# Blind accumulators for e-voting

**Sergey Agievich**
Research Institute for Applied Problems of Mathematics and Informatics
Belarusian State University

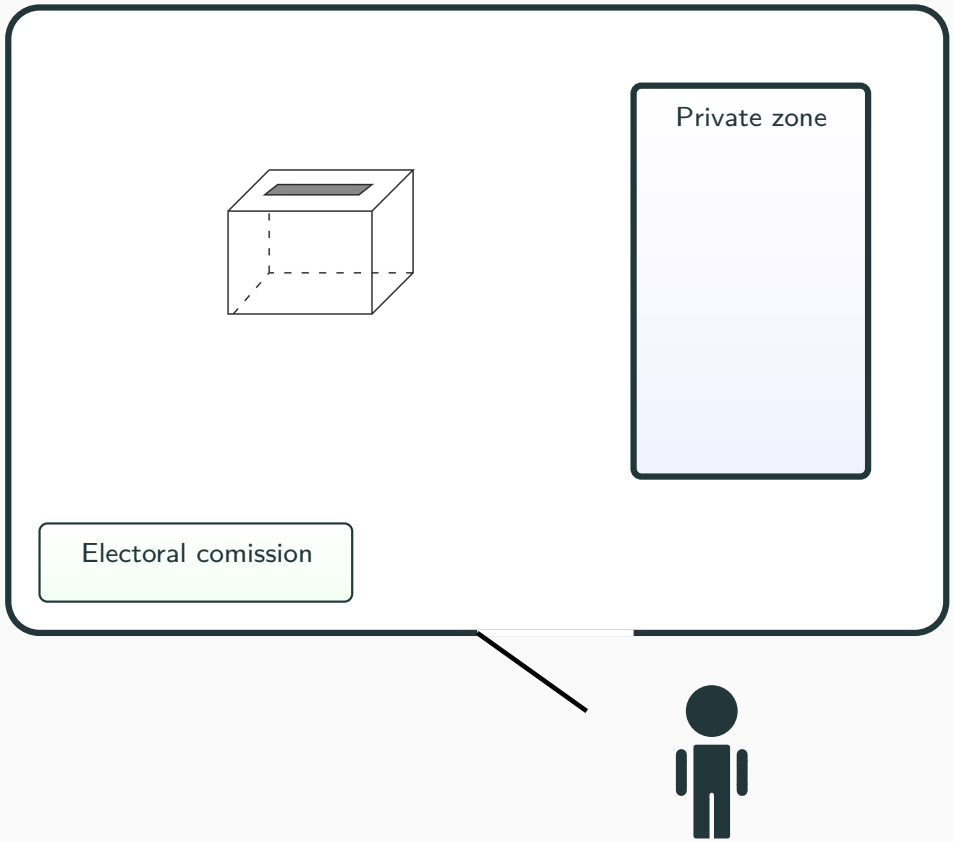
June 28, 2022 [Minsk – Smolenice]

