Return Code Schemes for Electronic Voting Systems

Shahram Khazaei
Douglas Wikström

Sharif University of Technology, Iran
KTH Royal Institute of Technology, Sweden

E-Vote-ID 2017

Presented by: Johannes Mueller

Overview

1. Contribution and motivation
2. Problem formulation
3. Our constructions
4. Concluding remarks

Contribution

- a unified treatment of designing return code schemes is presented
- several return code schemes are proposed
- highly flexible w.r.t. usability, security, and performance trade-offs

Main idea: generating return codes as a multiparty computation of a secure MAC tag applied on encrypted votes

Note: idea might be folklore, but has not received systematic treatment

End-to-end verifiability

- a desirable property that can be achieved by electronic voting systems
- includes three different kinds of verifications
  - cast-as-intended
  - recorded-as-cast
  - counted-as-recorded
- return codes: for ensuring cast-as-intended verifiability

Cast-as-intended

- assures a voter that his vote has been cast according to his intention
- bypasses the required trust on voting devices
- two basic approaches:
  - verify that the right choice was encrypted
    - ★ **solution I**: simply perform the encryption independently on a different device and compare the results as is done in Estonia
    - ★ **solution II**: continuous blackbox testing (called Benaloh challenge), as in Helios, Wombat, and VoteBox
  - verify that the ciphertext decrypts to the intended choice
    - ★ **solution III**: return codes, used in Norway and Switzerland

Return codes: history

- have recently received considerable amount of attention
- mainly due to their usability properties
- proposals include [PG11,AHL+11,G10,GL15,HLL10,L11,PG12]
- have been used in nation-wide elections in Norway and Switzerland
- idea stems from *Chaum's code voting (2001)*

Return codes: high level description

1. each candidate choice is associated with a random code
2. each voter receives a printed voting card
3. printed voting card includes all (candidate choice,random code) pairs
4. printed voting cards are voter specific
5. voter submits an encryption of his candidate choice
6. random code is returned back to the voter
7. voter accepts if he receives the expected *return code*

Return codes: high level description

1. each candidate choice is associated with a random code
2. each voter receives a printed voting card
3. printed voting card includes all (candidate choice,random code) pairs
4. printed voting cards are voter specific
5. voter submits an encryption of his candidate choice
6. random code is returned back to the voter
7. voter accepts if he receives the expected *return code*

Drawback: does not guarantee voter's privacy
Alternative: use Chaum's code voting

Return codes: technical requirements

- individual codes must reveal nothing about the voting choice
- the random codes of different voters are chosen independently
- no need for the codes to uniquely identify the choices
- a random-looking keyed map from set of choices to set of codes works fine

Conclusion: use a MAC with a guaranteed privacy property

Return codes: a two phase multiparty computation

Offline phase: return codes are computed by a printer but the servers hold the secret key

Online phase: voter computes the encrypted vote and servers compute the MAC without decryption

Offline phase (secure printing?!)

A simple solution is to share the MAC keys with the printer.

Probably used in practice by voting companies!

A safer solution is given in next slide ...

Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption

Servers with shared inputs sk, k_i	Printer with input pk
for each voting option m:	

Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption

Servers with shared inputs sk, k_i | Printer with input pk

for each voting option m:

$$\underbrace{ERC = \mathrm{Enc}_{pk}(\mathrm{Mac}_{k_i}(m))}_{\text{without leaking extra information}}$$

Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption

Servers with shared inputs sk, k_i | Printer with input pk

for each voting option m:

$\underbrace{ERC = \mathrm{Enc}_{pk}(\mathrm{Mac}_{k_i}(m))}_{\text{without leaking extra information}} \xrightarrow{\quad ERC \quad}$

Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption

Servers with shared inputs sk, k_i | Printer with input pk

for each voting option m:

