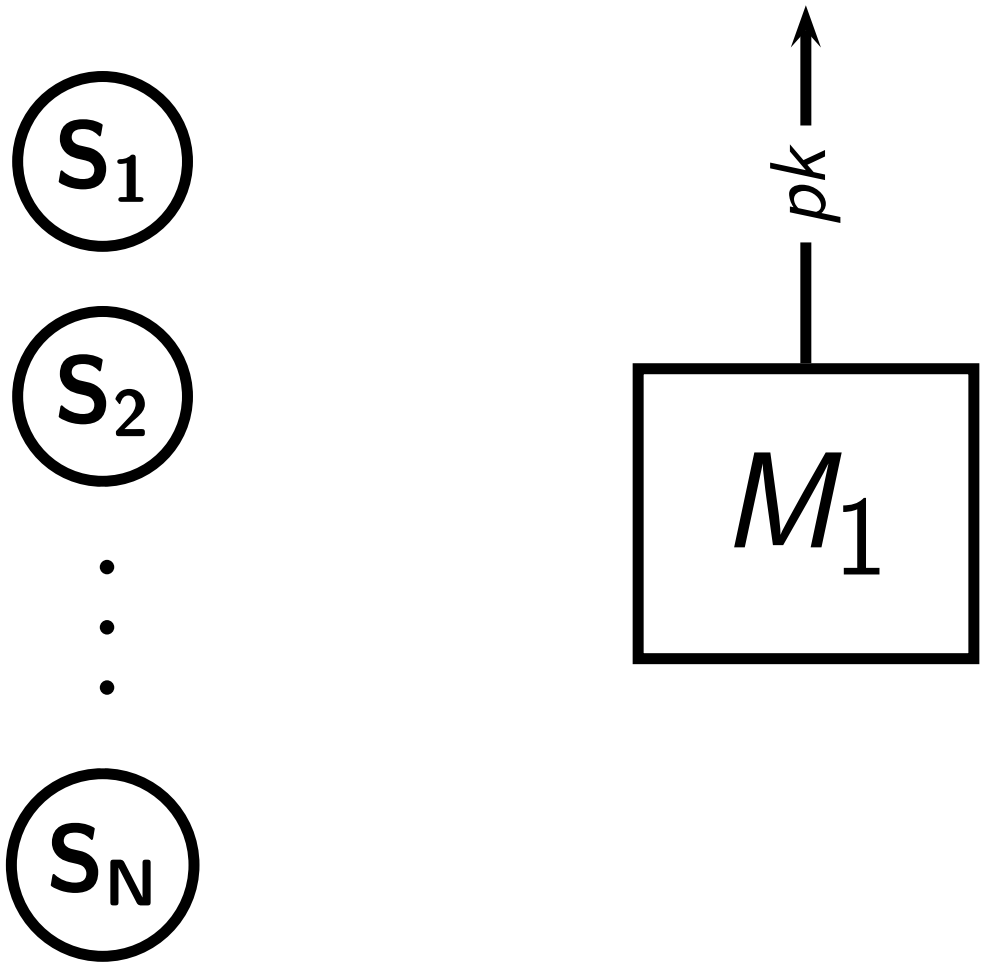
Single Trusted Mix-Server (3/3)



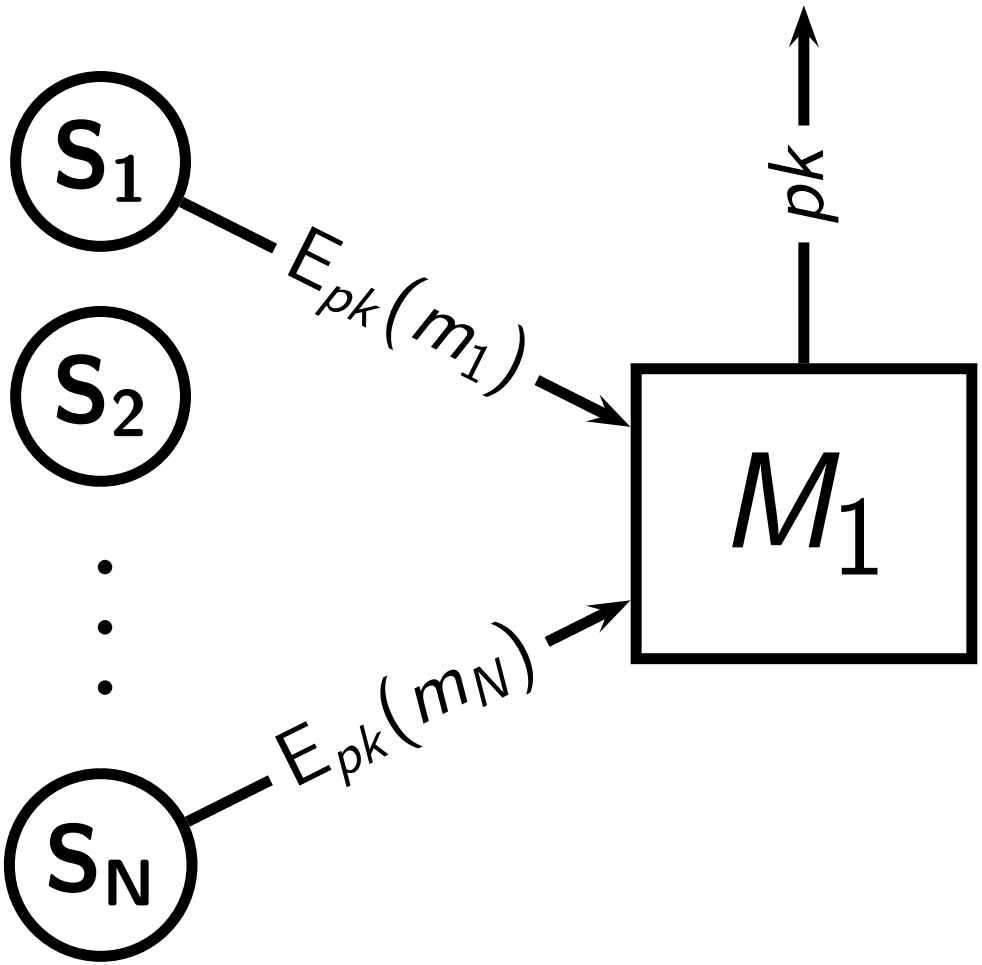
Necessity of Non-Malleability (Pfitzmann's Attack)