CECC 2022

# Table of contents

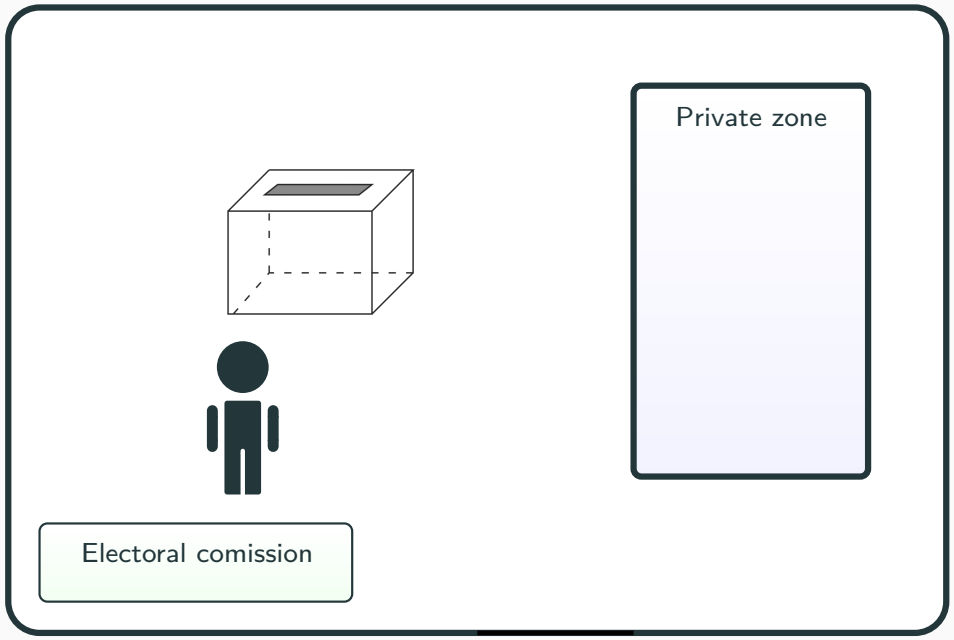
# Introduction

# Conventional voting




**voter**

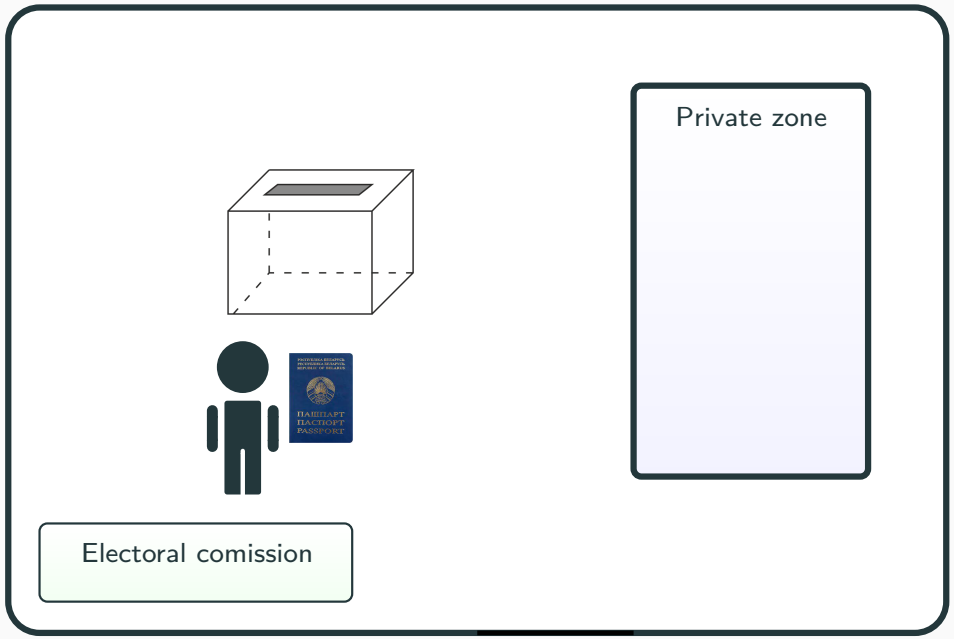
# Conventional voting




**voter**

1) enters a polling station

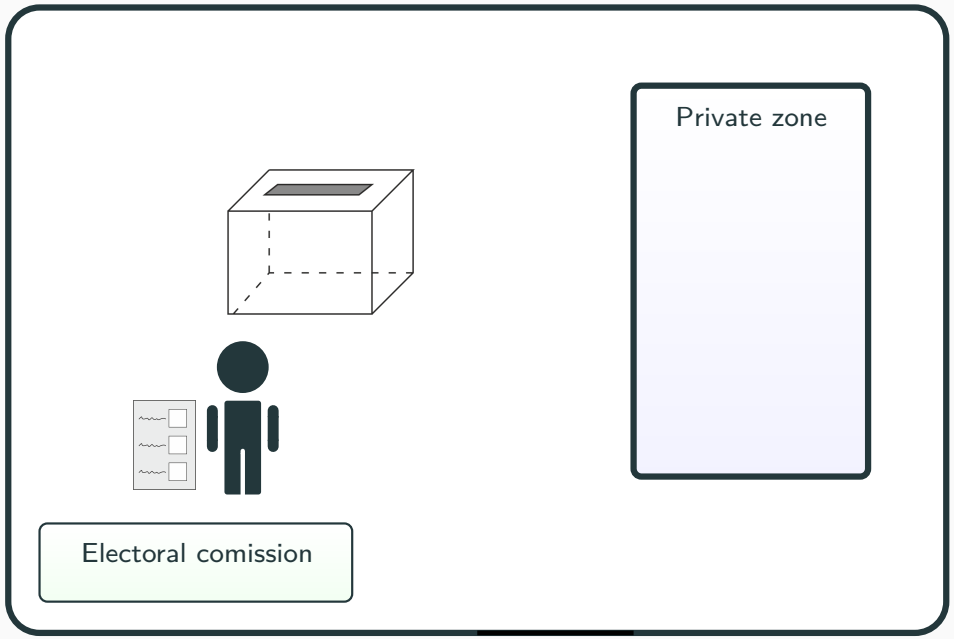
# Conventional voting




**voter**

1) enters a polling station
2) is authenticated
3) is checked for affiliation

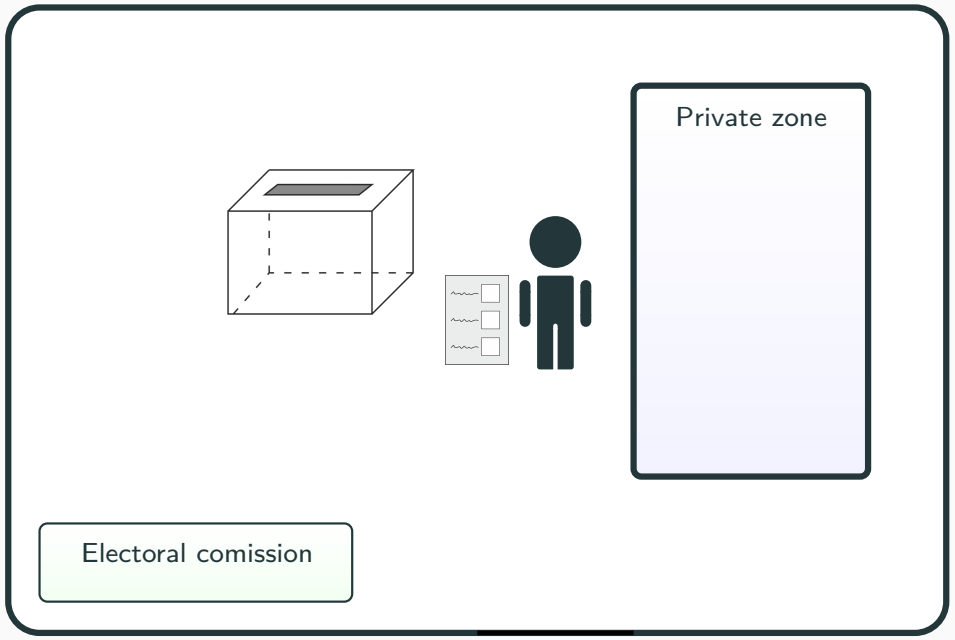
# Conventional voting




**voter**

1) enters a polling station
2) is authenticated
3) is checked for affiliation
4) gets a ballot

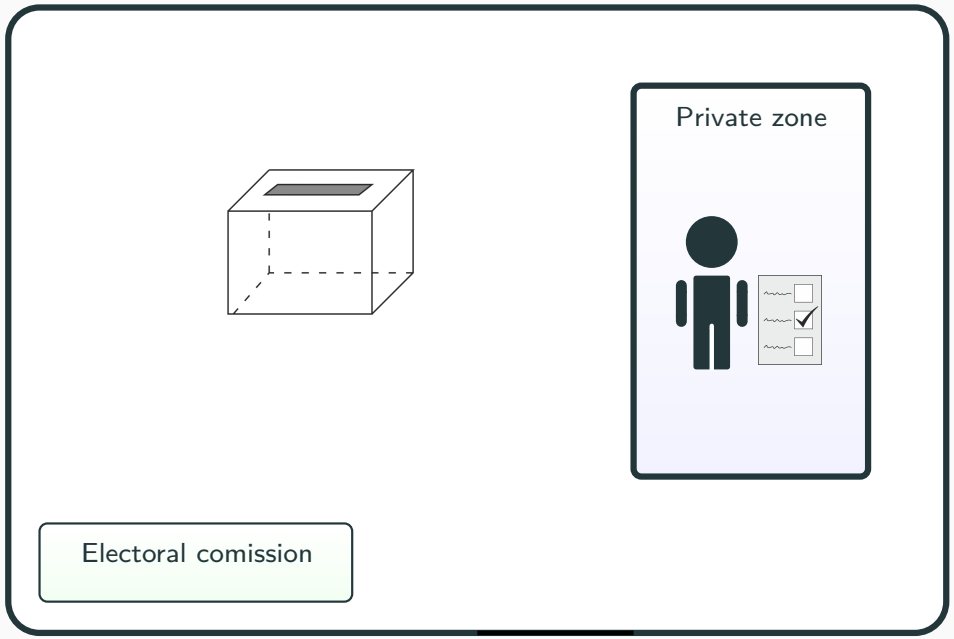
# Conventional voting




**voter**

1) enters a polling station
2) is authenticated
3) is checked for affiliation
4) gets a ballot
5) visits the private zone

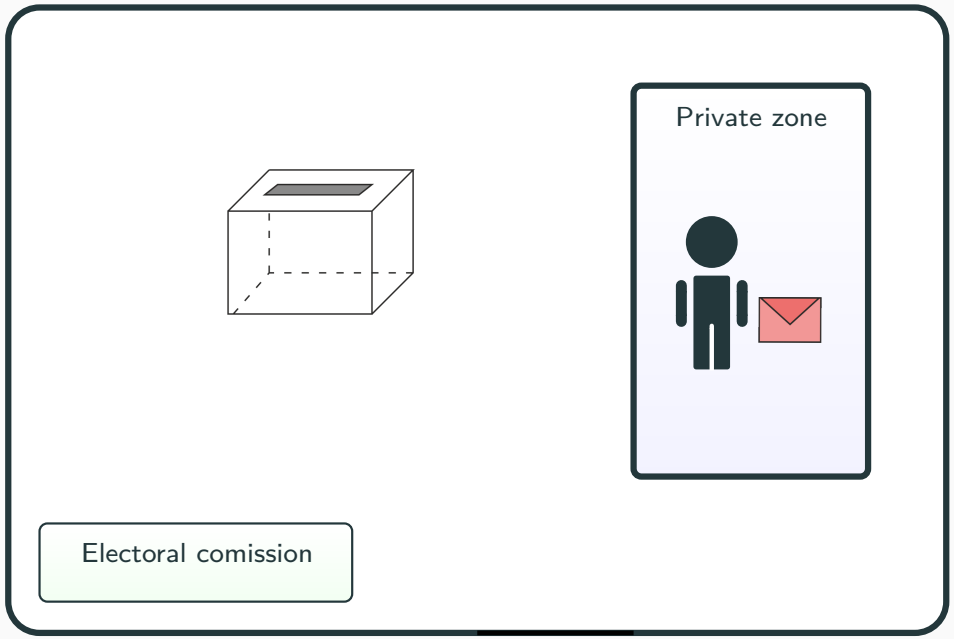
# Conventional voting




**voter**

1) enters a polling station
2) is authenticated
3) is checked for affiliation
4) gets a ballot
5) visits the private zone
6) fills out the ballot

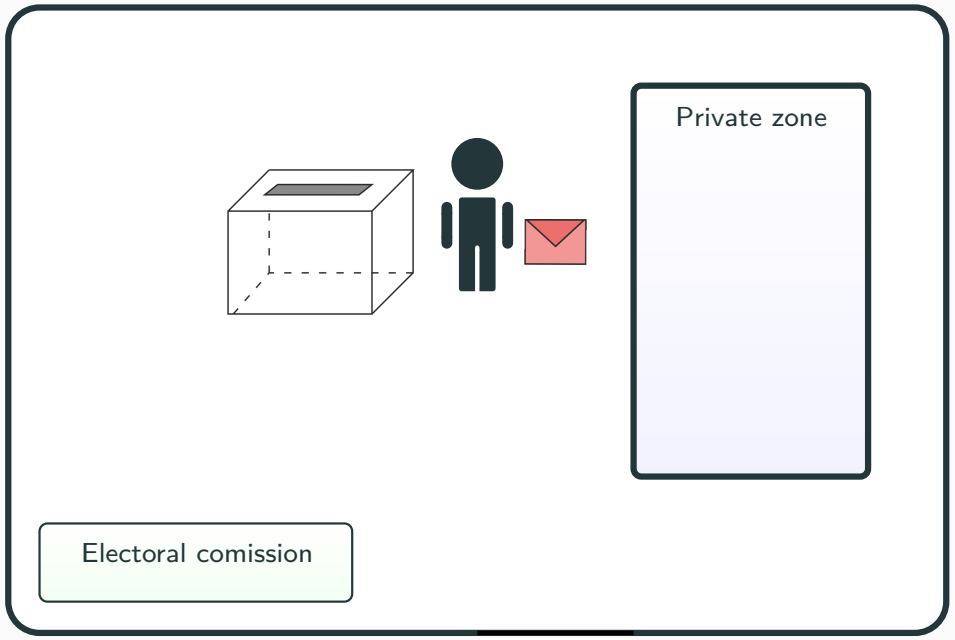
# Conventional voting




**voter**

1) enters a polling station
2) is authenticated
3) is checked for affiliation
4) gets a ballot
5) visits the private zone
6) fills out the ballot
7) folds the ballot

