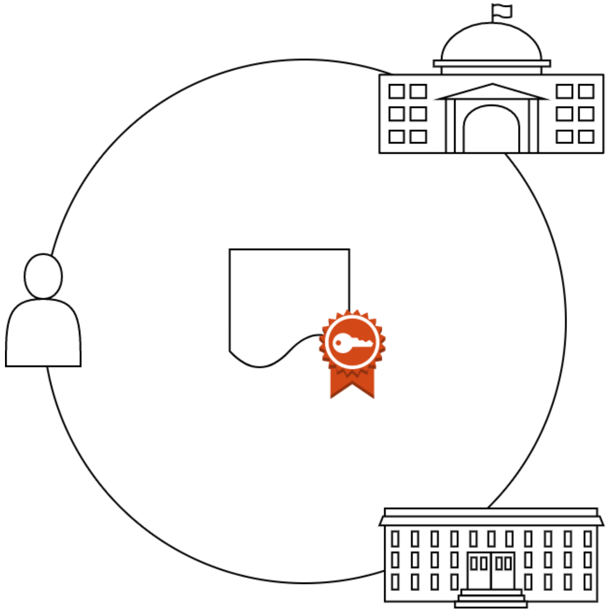


Cryptographic Primitives I

Markus Eggimann

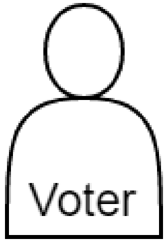
Content

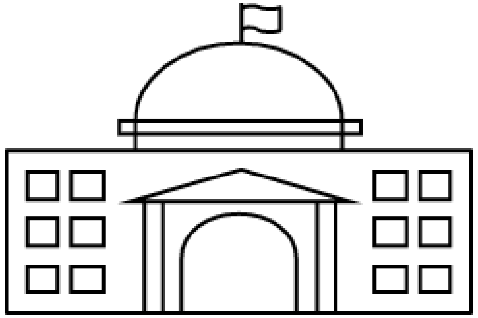
- Blind Signatures
- Homomorphic Encryption
- Commitment Schemes
- Discussion