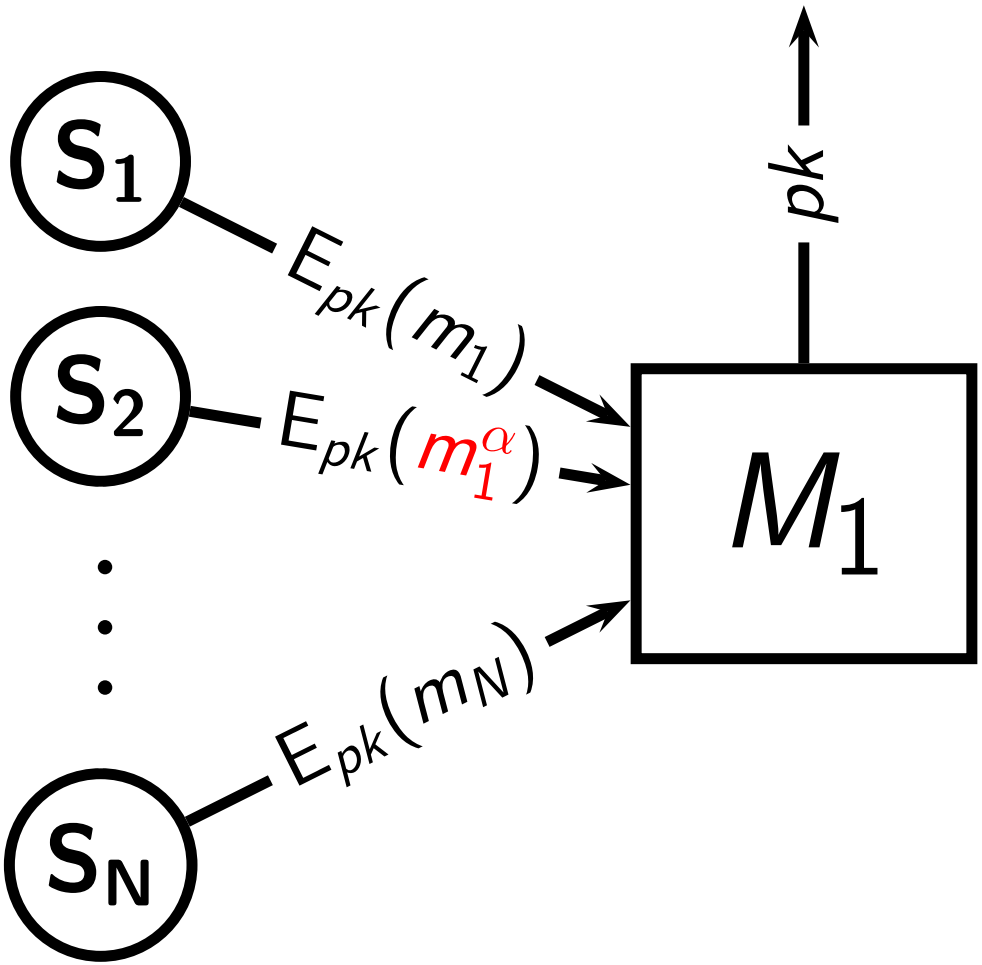
Necessity of Non-Malleability (Pfitzmann's Attack)