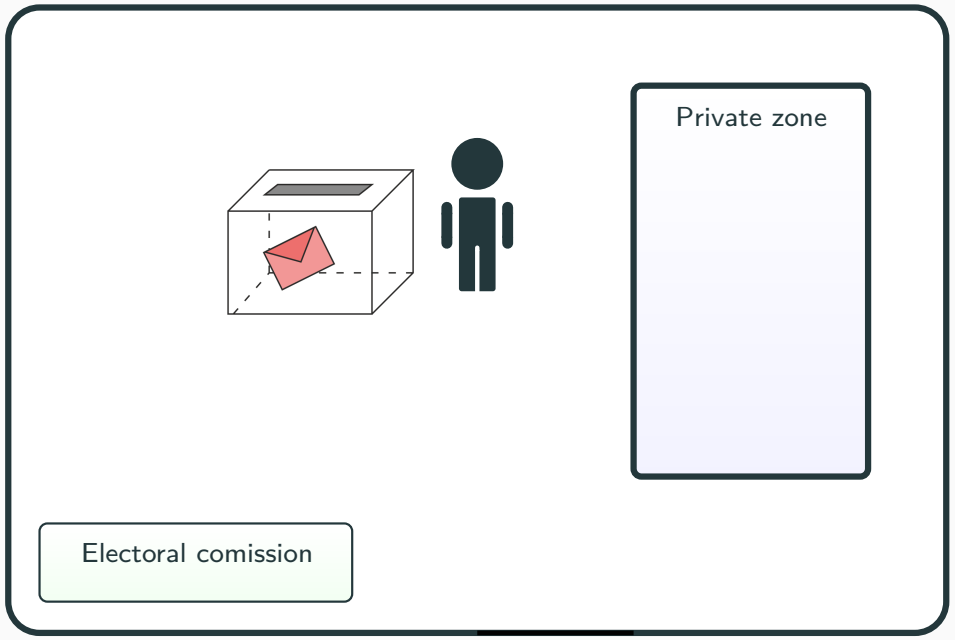
# Conventional voting




**voter**

1) enters a polling station
2) is authenticated
3) is checked for affiliation
4) gets a ballot
5) visits the private zone
6) fills out the ballot
7) folds the ballot
8) leaves the zone

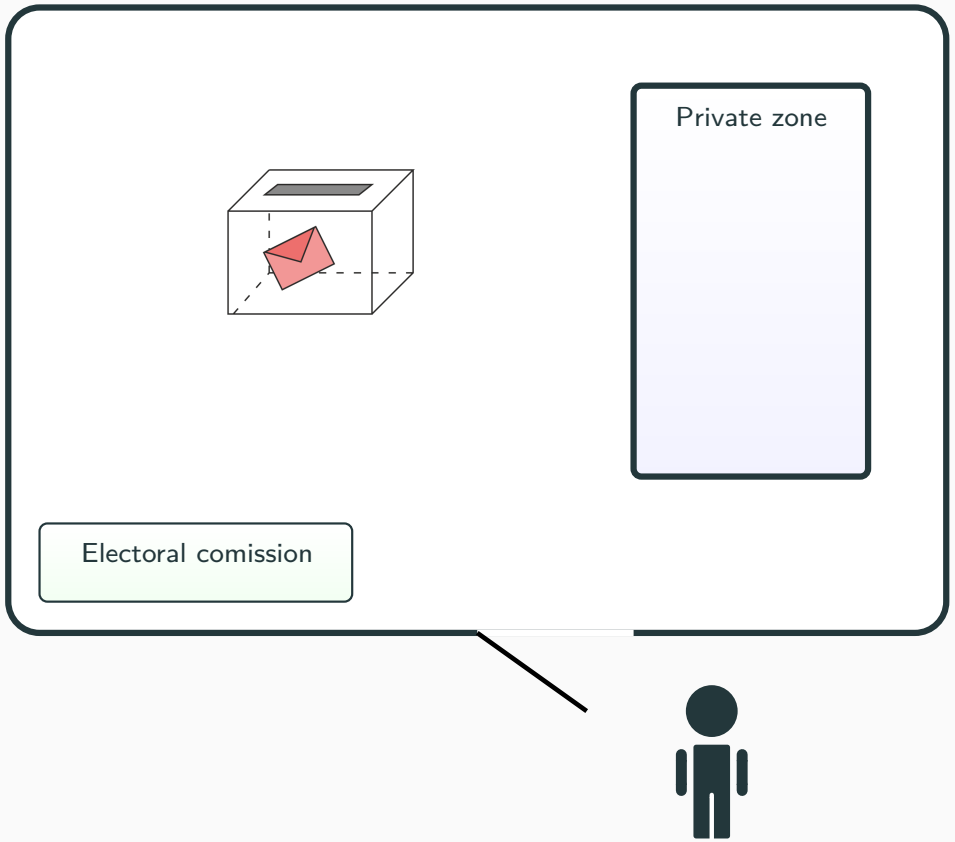
# Conventional voting




**voter**

1) enters a polling station
2) is authenticated
3) is checked for affiliation
4) gets a ballot
5) visits the private zone
6) fills out the ballot
7) folds the ballot
8) leaves the zone
9) casts the ballot

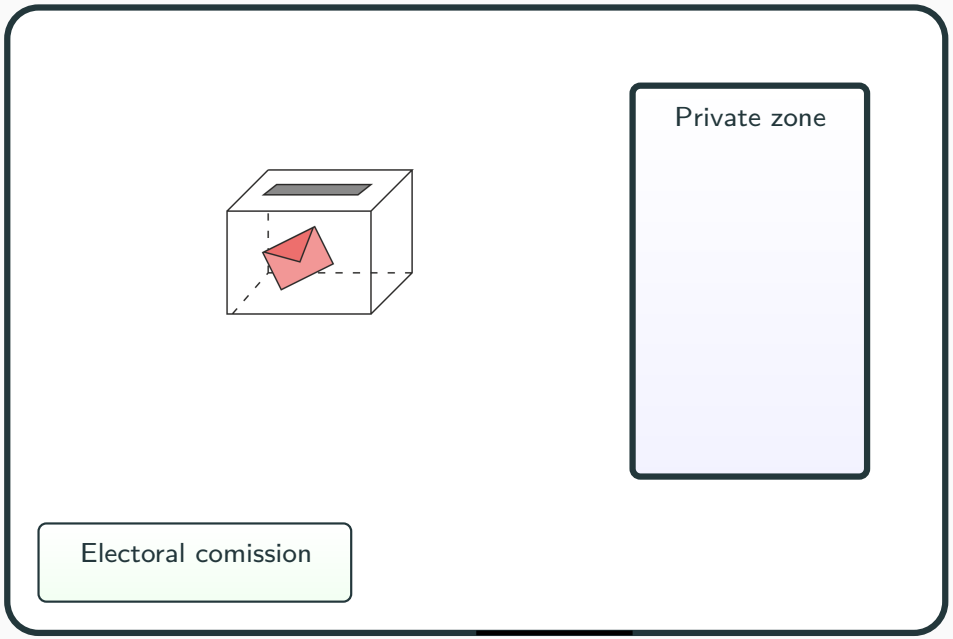
# Conventional voting




**voter**

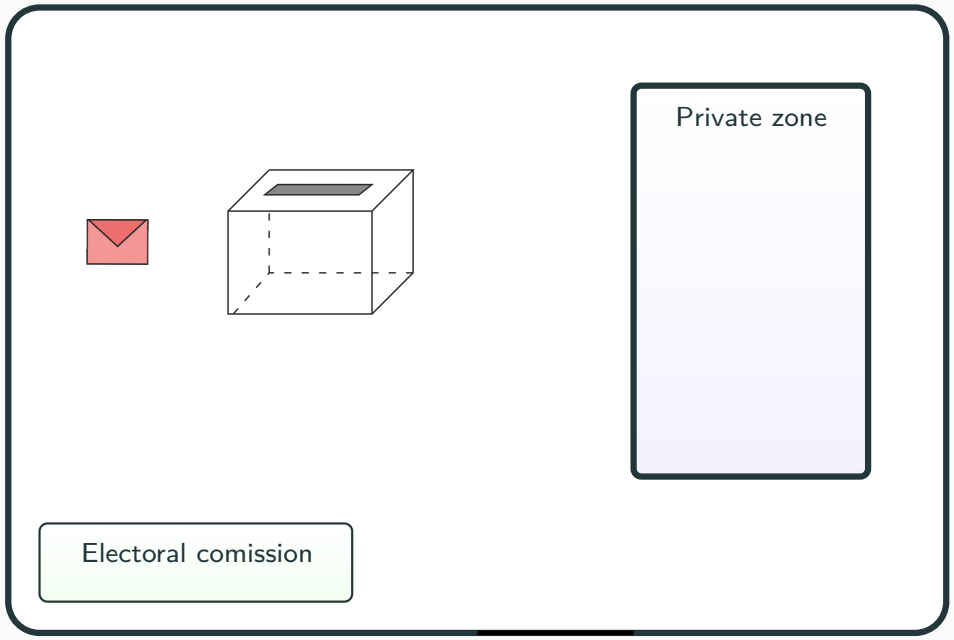
1) enters a polling station
2) is authenticated
3) is checked for affiliation
4) gets a ballot
5) visits the private zone
6) fills out the ballot
7) folds the ballot
8) leaves the zone
9) casts the ballot
10) exits the station