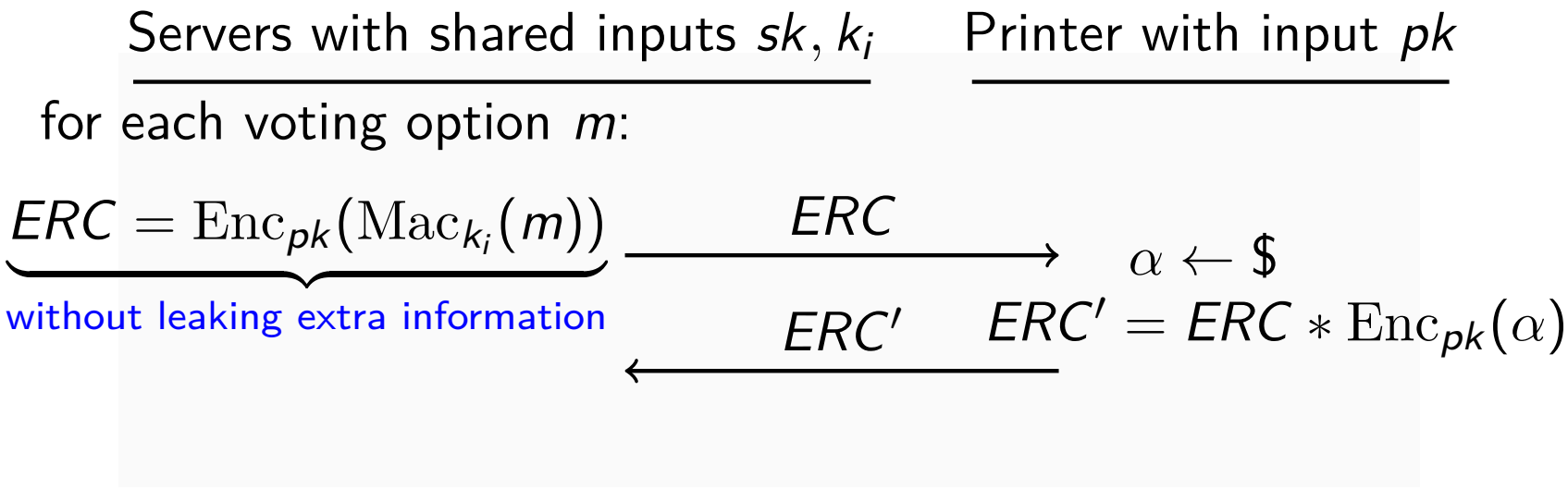
$\underbrace{ERC = \mathrm{Enc}_{pk}(\mathrm{Mac}_{k_i}(m))}_{\text{without leaking extra information}} \xrightarrow{ERC}$

$\alpha \leftarrow \$$

$ERC' = ERC * \mathrm{Enc}_{pk}(\alpha)$

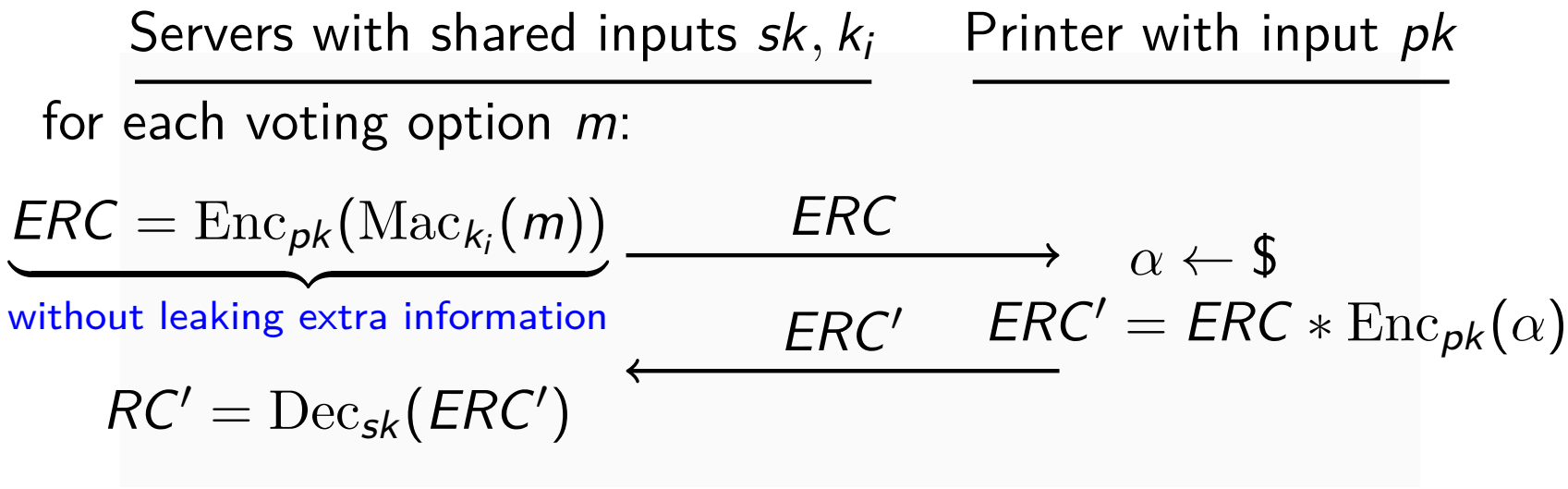
Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption