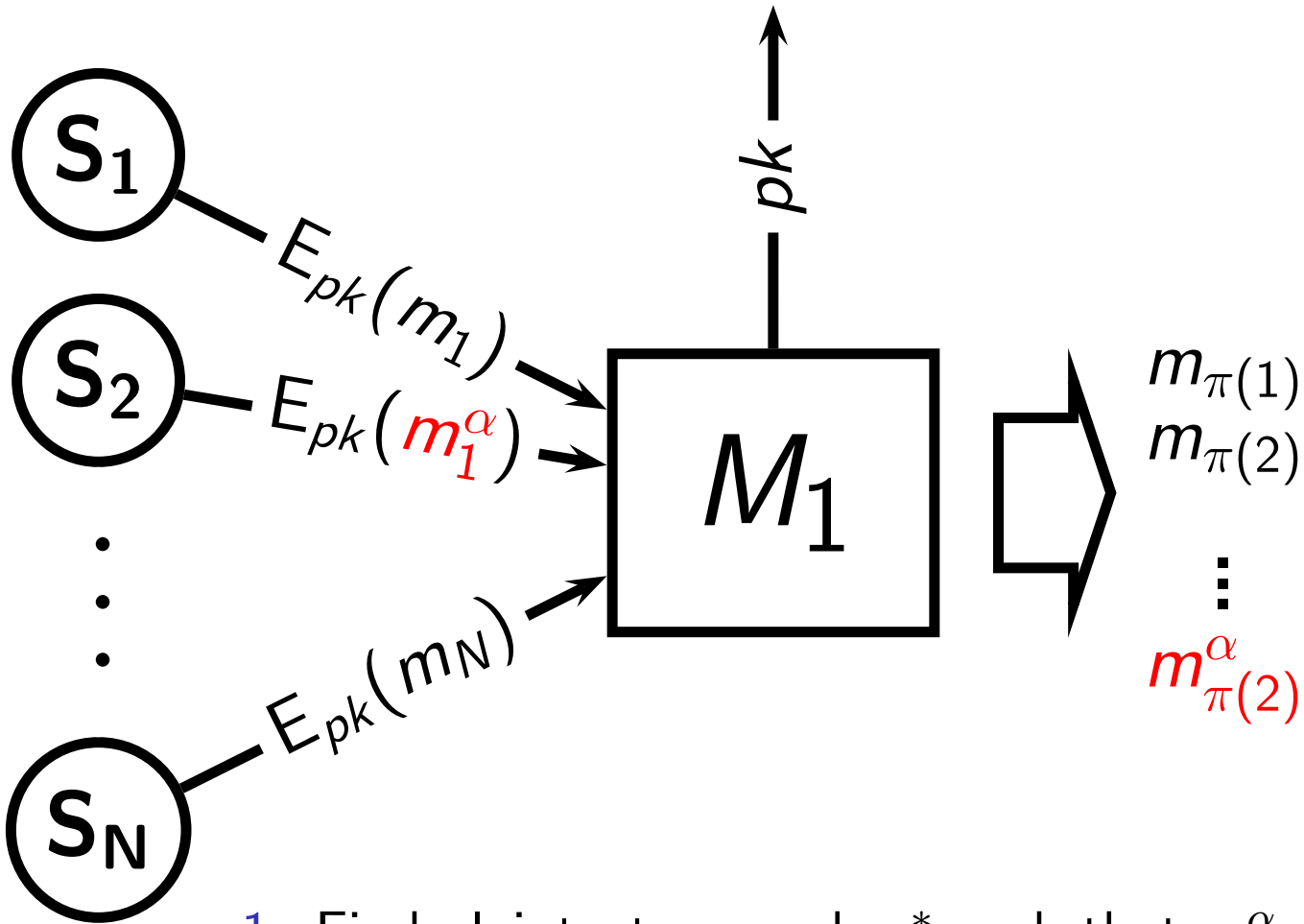
Necessity of Non-Malleability (Pfitzmann's Attack)



Necessity of Non-Malleability (Pfitzmann's Attack)



Necessity of Non-Malleability (Pfitzmann's Attack)



1. Find plaintexts m and m^* such that $m^\alpha = m^*$.
2. Very likely that $m_1 = m$!

Chaum's Question

Can we really trust a single mix-server?

Chaum's Question

Can we really trust a single mix-server?

- Use several mix-servers $M_1, \ldots, M_k$ to share trust.
- Communication takes place over over a public bulletin board.

Chaum's Mix-Net (1/2)

1. Each mix-server M_j publishes a public key pk_j.

Chaum's Mix-Net (1/2)

1. Each mix-server M_j publishes a public key pk_j.
2. S_i encrypts its message m_i as:

$$\mathrm{E}_{pk_1}(\mathrm{E}_{pk_2}(\cdots \mathrm{E}_{pk_k}(m_i)\cdots)) \ .$$

Chaum's Mix-Net (1/2)

1. Each mix-server M_j publishes a public key pk_j.
2. S_i encrypts its message m_i as:

$$\mathrm{E}_{pk_1}(\mathrm{E}_{pk_2}(\cdots \mathrm{E}_{pk_k}(m_i)\cdots)) \ .$$

3. For $j = 1, \ldots, k$:
   - 3.1 M_j decrypts each ciphertext of the output of M_{j-1} (M_0 decrypts the original ciphertexts).
   - 3.2 Outputs the resulting ciphertexts in random order.