Blind Signatures

Signing the unknown

Context



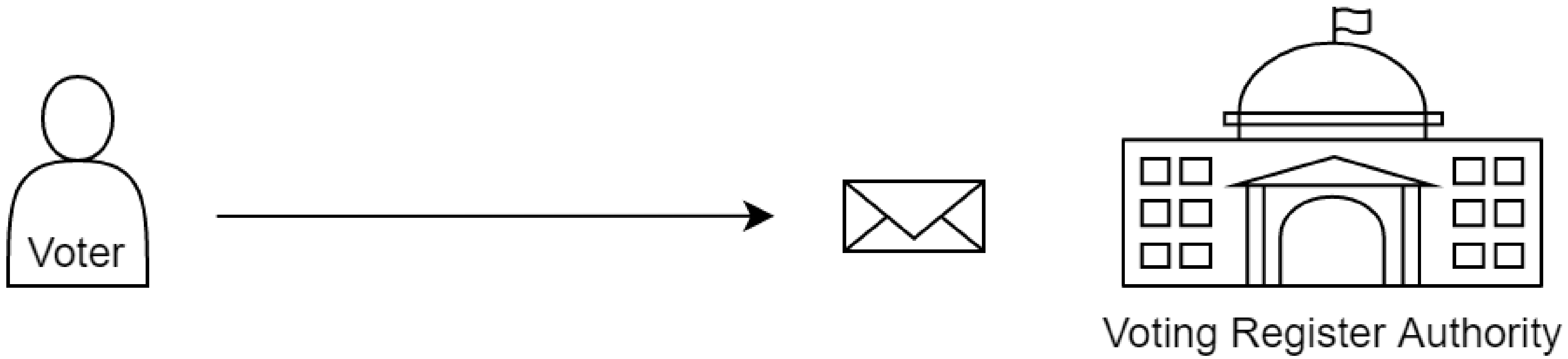


Voting Register Authority

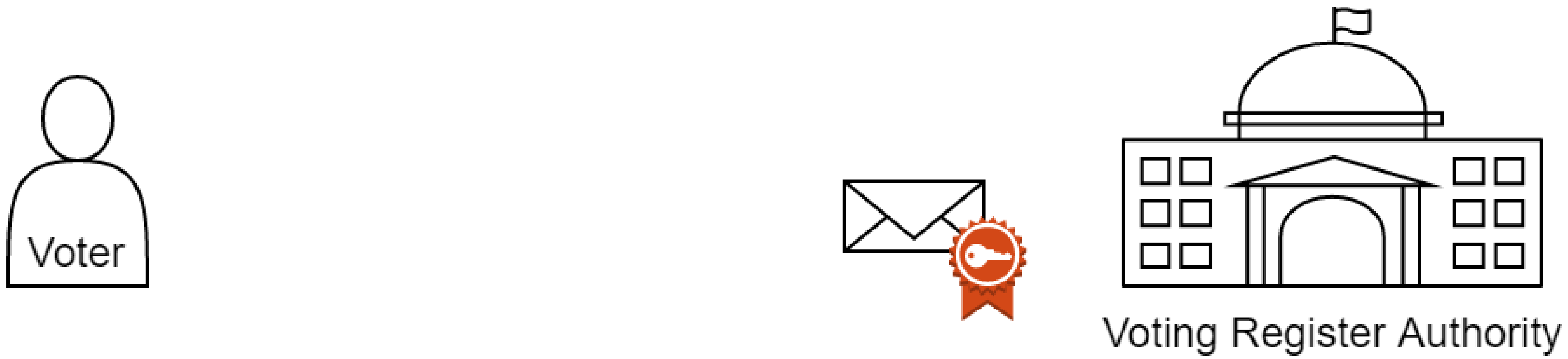
Context

Context

Context



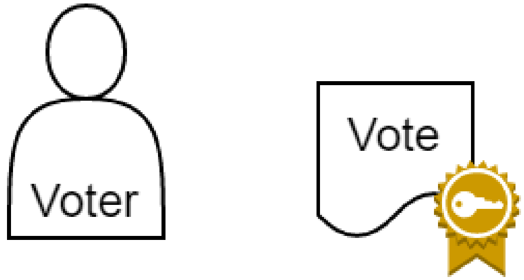
Context

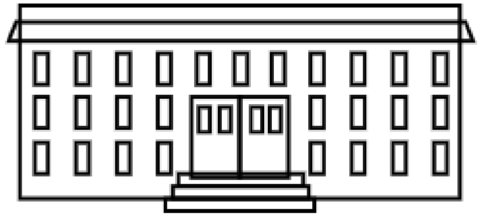


Context

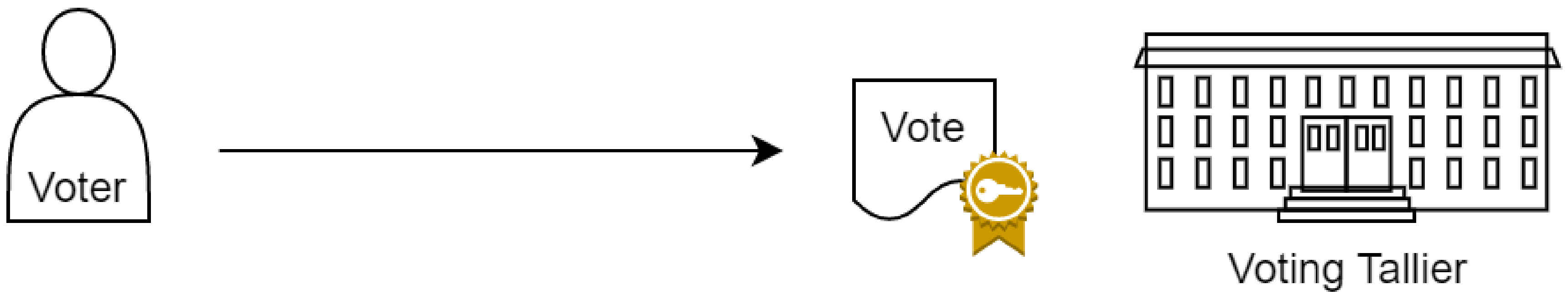
Context

Context

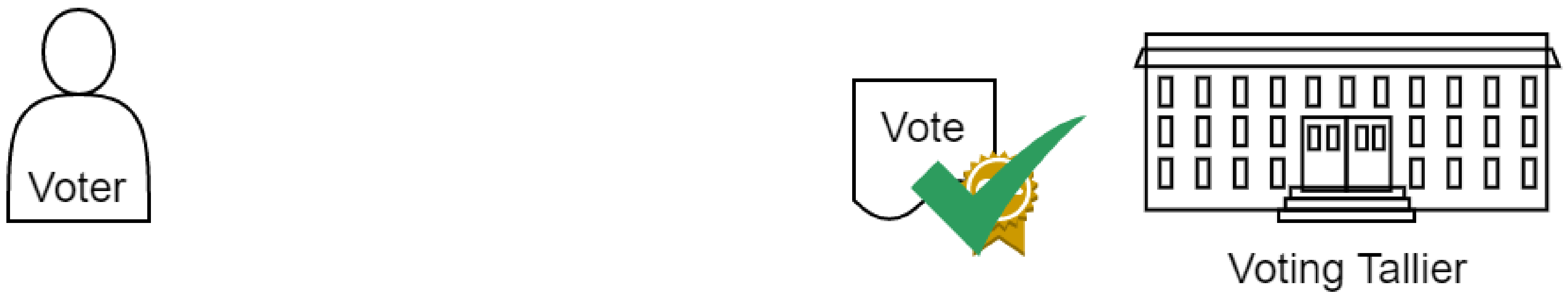




Context



Context



Definition

- $BS = (\mathrm{KeyGen}, \mathrm{Blind}, \mathrm{Sign}, \mathrm{Unblind}, \mathrm{Verify})$

- $(sk, vk) \leftarrow \mathrm{KeyGen}()$ // signing authority
- $(\tilde{m}, u) \leftarrow \mathrm{Blind}(m, vk)$ // voter
- $\tilde{\sigma} \leftarrow \mathrm{Sign}(sk, \tilde{m})$ // signing authority
- $\sigma \leftarrow \mathrm{Unblind}(vk, \tilde{\sigma}, u)$ // voter
- $v \leftarrow \mathrm{Verify}(vk, m, \sigma)$ // tallier

Properties

- Correctness
- Security
  - For signing authority: signatures are unforgeable
  - For voter: signing authority does not learn the vote

Implementation

- Based on **RSA** construction

- $\mathrm{KeyGen}() \coloneqq \big((N, e), (N,\ d)\big) \leftarrow \mathrm{RSA{-}KeyGen}()$
- $\mathrm{Blind}\big((N, e), m\big)\ \coloneqq (\widetilde{m}, u) \leftarrow u \coloneqq r\ \in \mathbb{Z}_N^*;\ \widetilde{m} \coloneqq \mathrm{H}(m) \cdot r^e \pmod N$
- $\mathrm{Sign}\big((N, d), \widetilde{m}\big) \coloneqq \tilde{\sigma} \leftarrow\ \widetilde{m}^d \pmod N$
- $\mathrm{Unblind}\big((N, e). \tilde{\sigma}, u\big)\ \coloneqq\ \sigma \leftarrow\ \tilde{\sigma} \cdot u^{-1} \pmod N$
- $\mathrm{Verify}\big((N, e), m, \sigma\big)\ \coloneqq \begin{cases} 1, & \text{if } \sigma^e = \mathrm{H}(m) \pmod N \\ 0, & \text{otherwise} \end{cases}$