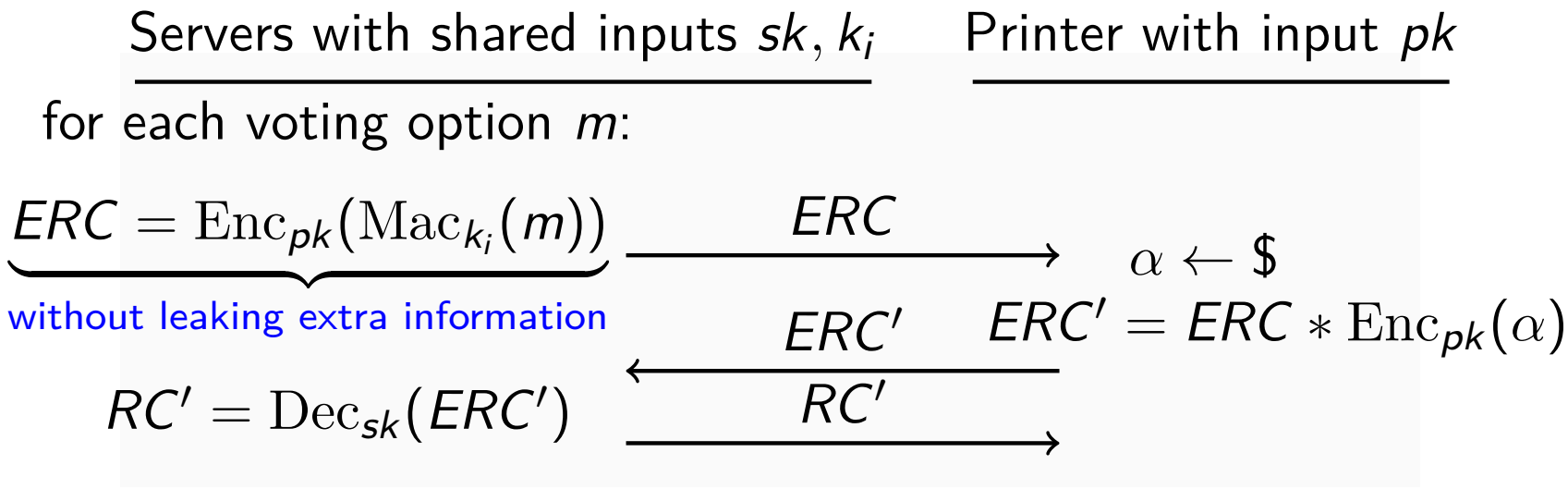
Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption



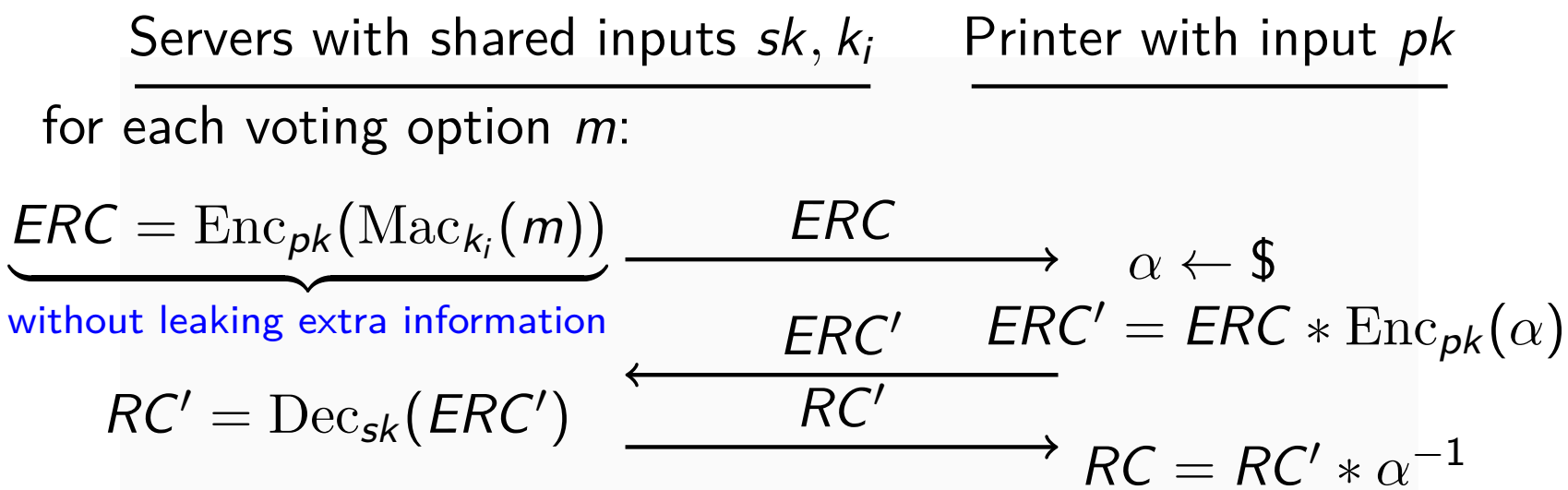
Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption



Offline phase (secure printing)

k_i: a (possibly voter-dependent) MAC key shared among servers
RC, ERC: the return code and its encryption
α: a random mask
RC', ERC': masked return code and its encryption



Online phase (distributed MAC evaluation)

c: encrypted vote
k_i: a (possibly voter-dependent) shared MAC key shared among servers
RC: the return code