Chaum's Mix-Net (1/2)

1. Each mix-server M_j publishes a public key pk_j.
2. S_i encrypts its message m_i as:

$$\mathrm{E}_{pk_1}(\mathrm{E}_{pk_2}(\cdots \mathrm{E}_{pk_k}(m_i)\cdots)) \ .$$

3. For $j = 1, \ldots, k$:
   - 3.1 M_j decrypts each ciphertext of the output of M_{j-1} (M_0 decrypts the original ciphertexts).
   - 3.2 Outputs the resulting ciphertexts in random order.
4. The output of M_k is the list of plaintexts $m_1, \ldots, m_N$ in random order.

Chaum's Mix-Net (2/2)

Theorem

If the cryptosystem is semantically secure, then Chaum's mix-net is secure against **passive** *adversaries that statically corrupts all except one mix-server.*

Proof.

Repeated application of the single trusted mix-server case. □

Chaum's Mix-Net (2/2)

Theorem

If the cryptosystem is semantically secure, then Chaum's mix-net is secure against **passive** *adversaries that statically corrupts all except one mix-server.*

Proof.

Repeated application of the single trusted mix-server case. □

What about **active** adversaries?

Drawbacks of Chaum's Mix-Net (1/3)

Corrupt Mix-Servers.

- Any mix-server can replace all ciphertexts and change the output to its liking!

Drawbacks of Chaum's Mix-Net (1/3)

Corrupt Mix-Servers.

- Any mix-server can replace all ciphertexts and change the output to its liking!

- But S_i can **verify** that its ciphertext was processed correctly and point out a cheater.
  Beautiful idea, but typically **useless in voting**!

Drawbacks of Chaum's Mix-Net (1/3)

Corrupt Mix-Servers.

- Any mix-server can replace all ciphertexts and change the output to its liking!
- But S_i can **verify** that its ciphertext was processed correctly and point out a cheater.
  Beautiful idea, but typically **useless in voting**!
- We must enforce correctness using additional techniques!

Drawbacks of Chaum's Mix-Net (2/3)

Corrupt Mix-Servers.

- Even with proofs of correct behavior a mix-server could prevent any output by halting!

Drawbacks of Chaum's Mix-Net (2/3)

Corrupt Mix-Servers.

- Even with proofs of correct behavior a mix-server could prevent any output by halting!
- We must let the keys of the mix-servers verifiably secret shared and enforce usage of these keys to get robustness.

Drawbacks of Chaum's Mix-Net (2/3)

Corrupt Mix-Servers.

- Even with proofs of correct behavior a mix-server could prevent any output by halting!
- We must let the keys of the mix-servers verifiably secret shared and enforce usage of these keys to get robustness.
- We must accept a tradeoff between privacy and robustness.

Drawbacks of Chaum's Mix-Net (3/3)

Corrupt Senders.

- Our simulator must be able to **extract** the submitted **plaintexts** of corrupted senders **without the secret key**.

Drawbacks of Chaum's Mix-Net (3/3)

Corrupt Senders.

- Our simulator must be able to **extract** the submitted **plaintexts** of corrupted senders **without the secret key**.
- Letting servers verifiably secret share their secret keys does not help.

Drawbacks of Chaum's Mix-Net (3/3)

Corrupt Senders.

- Our simulator must be able to **extract** the submitted **plaintexts** of corrupted senders **without the secret key**.
- Letting servers verifiably secret share their secret keys does not help.
- Repeated encryption makes it hard to construct an efficient proof of knowledge of message and gives large ciphertexts.

Drawbacks of Chaum's Mix-Net (3/3)

Corrupt Senders.

- Our simulator must be able to **extract** the submitted **plaintexts** of corrupted senders **without the secret key**.
- Letting servers verifiably secret share their secret keys does not help.
- Repeated encryption makes it hard to construct an efficient proof of knowledge of message and gives large ciphertexts.
- Repeated encryption makes it hard to jointly generate ciphertexts with known, but permuted, plaintexts as needed in, e.g., Prêt a Votér.

Homomorphic Mix-Nets

Homomorphic Cryptosystem

Messages, ciphertexts, and randomness form groups:

- M_{pk} denotes message space.
- C_{pk} denotes ciphertext space.
- R_{pk} denotes randomness space.

and encryption satisfies:

$$\mathsf{E}_{pk}(m_0, r_0) \times \mathsf{E}_{pk}(m_1, r_1) = \mathsf{E}_{pk}(m_0 \cdot m_1, r_0 + r_1)$$

Examples: El Gamal, Paillier, ...

Re-encryption

- A ciphertext $u = \mathsf{E}_{pk}(m, r)$ can be **re-encrypted** using fresh randomness $r' \in \mathsf{R}_{pk}$ using only the public key:

$$u' = \mathsf{E}_{pk}(1, r') \times u = \mathsf{E}_{pk}(m, r' + r)$$

Re-encryption

- A ciphertext $u = \mathsf{E}_{pk}(m, r)$ can be **re-encrypted** using fresh randomness $r' \in \mathsf{R}_{pk}$ using only the public key:

$$u' = \mathsf{E}_{pk}(1, r') \times u = \mathsf{E}_{pk}(m, r' + r)$$

- The group structure of R_{pk} implies that
  - $r' + r$ and r are identically distributed, so
  - u and u' are identically distributed.