Demo

- Implemented using C# and .NET Framework 6
- Functional tests using Xunit

Homomorphic Encryption

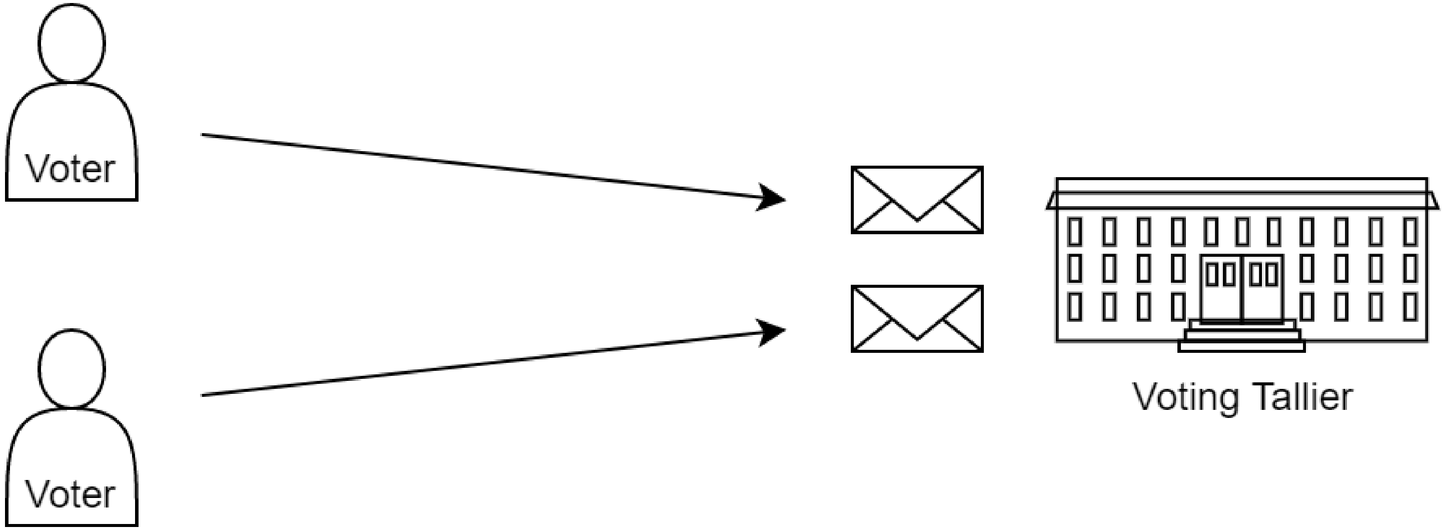
The art of adding two numbers – without actually knowing the numbers