Servers with shared inputs sk, k_i	Voter i with input m

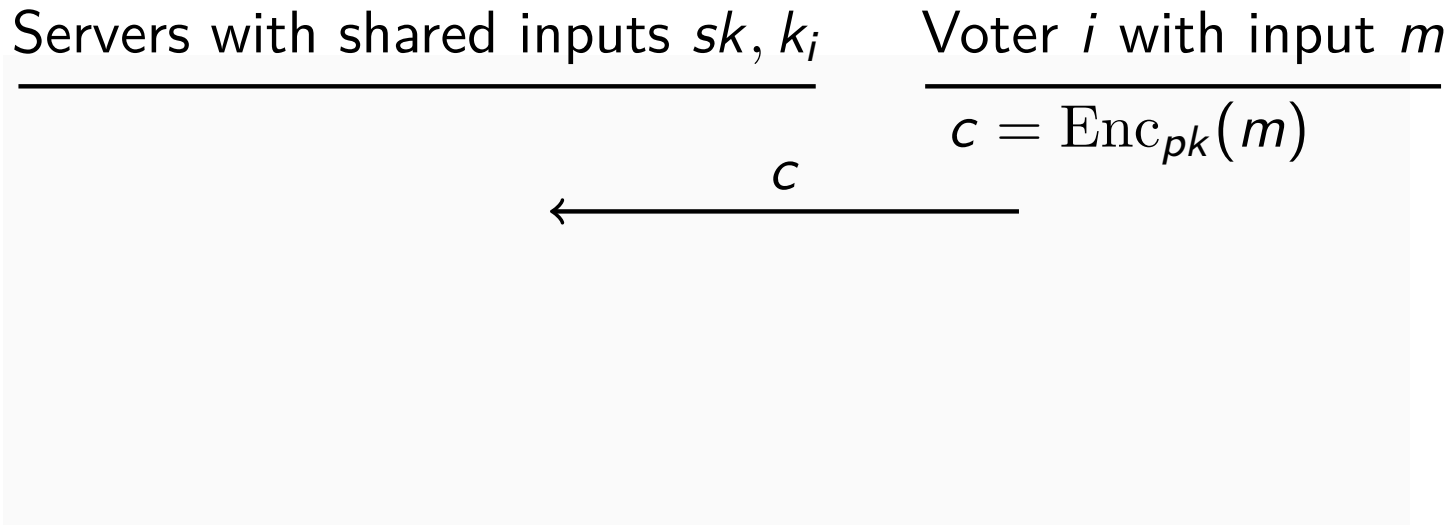
Online phase (distributed MAC evaluation)

c: encrypted vote
k_i: a (possibly voter-dependent) shared MAC key shared among servers
RC: the return code

Servers with shared inputs sk, k_i	Voter i with input m
	$c = \mathrm{Enc}_{pk}(m)$

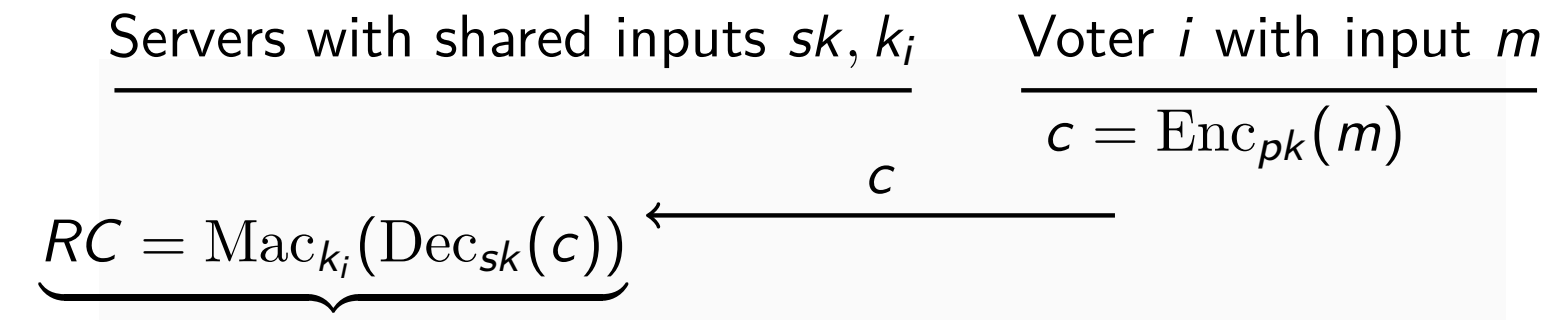
Online phase (distributed MAC evaluation)

c: encrypted vote
k_i: a (possibly voter-dependent) shared MAC key shared among servers
RC: the return code