Distributed Key-Generation and Decryption

Distributed Key Generation

- A random public key with associated verifiably secretly shared secret key can be jointly generated.
- The simulator takes a public key as input and simulates the protocol to output it.

Distributed Decryption

- A ciphertext can be jointly decrypted using the verifiably shared secret key, without recovering it.
- The simulator of the protocol takes public key, ciphertext, and plaintext as input.

Sako and Kilian's Mix-Net

1. The mix-servers jointly generate a public key pk.
2. S_i encrypts its message m_i as $\mathrm{E}_{pk}(m_i)$.
3. For $j = 1, \ldots, k$:
   - M_j re-encrypts each ciphertext output by the previous mix-server and permutes the resulting ciphertexts.
   - Proves that it did this correctly.
4. The mix-servers jointly decrypt L_k and output the resulting plaintexts.

Sako and Kilian's Mix-Net

1. The mix-servers jointly generate a public key pk.
2. S_i encrypts its message m_i as $u_{0,i} = \mathsf{E}_{pk}(m_i)$, makes m_i extractable, and $L_0 = (u_{0,1}, \ldots, u_{0,N})$ is the list of ciphertexts.
3. For $j = 1, \ldots, k$:
   - M_j re-encrypts each ciphertext output by the previous mix-server and permutes the resulting ciphertexts.
   - Proves that it did this correctly.
4. The mix-servers jointly decrypt L_k and output the resulting plaintexts.

Sako and Kilian's Mix-Net

1. The mix-servers jointly generate a public key pk.
2. S_i encrypts its message m_i as $u_{0,i} = \mathsf{E}_{pk}(m_i)$, makes m_i extractable, and $L_0 = (u_{0,1}, \ldots, u_{0,N})$ is the list of ciphertexts.
3. For $j = 1, \ldots, k$:
   - M_j chooses $r_{j,i} \in \mathsf{R}_{pk}$ and π_j randomly and defines

     $$L_j = (u_{j,i}) \quad \text{where} \quad u_{j,i} = \mathsf{E}_{pk}(1, r_{j,\pi_j(i)}) \times u_{j-1,\pi_j(i)}$$

   - M_j gives zero-knowledge proof of knowledge of $r_{j,i}$ and π_j.
4. The mix-servers jointly decrypt L_k and output the resulting plaintexts.

Sako and Kilian's Mix-Net

1. The mix-servers jointly generate a public key pk.

2. S_i encrypts its message m_i as $u_{0,i} = \mathsf{E}_{pk}(m_i)$, makes m_i extractable, and $L_0 = (u_{0,1}, \ldots, u_{0,N})$ is the list of ciphertexts.

3. For $j = 1, \ldots, k$:

   - M_j chooses $r_{j,i} \in \mathsf{R}_{pk}$ and π_j randomly and defines

     $$L_j = (u_{j,i}) \quad \text{where} \quad u_{j,i} = \mathsf{E}_{pk}(1, r_{j,\pi_j(i)}) \times u_{j-1,\pi_j(i)}$$

   - M_j gives zero-knowledge proof of knowledge of $r_{j,i}$ and π_j.

4. The mix-servers jointly decrypt L_k and output the resulting plaintexts.

A Remark On Proofs of Shuffles

- Andy, Jun, Kazue, Jens, and me are discussing which proof of a shuffle to use. Needless to say, we disagree...

A Remark On Proofs of Shuffles

- Andy, Jun, Kazue, Jens, and me are discussing which proof of a shuffle to use. Needless to say, we disagree...
- Why should we be forced to use the same protocol, when any proof of a shuffle suffices?

A Remark On Proofs of Shuffles

- Andy, Jun, Kazue, Jens, and me are discussing which proof of a shuffle to use. Needless to say, we disagree...
- Why should we be forced to use the same protocol, when any proof of a shuffle suffices?
- Fiat-Shamir heuristic is ok in practice, right?

A Remark On Proofs of Shuffles

- Andy, Jun, Kazue, Jens, and me are discussing which proof of a shuffle to use. Needless to say, we disagree...
- Why should we be forced to use the same protocol, when any proof of a shuffle suffices?
- Fiat-Shamir heuristic is ok in practice, right?
- Can we allow any zero-knowledge proof of correct re-encryption in the random oracle model?

A Remark On Proofs of **KNOWLEDGE** of Shuffles

- Andy, Jun, Kazue, Jens, and me are discussing which proof of a shuffle to use. Needless to say, we disagree...
- Why should we be forced to use the same protocol, when any proof of a shuffle suffices?
- Fiat-Shamir heuristic is ok in practice, right?
- Can we allow any zero-knowledge proof of correct re-encryption in the random oracle model?

Sako and Kilian's Mix-Net (contd.)