# Conventional voting




**electoral comission**

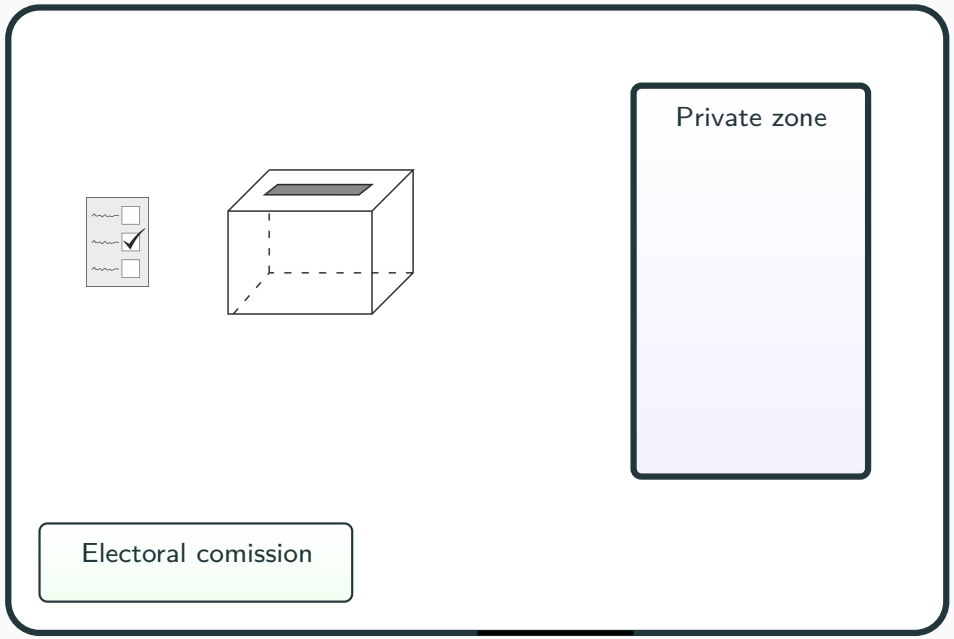
# Conventional voting




**electoral comission**

1) opens the ballot box

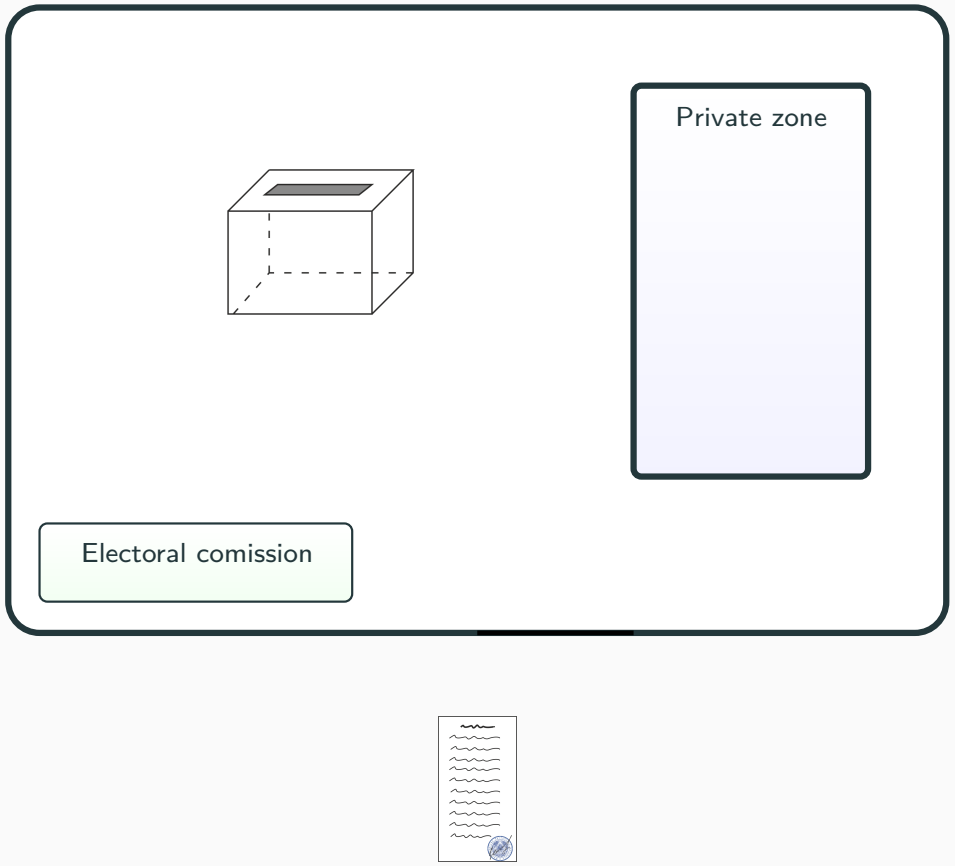
# Conventional voting




**electoral comission**

1) opens the ballot box
2) unfolds the ballots
3) sums up the votes

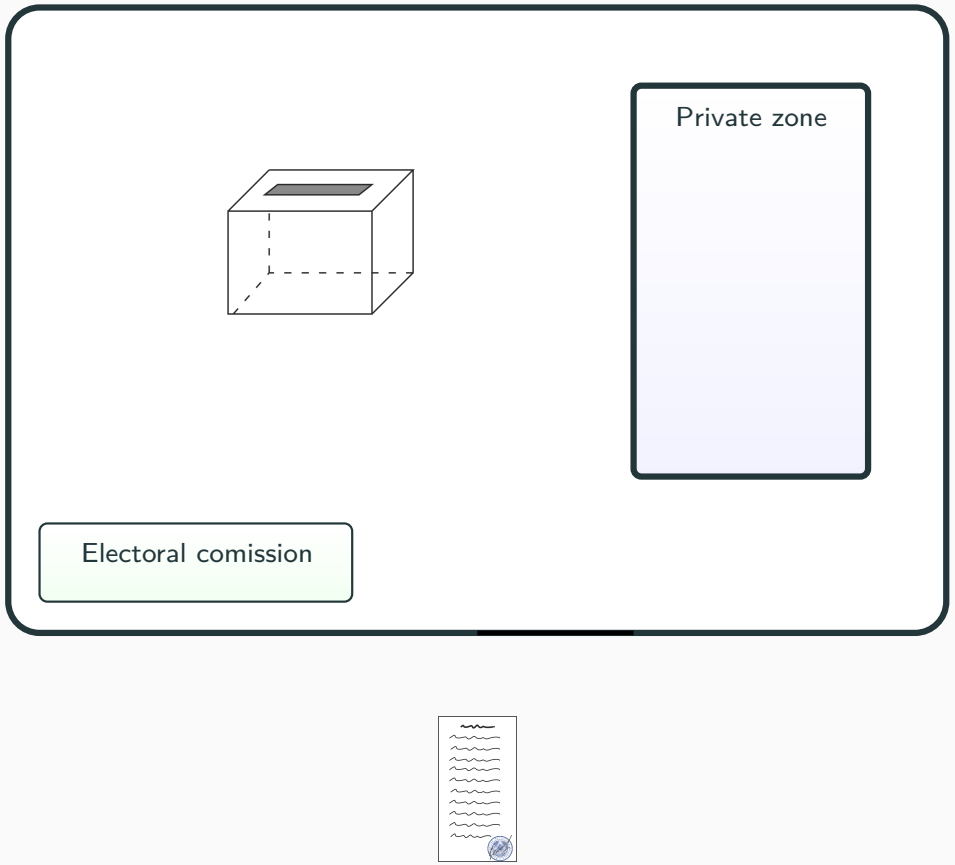
# Conventional voting




**electoral comission**

1) opens the ballot box
2) unfolds the ballots
3) sums up the votes
4) publishes the result

# Conventional voting*




**electoral comission**

1) opens the ballot box
2) unfolds the ballots
3) sums up the votes
4) publishes the result

*__Informally__: “enclave” (private zone) in “conclave” (polling station).