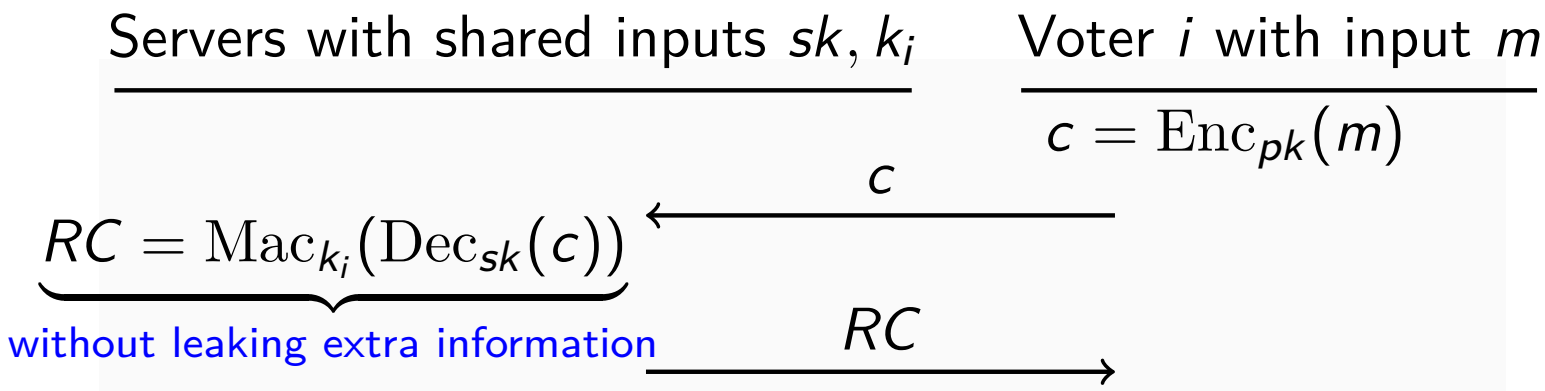
Online phase (distributed MAC evaluation)

c: encrypted vote
k_i: a (possibly voter-dependent) shared MAC key shared among servers
RC: the return code



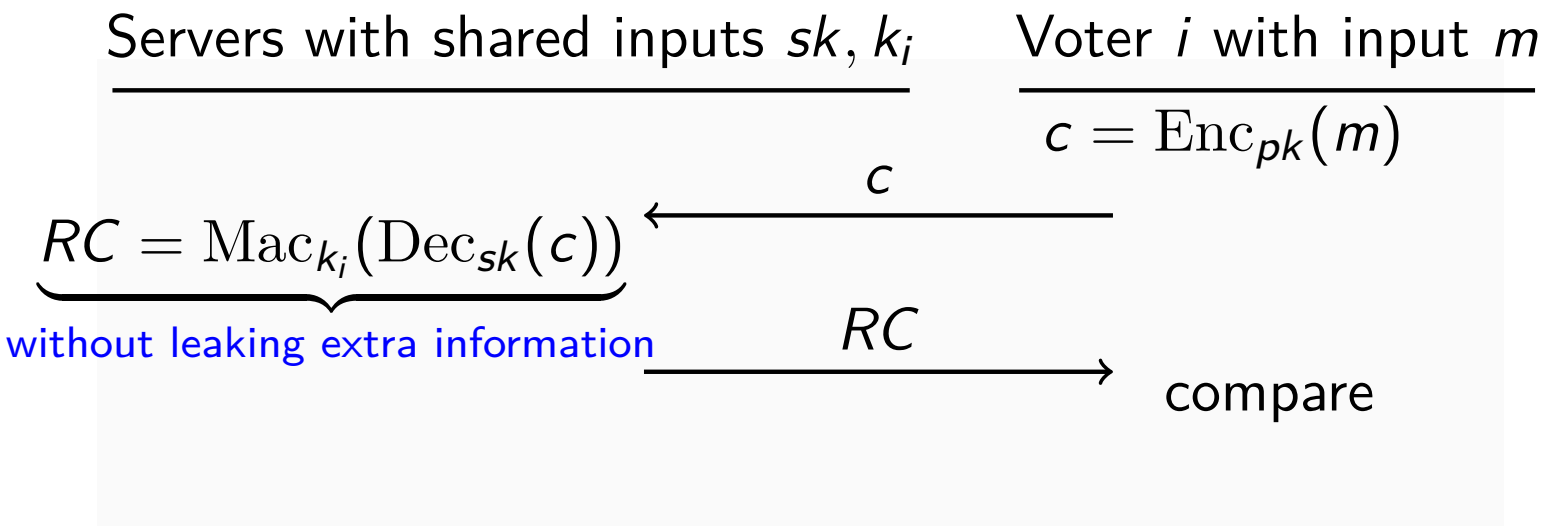
Online phase (distributed MAC evaluation)

c: encrypted vote
k_i: a (possibly voter-dependent) shared MAC key shared among servers
RC: the return code