Theorem

If the cryptosystem is semantically secure, then Sako and Kilian's mix-net is secure against **active** *adversaries* **statically** *corrupting* **any minority** *of the mix-servers.*

Proof.

Changes to the simulator compared to the one in the proof of security of the *Single Trusted Mix-Server*:

Sako and Kilian's Mix-Net (contd.)

Theorem

If the cryptosystem is semantically secure, then Sako and Kilian's mix-net is secure against **active** *adversaries* **statically** *corrupting* **any minority** *of the mix-servers.*

Proof.

Changes to the simulator compared to the one in the proof of security of the *Single Trusted Mix-Server*:

- Extraction of plaintexts of corrupt senders is more involved.

Sako and Kilian's Mix-Net (contd.)

Theorem

If the cryptosystem is semantically secure, then Sako and Kilian's mix-net is secure against **active** *adversaries* **statically** *corrupting* **any minority** *of the mix-servers.*

Proof.

Changes to the simulator compared to the one in the proof of security of the *Single Trusted Mix-Server*:

- Extraction of plaintexts of corrupt senders is more involved.
- Proofs of shuffles of uncorrupted mix-servers are simulated.

Sako and Kilian's Mix-Net (contd.)

Theorem

If the cryptosystem is semantically secure, then Sako and Kilian's mix-net is secure against **active** *adversaries* **statically** *corrupting* **any minority** *of the mix-servers.*

Proof.

Changes to the simulator compared to the one in the proof of security of the *Single Trusted Mix-Server*:

- Extraction of plaintexts of corrupt senders is more involved.
- Proofs of shuffles of uncorrupted mix-servers are simulated.
- Permutations used by corrupt mix-servers are extracted.

Sako and Kilian's Mix-Net (contd.)

Theorem

If the cryptosystem is semantically secure, then Sako and Kilian's mix-net is secure against **active** *adversaries* **statically** *corrupting* **any minority** *of the mix-servers.*

Proof.

Changes to the simulator compared to the one in the proof of security of the *Single Trusted Mix-Server*:

- Extraction of plaintexts of corrupt senders is more involved.
- Proofs of shuffles of uncorrupted mix-servers are simulated.
- Permutations used by corrupt mix-servers are extracted.
- This allows correct simulation of decryption of messages sent by corrupted senders.

Proofs of Shuffles

Batch-Proof of Re-encryption

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that

$$u'_i = \mathsf{E}_{pk}(1, r_i) \times u_i$$

Batch-Proof of Re-encryption

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_i) \times u_i$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.

Batch-Proof of Re-encryption

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_i) \times u_i$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.

2. P and V define

$$u = \prod_i u_i^{e_i}$$

Batch-Proof of Re-encryption

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_i) \times u_i$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.
2. P and V define
$$u = \prod_i u_i^{e_i}$$
3. P and V define
$$u' = \prod_i (u'_i)^{e_i}$$

Batch-Proof of Re-encryption

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_i) \times u_i$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.
2. P and V define
$$u = \prod_i u_i^{e_i}$$
3. P and V define
$$u' = \prod_i (u'_i)^{e_i}$$
4. P proves knowledge of t such that
$$u' = \mathsf{E}_{pk}(1, t) \times u$$

Batch-Proof of Re-encryption

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_i) \times u_i$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.
2. P and V define
$$u = \prod_i u_i^{e_i}$$
3. P and V define
$$u' = \prod_i (u'_i)^{e_i} = \prod_i \mathsf{E}_{pk}(1, r_i)^{e_i} \prod_i u_i^{e_i}$$
4. P proves knowledge of $t = \sum_i e_i r_i$ such that
$$u' = \mathsf{E}_{pk}(1, t) \times u$$

Analysis

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathsf{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u_i')^{e_{j,i}}$$

Analysis

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u_i')^{e_{j,i}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

Analysis

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathsf{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u_i')^{e_{j,i}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$= \prod_j \left(\mathsf{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} \right)^{a_{l,j}}$$

Analysis

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathsf{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u_i')^{e_{j,i}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$\mathsf{E}_{pk}\left(1, \sum\nolimits_j t_j a_{l,j}\right) u_l = \prod\nolimits_j \left(\mathsf{E}_{pk}(1, t_j) \prod\nolimits_i u_i^{e_{j,i}}\right)^{a_{l,j}}$$

Analysis

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathsf{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u_i')^{e_{j,i}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$\begin{aligned} \mathsf{E}_{pk}\left(1, \sum\nolimits_j t_j a_{l,j}\right) u_l &= \prod\nolimits_j \left(\mathsf{E}_{pk}(1, t_j) \prod\nolimits_i u_i^{e_{j,i}}\right)^{a_{l,j}} \\ &= \prod\nolimits_j \left(\prod\nolimits_i (u_i')^{e_{j,i}}\right)^{a_{l,j}} \end{aligned}$$

Analysis

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathsf{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u_i')^{e_{j,i}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$\begin{aligned}
\mathsf{E}_{pk}\left(1, \sum\nolimits_j t_j a_{l,j}\right) u_l &= \prod\nolimits_j \left(\mathsf{E}_{pk}(1, t_j) \prod\nolimits_i u_i^{e_{j,i}}\right)^{a_{l,j}} \\
&= \prod\nolimits_j \left(\prod\nolimits_i (u_i')^{e_{j,i}}\right)^{a_{l,j}} \\
&= \prod\nolimits_i (u_i')^{\sum_j a_{l,j} e_{j,i}} \\
&= u_l'
\end{aligned}$$

Batch-Proof of Re-encryption Under Permutation [N01,FS01]

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that

$$u'_i = \mathsf{E}_{pk}(1, r_{\pi(i)}) \times u_{\pi(i)}$$

Batch-Proof of Re-encryption Under Permutation [N01,FS01]

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_{\pi(i)}) \times u_{\pi(i)}$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.