Context

Context

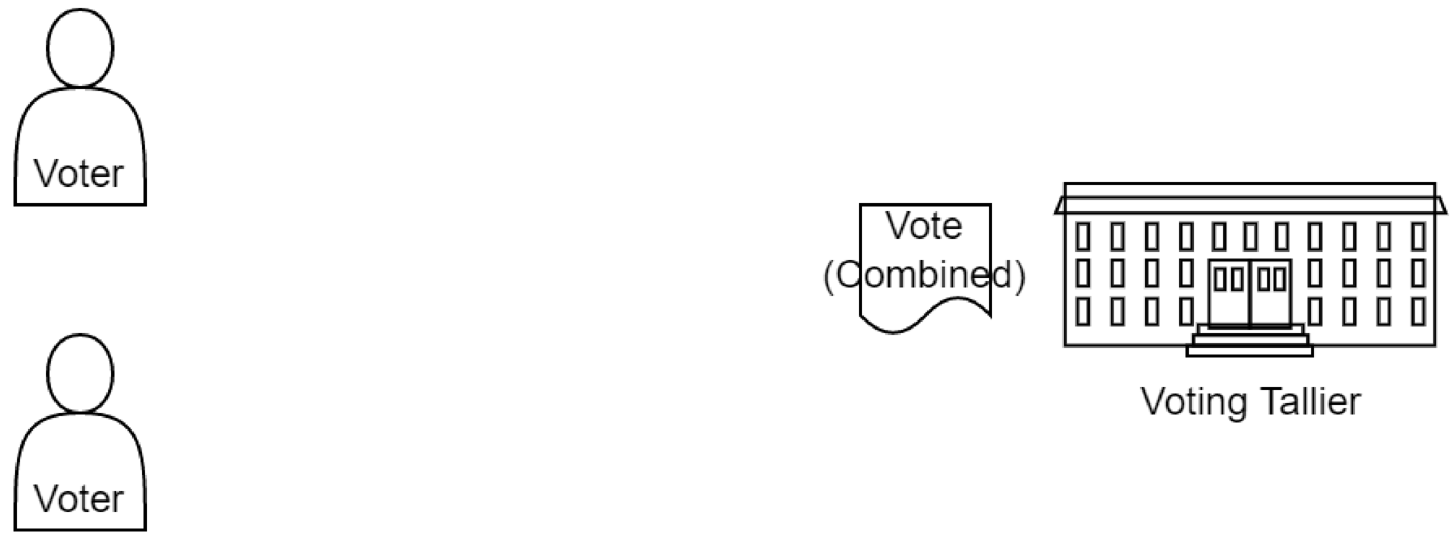
Context

Context



Context

Context



Definition

- $\mathrm{E} = (\mathrm{KeyGen}, \mathrm{Encrypt}, \mathrm{Decrypt}, \oplus, \mathrm{Add})$

- $(pk, sk) \leftarrow \mathrm{KeyGen}()$
- $c_1 \leftarrow \mathrm{Encrypt}(pk, m_1)$
- $c_2 \ \leftarrow \mathrm{Encrypt}(pk, m_2)$
- $c \ \leftarrow \mathrm{Add}(pk, c_1, c_2)$
- $d \ \leftarrow \mathrm{Decrypt}(sk, c)$

Properties

- Correctness
  - $\mathrm{Decrypt}\big(sk, \mathrm{Encrypt}(pk, m)\big) == m$
  - $\mathrm{Decrypt}\Big(sk, \mathrm{Add}\big(pk, \mathrm{Encrypt}(pk, m_1), \mathrm{Encrypt}(pk, m_2)\big)\Big) \ == m_1 \oplus m_2$

- Security
  - Attacker cannot learn anything about the plain-text
  - Decryptor cannot tell whether message is composite

Implementation

- Based on the **El Gamal** crypto-system
- Operates in Diffie-Hellman group $\langle g \rangle \coloneqq (p, q, g) \subset \mathbb{Z}_p^*$
- Based on **Discrete Logarithm Problem**

- $\mathrm{KeyGen}() \coloneqq (sk, pk) \leftarrow sk \in \mathbb{Z}_q^*; pk \coloneqq g^{sk} \pmod p$
- $\mathrm{Encrypt}(pk, m) \coloneqq (c, d) \leftarrow c \coloneqq g^r \pmod p; d \coloneqq m \cdot pk^r \pmod p$
- $\mathrm{Decrypt}\big(sk, (c, d)\big) \coloneqq \widetilde{m} \leftarrow d \cdot \left(c^{sk}\right)^{-1} \pmod p$
- $\oplus \coloneqq m_1 \cdot m_2 \pmod p$
- $\mathrm{Add}\big(pk, (c_1, d_1), (c_2, d_2)\big) \coloneqq (c', d') \leftarrow \big(c_1 \cdot c_2 \pmod p, d_1 \cdot d_2 \pmod p\big)$