# Properties

## The "conclave" functionality

1. **Consistency**: at any time during the voting, a voting system is in a correct state.
2. **Eligibility**: only eligible voters vote.

## The "enclave" functionality

3. **Privacy**: individual votes remain secret.

# Properties

## The "conclave" functionality

1. **Consistency**: at any time during the voting, a voting system is in a correct state.
2. **Eligibility**: only eligible voters vote.

## The "enclave" functionality

3. **Privacy**: individual votes remain secret.

## Additional properties (not satisfied but we need them)

4. **Verifiability**: voters should be able to verify if their votes are correctly accounted for.
5. **Decentralization**: There is no electoral commission, voters jointly control the voting process.

# The concept

To construct e-voting systems satisfying properties 1 – 5, we propose to use **blind accumulators**:

- A blind accumulator acts as a **digital conclave** that collects private keys from **digital enclaves** of voters doing this in a decentralized manner and not getting information about the keys.
- Once the accumulation is complete, a voter processes the resulting accumulator deriving a public key that refers to the private key previously added by this voter.
- Public keys are derived deterministically and can therefore stand as fixed voter pseudonyms.
- The voter can prove that the derived key refers to some accumulated private key without revealing neither that key nor the voter itself.
- The voter uses the accumulated private key to sign a ballot. The corresponding public key is used to verify the signature.

# Blind accumulators

# Accumulators

Cryptographic accumulators are special encodings of tuples of objects.

We write $\mathbf{a} = [S]$ to denote that an accumulator $\mathbf{a}$ encodes a tuple $S$.

Accumulators are managed by algorithms that translate operations involving $S$ into operations over $[S]$.

# Accumulators

Cryptographic accumulators are special encodings of tuples of objects.

We write $\mathbf{a} = [S]$ to denote that an accumulator $\mathbf{a}$ encodes a tuple $S$.

Accumulators are managed by algorithms that translate operations involving $S$ into operations over $[S]$.

Typically, an accumulator $[S]$ as well as the underlying tuple $S$ are public. In our case, this is not true: $[S]$ remains public but $S$ consists of private keys known only to their owners. Informally speaking, the accumulator collects objects **blindly**. That is why we call such accumulators **blind**.

We avoid the usual requirement that the encoding $[S]$ has to be succinct.

# Syntax

A **blind accumulator scheme** is a tuple of polynomial-time algorithms $\mathbf{BAcc} = (\mathbf{Init}, \mathbf{Add}, \mathbf{PrvAdd}, \mathbf{VfyAdd}, \mathbf{Der}, \mathbf{PrvDer}, \mathbf{VfyDer})$:*

- $\mathbf{Init}: 1^l \mapsto \mathbf{a}_0$: $l$ is a security level and $\mathbf{a}_0 = [\varnothing]$;
- $\mathbf{Add}: (\mathbf{a}, sk) \mapsto \mathbf{a}'$: $\mathbf{a} = [S]$, $sk$ is a private key, and $\mathbf{a}' = [S \cup \{sk\}]$;
- $\mathbf{PrvAdd}: (\mathbf{a}, \mathbf{a}', sk) \mapsto \alpha$: $\alpha$ is a proof that $\mathbf{a}' = \mathbf{Add}(\mathbf{a}, sk)$;
- $\mathbf{VfyAdd}: (\mathbf{a}, \mathbf{a}', \alpha) \mapsto b$: $b = 1$ if the proof $\alpha$ is accepted and $b = 0$ if rejected;
- $\mathbf{Der}: (\mathbf{a}, sk) \mapsto pk \mid \perp$: $pk$ is a public key associated with $sk$ and $\perp$ is the error symbol;
- $\mathbf{PrvDer}: (\mathbf{a}, pk, sk) \mapsto \delta$: $\delta$ is a proof that $pk = \mathbf{Der}(\mathbf{a}, sk)$;
- $\mathbf{VfyDer}: (\mathbf{a}, pk, \delta) \mapsto b$: $b = 1$ if the proof $\delta$ is accepted and $b = 0$ if rejected.

*$\mathbf{Der}$ = Derive.

# Proofs

## Proof of consistency ($\alpha$)

Blind accumulators are not managed by any trusted party which is usually responsible for maintaining the consistency of accumulators during their updates. Without a trusted party, consistency is maintained in a decentralized manner by validating transitions between $[S]$ and $[S \cup \{sk\}]$. Each transition is accompanied by a proof of consistency generated by a party who adds $sk$ to $S$.

## Proof of membership ($\delta$)

A private key $sk \in S$ added to the accumulator $[S]$ relates to a public key $pk$ which is derived from $[S]$ with $sk$. The derived key is accompanied by a proof that $sk \in S$.

# Security requirements

## Consistency

An (adversarial) algorithm $\mathcal{A}$ that claims to generate a correct proof $\alpha$ not using a private key *sk* actually almost certainly uses it.

So, a transition from **a** to $\mathbf{a}'$ that is confirmed by **VfyAdd** is almost certainly driven by a valid private key and $\mathbf{a}'$ is consistent (that is, correctly encodes a tuple of private keys) provided that **a** is consistent.

## Soundness

Soundness means that if an algorithm $\mathcal{A}$ is able to generate a correct proof $\delta$ that a derived public key *pk* refers to some private key *sk* from an accumulator, then this algorithm almost certainly uses this *sk*.

Therefore, the algorithm is run by an eligible party who previously added *sk* to the accumulator.

# Security requirements (continued)

**Blindness**

The proofs $\alpha$ and $\delta$ generated by the algorithms **PrvAdd** and **PrvDer** do not reveal information about $sk$.

**Unlinkability**

It is hard for a coalition of dishonest parties involved in the accumulator management to distinguish a public key of some (unknown) honest party from a random key.

> This implies the hardness of associating public keys with their owners.

# Pseudonymous key generation

# The PKG protocol

Blind accumulators are embedded in the Pseudonymous Key Generation (PKG) protocol which details the use of accumulators in practical settings close to e-voting.

**Participants**

In PKG, $n$ authorized parties (voters) and a moderator participate.

The parties confirm their authenticity by signing messages with long-term private keys. The corresponding public keys are registered in a trusted infrastructure.

**Pseudonymization**

The PKG protocol performs pseudonymization of public keys: an input public key associated with a **particular** party of the protocol is turned into a public key associated with **some** party.

# The moderator

**Functions**

- initializing the protocol;
- storing accumulators that are updated by the parties during the protocol execution;
- providing access to the accumulators;
- verifying proofs of consistency of the accumulators.

These functions are partially duplicated by the parties themselves, who independently verify the consistency.

A virtual moderation through consensus decisions of the parties is potentially possible.

# Outside of PKG

A party $P$ uses the resulting triple $(sk, pk, \delta)$ in cryptographic systems outside of PKG.

The public key $pk$ stands as a fixed pseudonym of the party.

Each time the pseudonym $pk$ is used, the party has to prove knowledge of $sk$ or, in other words, ownership of the pseudonym.

# Outside of PKG

A party $P$ uses the resulting triple $(sk, pk, \delta)$ in cryptographic systems outside of PKG.

The public key $pk$ stands as a fixed pseudonym of the party.

Each time the pseudonym $pk$ is used, the party has to prove knowledge of $sk$ or, in other words, ownership of the pseudonym.

To prove knowledge of a private key, **BAcc**-friendly systems should be used. These systems are compatible with the relationship between $sk$ and $pk$ established in PKG by the **BAcc** algorithms.

If a **BAcc**-friendly digital signature is constructed, then a party $P$ signs the data with $sk$ and accompanies the signature with the pair $(pk, \delta)$.

For example, a voter signs a ballot.

# Back to e-voting

The correctness of the signature as well as the proof $\delta$ relative to $(\mathbf{a}, pk)$ means that the ballot is signed by one of the eligible voters (**eligibility**) that took part in creating the accumulator $\mathbf{a}$ although it is not known which exactly voter signed (**privacy**).

The proofs accompanying $\mathbf{a}$ ensure the **consistency** of the accumulator and the e-voting in general and non-volatility of $pk$ supports **verifiability**.

The moderation, the only element of centralization in PKG, reduces to providing access to the accumulator $\mathbf{a}$ and accompanying proofs (**decentralization**).

# Implementation

# BAcc-DH

**BAcc-DH** is an implementation of the **BAcc** scheme whose main computations resemble the Diffie–Hellman protocol.

**Public parameters**

A cyclic group $\mathbb{G}_q$ of large prime order $q$.

Conventions and notations:

- $\mathbb{G}_q$ is written additively;
- $\mathbb{G}_q^* = \mathbb{G}_q \setminus \{O\}$;
- $\mathbb{Z}_q$ is the ring of residues of integers modulo $q$;
- $\mathbb{Z}_q^*$ is the set of nonzero (invertible) residues.