Batch-Proof of Re-encryption Under Permutation [N01,FS01]

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_{\pi(i)}) \times u_{\pi(i)}$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.

2. P and V define

$$u = \prod_i u_i^{e_i}$$

Batch-Proof of Re-encryption Under Permutation [N01,FS01]

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_{\pi(i)}) \times u_{\pi(i)}$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.
2. P and V define
$$u = \prod_i u_i^{e_i}$$
3. P computes (since V can't)
$$u' = \prod_i (u'_i)^{e_{\pi(i)}}$$

Batch-Proof of Re-encryption Under Permutation [N01,FS01]

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_{\pi(i)}) \times u_{\pi(i)}$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.
2. P and V define
$$u = \prod_i u_i^{e_i}$$
3. P computes (since V can't)
$$u' = \prod_i (u'_i)^{e_{\pi(i)}}$$
4. P proves knowledge of correct form of u' and knowledge of t such that
$$u' = \mathsf{E}_{pk}(1, t) \times u$$

Batch-Proof of Re-encryption Under Permutation [N01,FS01]

Goal. Given pk, $(u_1, \ldots, u_N)$ and $(u'_1, \ldots, u'_N)$, prove knowledge of r_i such that $u'_i = \mathsf{E}_{pk}(1, r_{\pi(i)}) \times u_{\pi(i)}$.

Protocol.

1. V chooses $e_1, \ldots, e_N$ randomly.
2. P and V define
$$u = \prod_i u_i^{e_i}$$
3. P computes (since V can't)
$$u' = \prod_i (u'_i)^{e_{\pi(i)}} = \prod_i \mathsf{E}_{pk}(1, r_{\pi(i)})^{e_{\pi(i)}} \prod_i u_i^{e_{\pi(i)}}$$
4. P proves knowledge of correct form of u' and knowledge of $t = \sum_i e_{\pi(i)} r_{\pi(i)}$ such that
$$u' = \mathsf{E}_{pk}(1, t) \times u$$

Analysis With Permutation

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u'_i)^{e_{j,\pi(i)}}$$

Analysis With Permutation

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u'_i)^{e_{j,\pi(i)}}$$

- Find a_{lj} such that $\sum_j a_{lj}\overline{e}_j$ is lth unit vector.

Analysis With Permutation

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u'_i)^{e_{j,\pi(i)}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$= \prod_j \left(\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} \right)^{a_{l,j}}$$

Analysis With Permutation

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u'_i)^{e_{j,\pi(i)}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$\mathrm{E}_{pk}\left(1, \sum_j t_j a_{l,j}\right) u_l = \prod_j \left(\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}}\right)^{a_{l,j}}$$

Analysis With Permutation

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u'_i)^{e_{j,\pi(i)}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$\begin{aligned} \mathrm{E}_{pk}\left(1, \sum\nolimits_j t_j a_{l,j}\right) u_l &= \prod\nolimits_j \left(\mathrm{E}_{pk}(1, t_j) \prod\nolimits_i u_i^{e_{j,i}}\right)^{a_{l,j}} \\ &= \prod\nolimits_j \left(\prod\nolimits_i (u'_i)^{e_{j,\pi(i)}}\right)^{a_{l,j}} \end{aligned}$$

Analysis With Permutation

- Extract N linearly independent vectors $\overline{e}_j = (e_{j,1}, \ldots, e_{j,N})$ and t_j such that:

$$\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}} = \prod_i (u'_i)^{e_{j,\pi(i)}}$$

- Find a_{lj} such that $\sum_j a_{lj} \overline{e}_j$ is lth unit vector.

- Conclude that

$$\begin{aligned}
\mathrm{E}_{pk}\left(1, \sum_j t_j a_{l,j}\right) u_l &= \prod_j \left(\mathrm{E}_{pk}(1, t_j) \prod_i u_i^{e_{j,i}}\right)^{a_{l,j}} \\
&= \prod_j \left(\prod_i (u'_i)^{e_{j,\pi(i)}}\right)^{a_{l,j}} \\
&= \prod_i (u'_i)^{\sum_j a_{l,j} e_{j,\pi(i)}} \\
&= u_{\pi^{-1}(l)}{}'
\end{aligned}$$

What Remains?

Great, batch-proofs almost work under permutation, but how do we prove correct form of $u' = \prod_i (u'_i)^{e_{\pi(i)}}$?