Demo

- Implemented using C# and .NET Framework 6
- Functional tests using Xunit

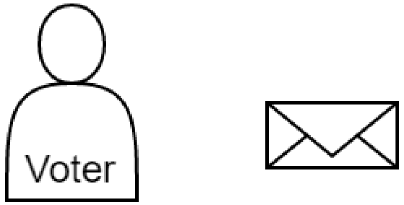
Commitment Schemes

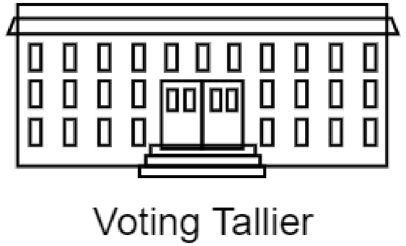
You cannot simply change your opinion

Context

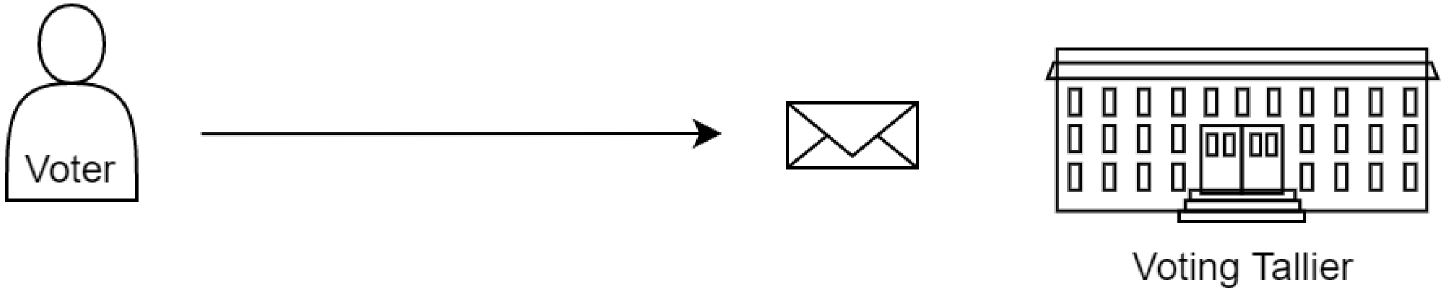
Context

Context

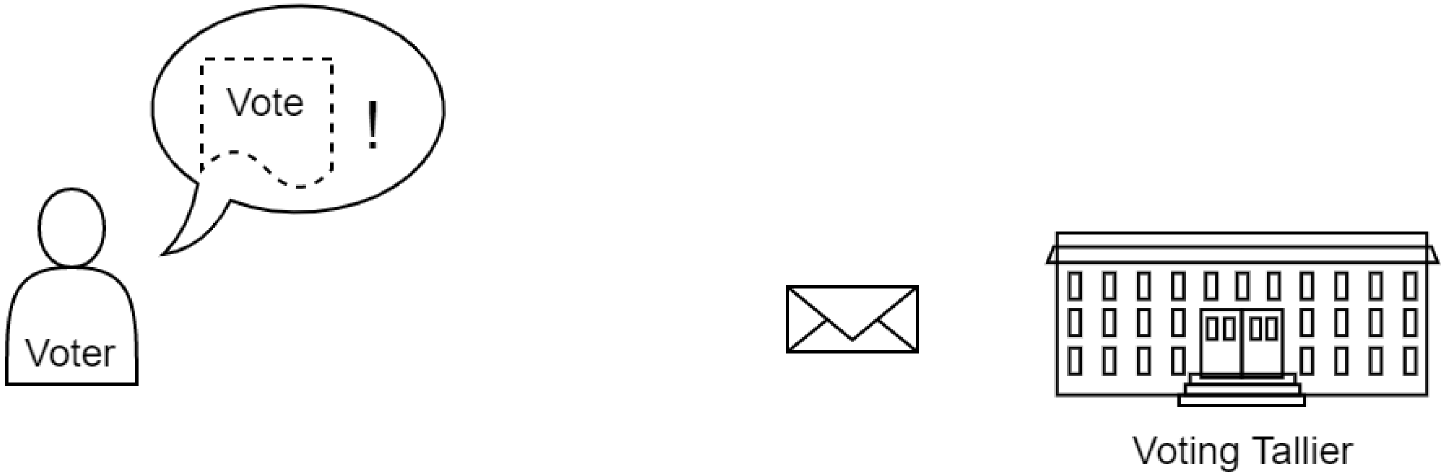




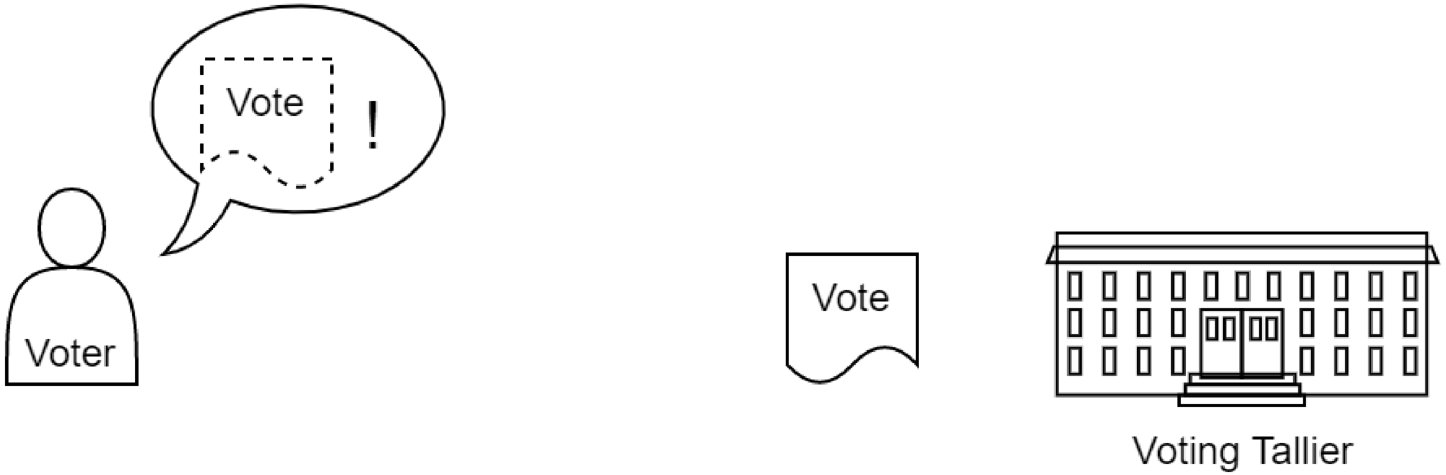
Context



Context



Context



Definition

- $CS = (\mathrm{Setup}, \mathrm{Commit}, \mathrm{Reveal})$

- $pk \leftarrow \mathrm{Setup}()$ // receiver
- $(c, k) \leftarrow \mathrm{Commit}(pk, m)$ // committer, *c* sent to receiver
- $b \leftarrow \mathrm{Reveal}(pk, m, c, k)$ // receiver, after committer revealed *m* and *k*

Properties

- Correctness
- Security
  - Hiding: attacker cannot learn committed value
  - Binding: committer cannot change committed value