Accumulators are non-empty words in the alphabet $\mathbb{G}_q^*$.

# Initialization

**BAcc-DH.Init**

1. Takes a security level $l$.
2. Constructs $\mathbb{G}_q$ ($l$ determines the bit length of $q$).
3. Chooses $G \in \mathbb{G}_q^*$ and outputs $\mathbf{a}_0 = G$.

# Adding private keys

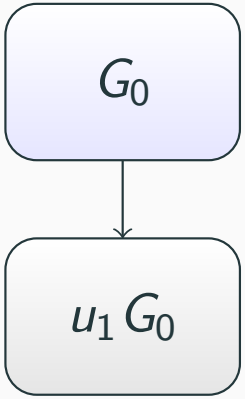
**BAcc-DH.Add**

1. Takes an accumulator $\mathbf{a} = G_0 G_1 \dots G_m$ and a private key $u \in \mathbb{Z}_q^*$.
2. Outputs

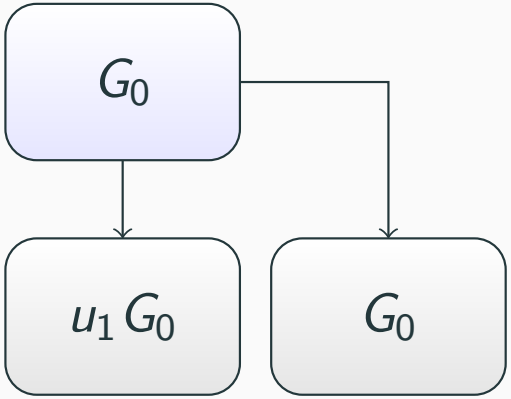
$$\mathbf{a}' = G_0' G_1' \dots G_m' G_0, \quad G_i' = uG_i.$$

# Graphically ($n$ private keys)...

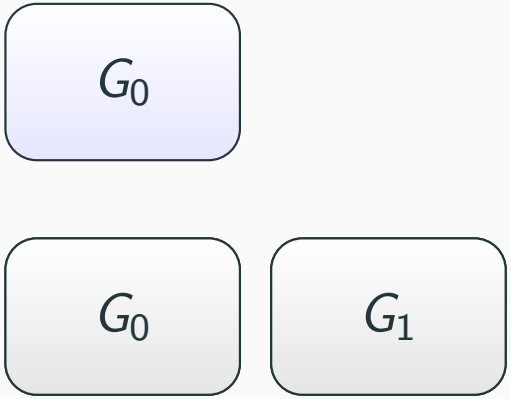
# Graphically ($n$ private keys)...



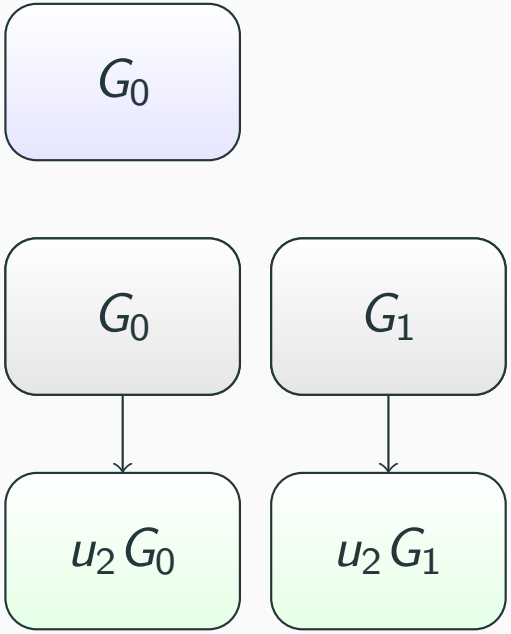
# Graphically ($n$ private keys)...



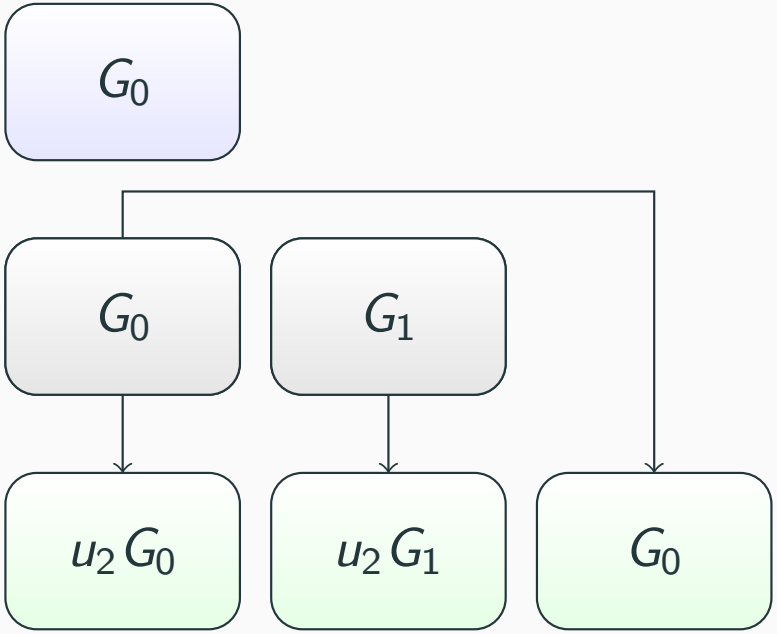
# Graphically ($n$ private keys)...



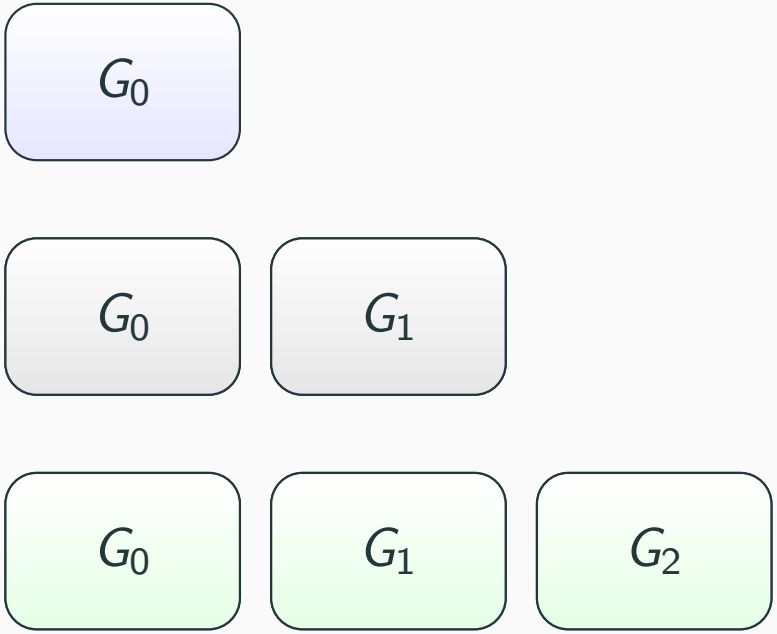
# Graphically ($n$ private keys)...



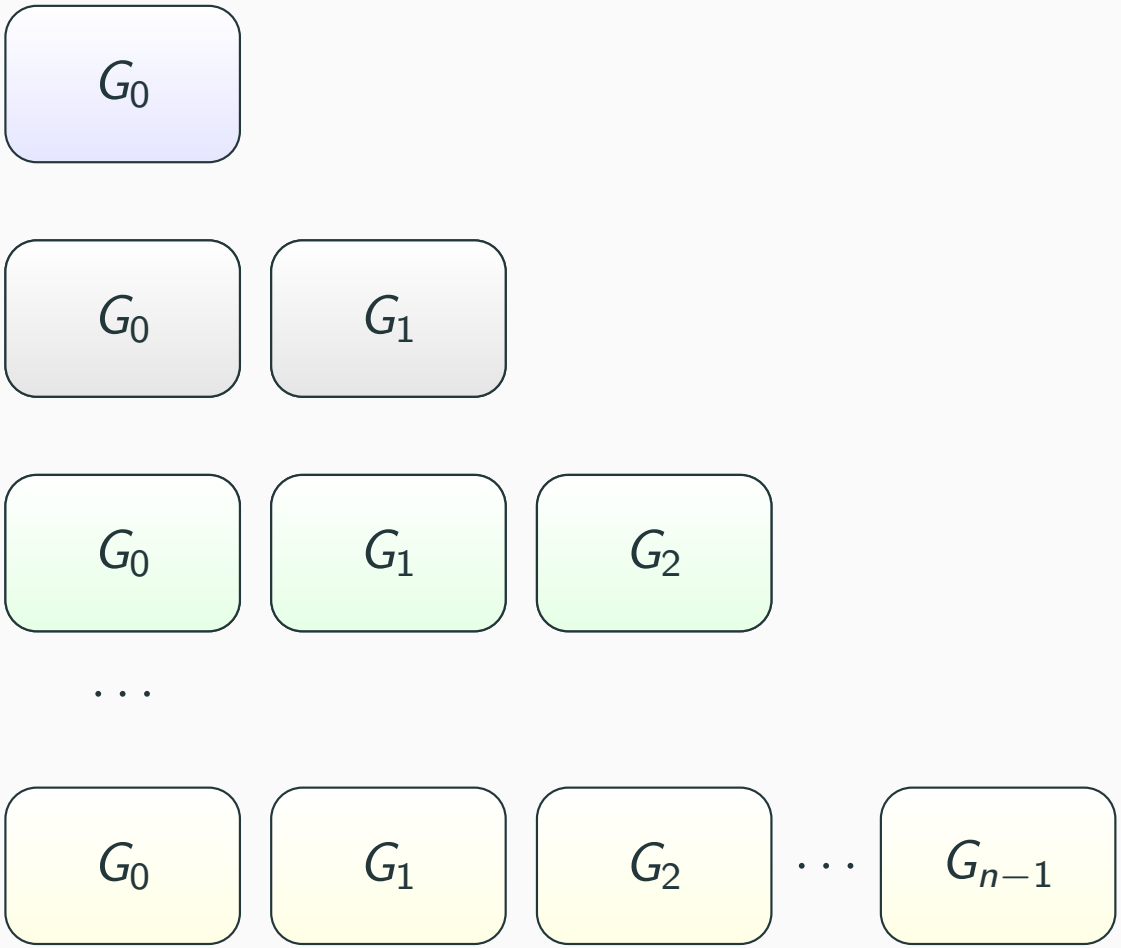
# Graphically ($n$ private keys)...