Schwartz-Zippel

Theorem (Schwartz-Zippel)

Let $f \in \mathbb{Z}_q[x_1, x_2, \ldots, x_N]$ be a non-zero polynomial of degree N, let $S \subset \mathbb{Z}_q$, and let $e_i \in S$ be randomly chosen. Then

$$\Pr[f(e_1, \ldots, e_N) = 0] \leq \frac{N}{|S|} \ .$$

Permutation-Matrix Test [TW10]

- Let M be an $N \times N$-matrix over $\mathbb{Z}_q$

$$M = \begin{pmatrix} m_1 \\ \vdots \\ m_N \end{pmatrix}$$

Permutation-Matrix Test [TW10]

- Let M be an $N \times N$-matrix over $\mathbb{Z}_q$

$$M = \begin{pmatrix} m_1 \\ \vdots \\ m_N \end{pmatrix}$$

- Define N-degree polynomial $f_M \in \mathbb{Z}_q[x_1, \ldots, x_N]$ by

$$f_M(x_1, \ldots, x_N) = \prod_i \langle m_i, \overline{x} \rangle = \prod_i \left(\sum_j m_{i,j} x_j \right)$$

Permutation-Matrix Test [TW10]

- Let M be an $N \times N$-matrix over $\mathbb{Z}_q$

$$M = \begin{pmatrix} m_1 \\ \vdots \\ m_N \end{pmatrix}$$

- Define N-degree polynomial $f_M \in \mathbb{Z}_q[x_1, \ldots, x_N]$ by

$$f_M(x_1, \ldots, x_N) = \prod_i \langle m_i, \overline{x} \rangle = \prod_i \left(\sum_j m_{i,j} x_j \right)$$

Theorem

M is a permutation matrix iff

$$f_M(\overline{x}) = \prod_j x_j \quad \textit{and} \quad M \cdot \overline{1} = \overline{1}$$

Committed Permutation-Matrix Test

1. P computes "homomorphic" commitment $C(M)$ of permutation matrix M.

2. V chooses $e_1, \ldots, e_N \in S$ randomly and computes

$$C(M) \cdot \overline{e} = C(M \cdot \overline{e}) = \left(C(\langle m_1, \overline{e} \rangle), \ldots, C(\langle m_N, \overline{e} \rangle)\right)$$

3. P proves (knowledge) using standard Schnorr-proofs that

$$f_M(e_1, \ldots, e_N) = \prod_j e_j \quad \text{and} \quad C(M) \cdot \overline{1} = C(\overline{1})$$

Commitment-Consistent Proof of a Shuffle

We need to prove correct form of $u' = \prod_i (u'_i)^{e_{\pi(i)}}$.

Commitment-Consistent Proof of a Shuffle

We need to prove correct form of $u' = \prod_i (u'_i)^{e_{\pi(i)}}$.

Protocol

1. P proves that $C(M)$ is a commitment to a permutation matrix M.

Commitment-Consistent Proof of a Shuffle

We need to prove correct form of $u' = \prod_i (u'_i)^{e_{\pi(i)}}$.

Protocol

1. P proves that $C(M)$ is a commitment to a permutation matrix M.

2. V chooses $e_1, \ldots, e_N \in S$ randomly.

Commitment-Consistent Proof of a Shuffle

We need to prove correct form of $u' = \prod_i (u'_i)^{e_{\pi(i)}}$.

Protocol

1. P proves that $C(M)$ is a commitment to a permutation matrix M.

2. V chooses $e_1, \ldots, e_N \in S$ randomly.

3. P proves, using Schnorr-proofs, knowledge of e'_i such that

$$C(M) \cdot \overline{e} = \left(C(e'_1), \ldots, C(e'_N)\right) \quad \text{and} \quad u' = \prod_i (u'_i)^{e'_i}$$

Commitment-Consistent Proof of a Shuffle

We need to prove correct form of $u' = \prod_i (u'_i)^{e_{\pi(i)}}$.

Protocol

1. P proves that $C(M)$ is a commitment to a permutation matrix M.

2. V chooses $e_1, \ldots, e_N \in S$ randomly.

3. P proves, using Schnorr-proofs, knowledge of e'_i such that

$$C(M) \cdot \overline{e} = \left(C(e_{\pi(1)}), \ldots, C(e_{\pi(N)})\right) \quad \text{and} \quad u' = \prod_i (u'_i)^{e_{\pi(i)}}$$

Submission Schemes

Submission Schemes

Must allow extracting the plaintexts of corrupt senders without secret key of cryptosystem. Many ways with pros and cons:

- Interactive zero-knowledge proof of knowledge.
- Non-interactive proof using Fiat-Shamir heuristic.
- Non-interactive zero-knowledge proof (of knowledge).
- Naor-Yung paradigm with "proof" of equality of plaintexts.
- Verifiable secret sharing based "proof of knowledge".
- Based on hashproofs and stripping Cramer-Shoup ciphertexts.
- By construction, e.g., Pret-a-Votér or trusted encryption machine.

Questions?