Online phase (distributed MAC evaluation)

c: encrypted vote
k_i: a (possibly voter-dependent) shared MAC key shared among servers
RC: the return code



Scheme I: Universal Hash Functions Used as MACs

Main idea: use standard one-time MAC $f_{a,b}(x) = ax + b$ in the exponent

Given a ciphertext $\mathrm{Enc}_{pk}(g^x)$**:**

- distributedly compute the encrypted return code

$$ERC = \mathrm{Enc}_{pk}(g^x)^{a_i}\mathrm{Enc}_{pk}(g^{b_i}) = \mathrm{Enc}_{pk}(g^{f_{a_i,b_i}(x)})$$

- distributedly decrypt ERC to compute $RC = g^{f_{a_i,b_i}(x)}$

Scheme I: Universal Hash Functions Used as MACs

Main idea: use standard one-time MAC $f_{a,b}(x) = ax + b$ in the exponent

Given a ciphertext $\mathrm{Enc}_{pk}(g^x)$:

- distributedly compute the encrypted return code

$$ERC = \mathrm{Enc}_{pk}(g^x)^{a_i}\mathrm{Enc}_{pk}(g^{b_i}) = \mathrm{Enc}_{pk}(g^{f_{a_i,b_i}(x)})$$

- distributedly decrypt ERC to compute $RC = g^{f_{a_i,b_i}(x)}$

Remarks:

- MAC key $k_i = (a_i, b_i)$ is user-dependent
- a_i: secret shared among the servers
- no need to secret share b_i: define $\mathrm{Enc}_{pk}(g^{b_i})$ as output of a RO
- use known efficient protocols for "distributed exponentiation"
- for efficiency, truncate RC

Scheme II: One-time Pad and Random Choice Representatives

Main idea: the insecure scheme $\mathrm{Mac}_k(m) = k \cdot m$ is a one-time secure MAC for a random choice of plaintext unknown to the adversary

Scheme II: One-time Pad and Random Choice Representatives

Main idea: the insecure scheme $\mathrm{Mac}_k(m) = k \cdot m$ is a one-time secure MAC for a random choice of plaintext unknown to the adversary

Details:

- MAC key k_i is user-dependent
- $\mathrm{Enc}_{pk}(k_i)$ is generated by a RO, i.e., k_i is known in encrypted form
- assign unique representative m_{ij} to jth choice of the ith voter
- $\mathrm{Enc}_{pk}(m_{ij})$ is generated by a RO, i.e., m_{ij} is known in encrypted form
- the corresponding return code is $RC = \mathrm{Mac}_{k_i}(m_{ij}) = k_i \cdot m_{ij}$
- voter i sends re-encryption of one of its designated ciphertexts + ZKP
- servers jointly decrypt the received ciphertext multiplied by $\mathrm{Enc}_{pk}(k_i)$

Scheme II: One-time Pad and Random Choice Representatives

Main idea: the insecure scheme $\mathrm{Mac}_k(m) = k \cdot m$ is a one-time secure MAC for a random choice of plaintext unknown to the adversary

Details:

- MAC key k_i is user-dependent
- $\mathrm{Enc}_{pk}(k_i)$ is generated by a RO, i.e., k_i is known in encrypted form
- assign unique representative m_{ij} to jth choice of the ith voter
- $\mathrm{Enc}_{pk}(m_{ij})$ is generated by a RO, i.e., m_{ij} is known in encrypted form
- the corresponding return code is $RC = \mathrm{Mac}_{k_i}(m_{ij}) = k_i \cdot m_{ij}$
- voter i sends re-encryption of one of its designated ciphertexts + ZKP
- servers jointly decrypt the received ciphertext multiplied by $\mathrm{Enc}_{pk}(k_i)$