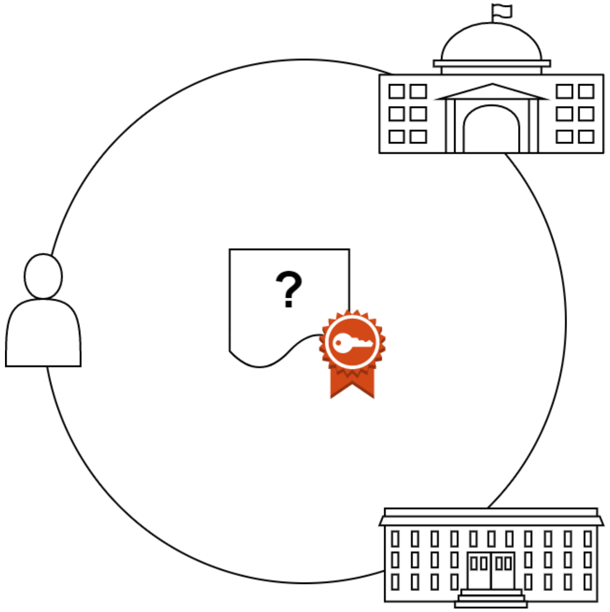
Implementation

- **Pedersen** Commitment Scheme
- Operates in Diffie-Hellman group $\langle g \rangle \coloneqq (p, q, g) \subset \mathbb{Z}_p^*$
- Based on **Discrete Logarithm Problem**

- Setup() $\coloneqq pk \leftarrow sk \in \mathbb{Z}_q^*; pk \coloneqq g^{sk} \pmod p$
- Commit() $\coloneqq (c, k) \leftarrow k \coloneqq r \in \mathbb{Z}_q^*; c \coloneqq g^m \cdot pk^r \pmod p$
- Reveal$(pk, m, c, k) \coloneqq b \leftarrow c == g^m \cdot pk^r \pmod p$

Demo

- Implemented using C# and .NET Framework 6
- Functional tests using Xunit



Discussion

References

Cryptographic Voting - A Gentle Introduction
David Bernhard, Bogdan Warinschi
https://eprint.iacr.org/2016/765
2013

Cryptography Made Simple
Nigel P. Smart
2016

Implementation of RSA Blind Signatures, El Gamal Homomorphic Encryption and Pedersen commitments
Markus Eggimann
https://github.com/meggima/seminar-crypto-fall-2022
2022

Backup Slides

RSA Blind Signature: Correctness

- $\tilde{m} \coloneqq H(m) \cdot u^e$
- $\tilde{\sigma} \ \coloneqq \tilde{m}^d (\bmod N) = (H(m) \cdot u^e)^d$
- $\sigma \ \coloneqq \tilde{\sigma} \cdot u^{-1} = (H(m) \cdot u^e)^d \cdot u^{-1}$
- Verify: $\sigma^e = \left((H(m) \cdot u^e)^d \cdot u^{-1}\right)^e$

$$= \left(H(m)^d \cdot (u^e)^d \cdot u^{-1}\right)^e$$

$$= \left(H(m)^d \cdot 1\right)^e$$

$$= H(m)^{d^e} \cdot 1^e$$

$$= H(m)$$

All calculations $(\bmod N)$

Homomorphic El Gamal: Correctness I

- Encryption/Decryption:

  $Dec\big(sk, Enc(pk, m)\big)$

  $= d \cdot \left(c^{sk}\right)^{-1}$

  $= (m \cdot pk^{r}) \cdot (g^{r})^{sk^{-1}}$

  $= \left(m \cdot g^{sk^{r}}\right) \cdot (g^{r})^{sk^{-1}}$

  $= m$

All operations $\mathrm{mod}\ (p)$

Homomorphic El Gamal: Correctness II

- Homomorphic property:

$c_1 \cdot c_2 = g^{r_1} \cdot g^{r_2}$

$d_1 \cdot d_2 = m_1 \cdot pk^{r_1} \cdot m_2 \cdot pk^{r_2}$

Decrypt: $d_1 \cdot d_2 \cdot (g^{r_1} \cdot g^{r_2})^{sk^{-1}}$

$$= m_1 \cdot pk^{r_1} \cdot m_2 \cdot pk^{r_2} \cdot (g^{r_1} \cdot g^{r_2})^{sk^{-1}}$$

$$= m_1 \cdot g^{sk^{r_1}} \cdot m_2 \cdot g^{sk^{r_2}} \cdot (g^{r_1} \cdot g^{r_2})^{sk^{-1}} = m_1 \cdot m_2$$

All operations $\mathrm{mod}\ (p)$