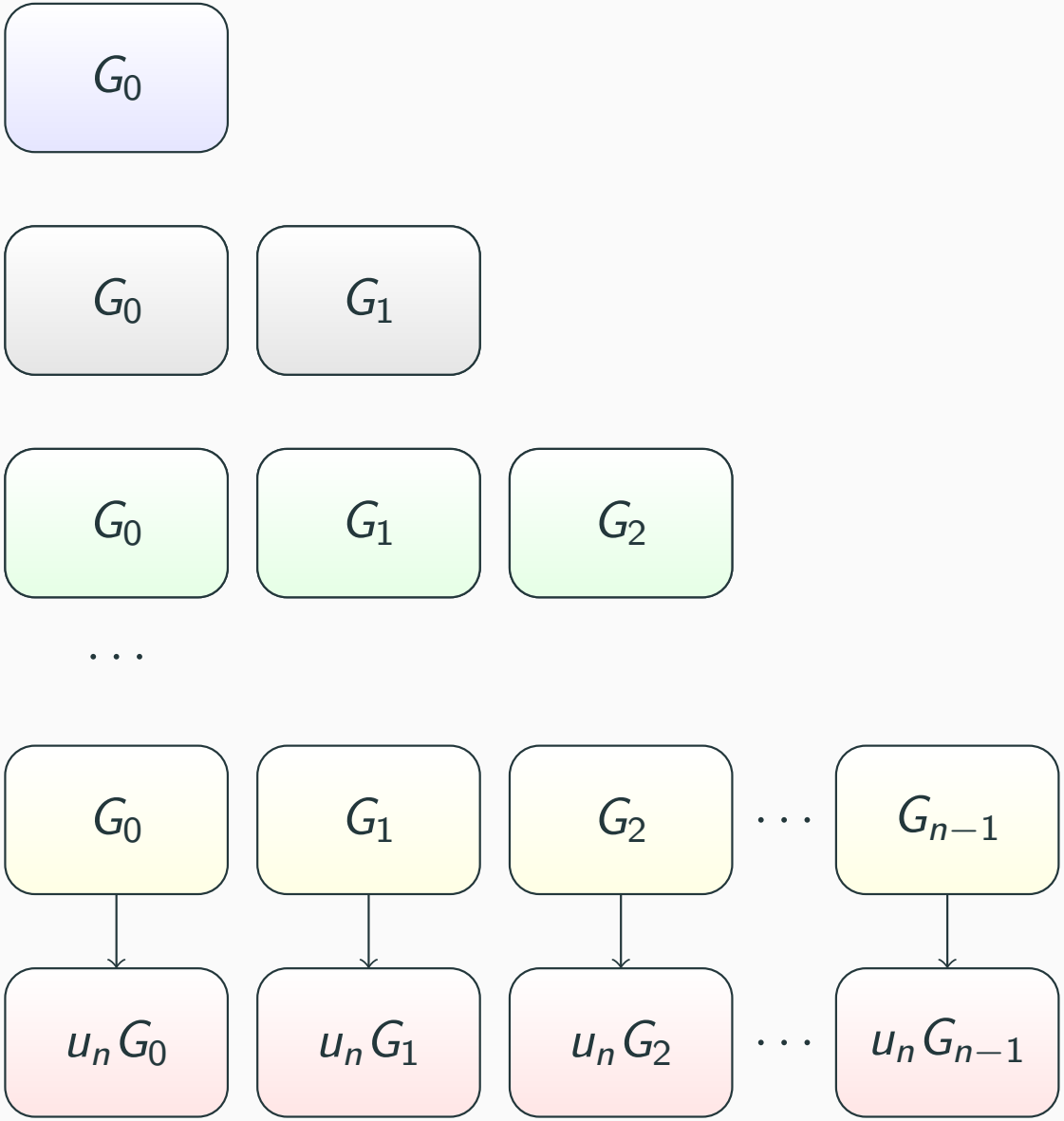
# Graphically ($n$ private keys)...



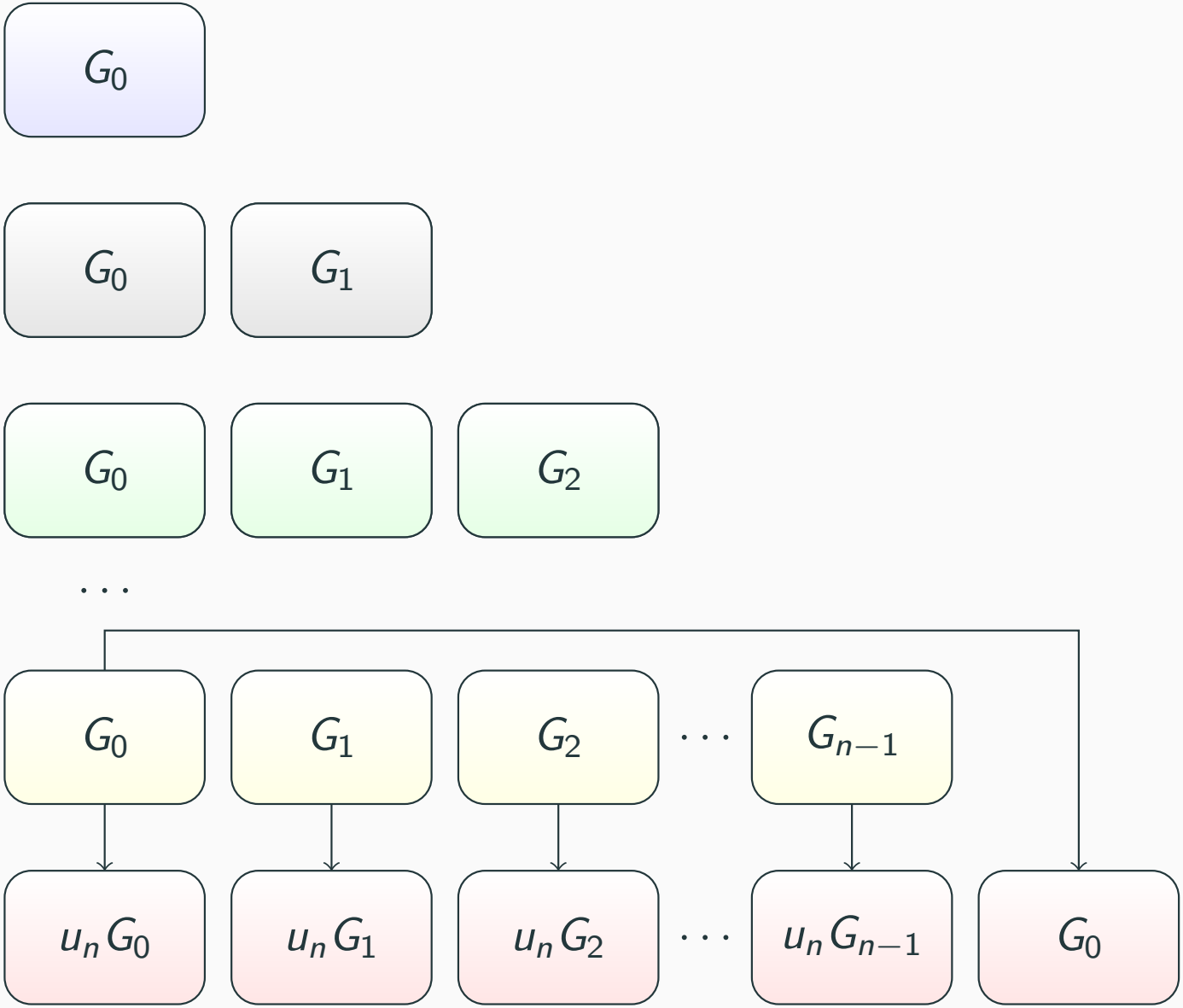
# Graphically ($n$ private keys)...