Tallying: to decode the votes, the shuffled lists of representatives are also decrypted but *after* all votes have been collected

Scheme III: One-time Pad and Standard MAC Schemes

The idea of one-time pad can be modified to work with standard MACs

$$\mathrm{Mac}'_{k,k_i}(m) = \mathrm{Mac}_k(k_i \cdot m)$$

- **drawback:** distributed MAC compution is not efficient in general
- **good news:** efficient solution exists

Scheme III: One-time Pad and Standard MAC Schemes

The idea of one-time pad can be modified to work with standard MACs

$$\mathrm{Mac}'_{k,k_i}(m) = \mathrm{Mac}_k(k_i \cdot m)$$

- **drawback:** distributed MAC compution is not efficient in general
- **good news:** efficient solution exists
- no need for random representatives
- from rump session of E-Vote-ID 2015 (by D. Wikström)
- apparently Scytl has used similar schemes
- details in the paper

Scheme IV: Diffie-Hellman MAC Schemes

Main idea: given g^{k_i} and g^{b_j}, hard to distinguish $g^{k_i b_j}$ from random

Scheme IV: Diffie-Hellman MAC Schemes

Main idea: given g^{k_i} and g^{b_j}, hard to distinguish $g^{k_i b_j}$ from random

- **First variant**
  - g^{b_j} is randomly chosen and publicly known, representing the jth choice
  - MAC key k_i is user-dependent and secret shared among the servers
  - voter i sends an encryption $\mathrm{Enc}_{pk}(g^{b_j})$ based on his choice
  - servers jointly exponentiate $\mathrm{Enc}_{pk}(g^{b_j})^{k_i}$ and decrypt $\mathrm{Enc}_{pk}(g^{k_i \cdot b_j})$

Scheme IV: Diffie-Hellman MAC Schemes

Main idea: given g^{k_i} and g^{b_j}, hard to distinguish $g^{k_i b_j}$ from random

- **First variant**
  - g^{b_j} is randomly chosen and publicly known, representing the jth choice
  - MAC key k_i is user-dependent and secret shared among the servers
  - voter i sends an encryption $\mathrm{Enc}_{pk}(g^{b_j})$ based on his choice
  - servers jointly exponentiate $\mathrm{Enc}_{pk}(g^{b_j})^{k_i}$ and decrypt $\mathrm{Enc}_{pk}(g^{k_i \cdot b_j})$
- **Second variant**
  - avoids using user-dependent secret shared values
  - **idea:** switch the roles of k_i and b_j
  - preparation and vote submission similar to Scheme II
  - representatives may be opened before receiving all encrypted votes
  - MAC function can be evaluated in batches
  - details in the paper

Offline/Online servers

Drawback of Schemes I-IV: servers need to be online

How to resolve:

- use two independent public keys, with shared secret keys
- one used by offline servers for tallying
- one used by online servers for collecting votes and computing RCs
- can be handled without considerable performance loss or organizational overhead

Scheme V: Offline Tallying Servers

- voter i submits a pair of ciphertexts (v_i, w_i)
- v_i:
  - ciphertext under public key of *tallying servers*
  - *does contain information* about voter's choice
- w_i:
  - encryption of a random value under public key of *vote collecting servers*
  - contains *no information* about the voter's choice
- underlying plaintexts are *paired* by construction
- two variants given in the paper
- to compute the return code, the online servers jointly decrypt w_i
- tallying can be performed behind an *airwall* by distributedly shuffling and decrypting v_i's

Concluding remarks

- our schemes generalize previously proposed schemes
- based on well-understood cryptographic notions
- easily analyzed
- a powerful toolbox to construct return codes suitable for many types of elections
- each has pros and cons — none can be consider *superior*
- subtle discussions omitted (e.g., write-ins, revoting, ZKP, ...)

Thank you