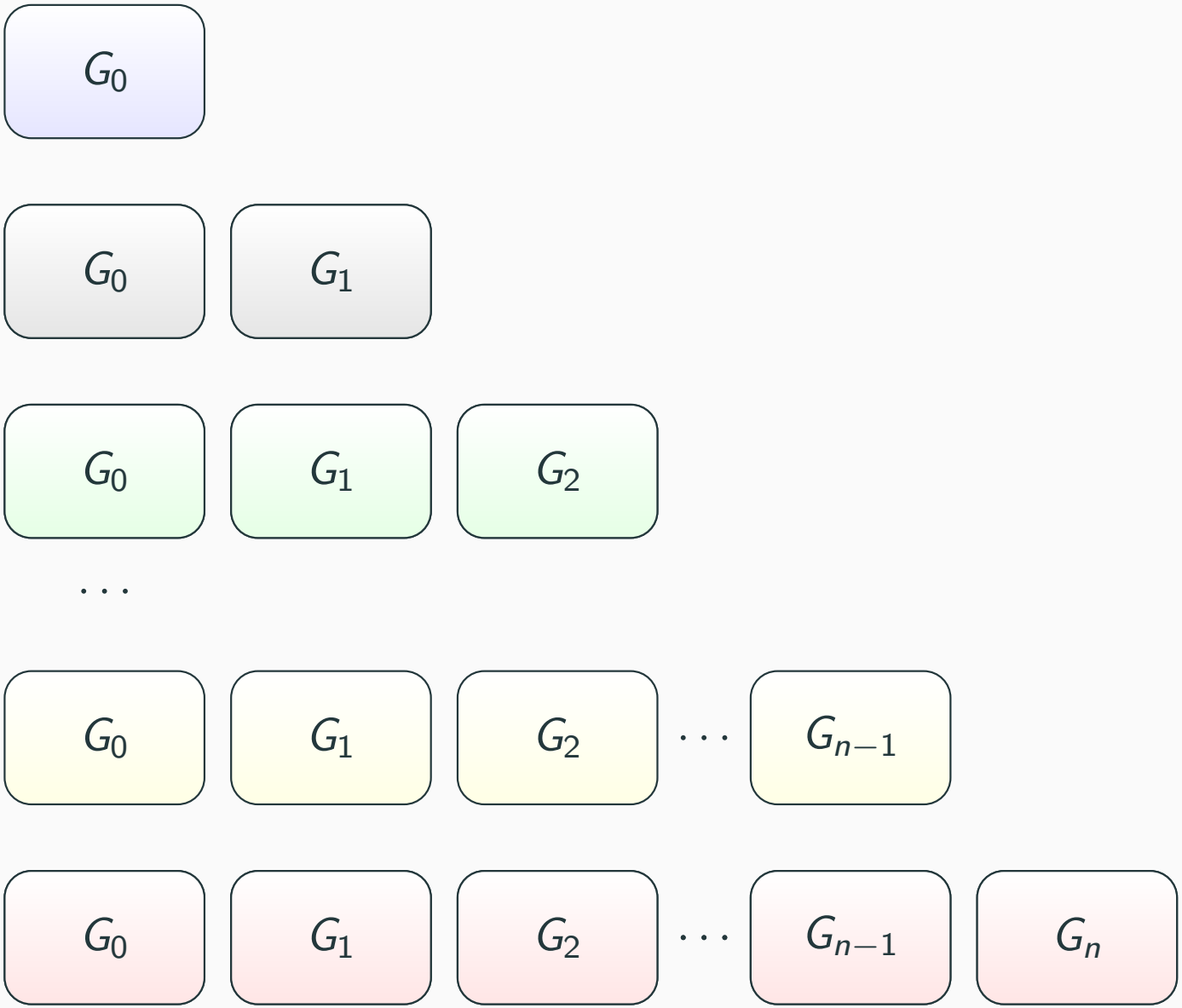
# Graphically ($n$ private keys)...



# Graphically ($n$ private keys)...



# Graphically ($n$ private keys)...



# Proofs of consistency

The updated accumulator $\mathbf{a}'$ is accompanied by a proof that

$$\log_{G_0} G'_0 = \log_{G_1} G'_1 = \ldots = \log_{G_m} G'_m.$$

Such a proof is a well-known ZKP tool.

It is constructed and verified in the algorithms **BAcc-DH.PrvAdd** and **BAcc-DH.VfyAdd**.

# Deriving public keys

After adding the private keys $u_1, u_2, \ldots, u_n$, the resulting accumulator is the word $G_0 G_1 \ldots G_n$ in which

$$G_0 = UG, \quad G_i = \frac{U}{u_i} G, \quad i = 1, 2, \ldots, n, \quad U = \prod_i u_i.$$

**BAcc-DH.Der**

1. Takes an accumulator $\mathbf{a} = G_0 G_1 \ldots G_n$ and a private key $u \in \mathbb{Z}_q^*$.
2. Finds $i \in \{1, 2, \ldots, n\}$ such that $uG_i = G_0$. If such $i$ does not exist, outputs $\perp$.
3. Outputs $V = uG_0$.

# Deriving public keys

After adding the private keys $u_1, u_2, \ldots, u_n$, the resulting accumulator is the word $G_0 G_1 \ldots G_n$ in which

$$G_0 = UG, \quad G_i = \frac{U}{u_i} G, \quad i = 1, 2, \ldots, n, \quad U = \prod_i u_i.$$

**BAcc-DH.Der**

1. Takes an accumulator $\mathbf{a} = G_0 G_1 \ldots G_n$ and a private key $u \in \mathbb{Z}_q^*$.
2. Finds $i \in \{1, 2, \ldots, n\}$ such that $uG_i = G_0$. If such $i$ does not exist, outputs $\perp$.
3. Outputs $V = uG_0$.

**Note**. The pair $(u, V)$ can be used in the ElGamal and Schnorr signature systems (**BAcc**-friendly signatures!).

# Proofs of membership

An owner of $u$ proves that

$$u = \log_{G_i} G_0 = \log_{G_0} V$$

This is the proof of knowledge of two equal discrete logarithms.

To hide $i$, the proof is concealed in the OR-composition

$$\bigvee_{j=1}^{n} \left[ \log_{G_0} V = \log_{G_j} G_0 \right].$$

Such a composition is a well-known ZKP tool.

It is used in the algorithms **BAcc-DH.PrvDer** and **BAcc-DH.VfyDer**.

# Complexity

## Memory

With $n$ voters, the proposed implementation requires storing $O(n^2)$ elements of $\mathbb{G}_q$ and $O(n)$ scalars of $\mathbb{Z}_q$ as final and intermediate accumulators and associated proofs.

## Time

Validating the correctness of all proofs requires $O(n^2)$ scalar multiplications in $\mathbb{G}_q$.

# Complexity

**Memory**

With $n$ voters, the proposed implementation requires storing $O(n^2)$ elements of $\mathbb{G}_q$ and $O(n)$ scalars of $\mathbb{Z}_q$ as final and intermediate accumulators and associated proofs.

**Time**

Validating the correctness of all proofs requires $O(n^2)$ scalar multiplications in $\mathbb{G}_q$.

The time and memory requirements are not burdensome with $n$ of several thousands. However, if $n$ is much greater, other implementations should be considered. One of the promising directions here is the division of voters into small random groups that separately run PKG. Once the grouping-then-PKG round is complete, voters use derived pseudonymous public keys in the second round, and then in several more rounds achieving full pseudonymization.

# Security

We justify the security of **BAcc-DH** examining 4 security requirements: **consistency**, **soundness**, **blindness**, **unlinkability**.

We mainly apply well-known ZKP techniques related to $\Sigma$-protocols.

To deal with **unlinkability**, we use the Square Decisional Diffie-Hellman (SDDH) problem.

**SDDH**

*Input*: $(G, uG, vG)$, where $u, v \in \mathbb{Z}_q^*$.

*Output*: $b \in \{0, 1\}$: $b = 1$ if $v \equiv u^2 \pmod{q}$ and $b = 0$ otherwise.

We show that the unlinkability is ensured if SDDH is hard.

# Further details

https://eprint.iacr.org/2022/373