PloneVote

A proposal for verifiable intra-institutional elections over Plone

Lázaro Clapp
Sergio Rajsbaum
Universidad Nacional Autónoma de México (UNAM)

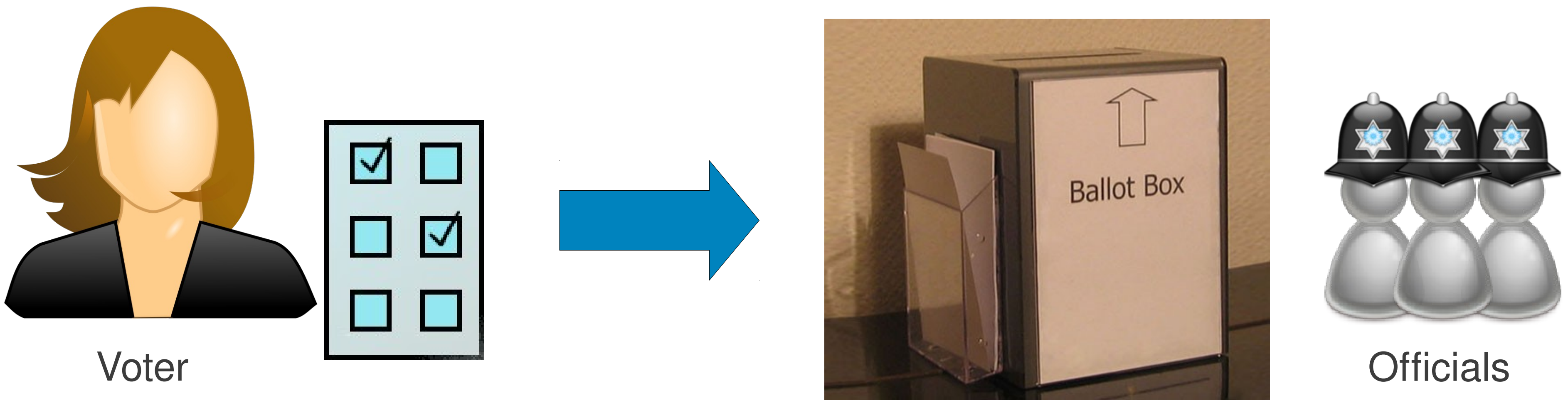
Plone Symposium East 2011

The case of the Institute of Mathematics (UNAM)

The Institute of Mathematics holds elections for:

- Director
- Internal committees
- Other internal positions

Until fairly recently, elections were done the old fashioned way:



What's the problem with that?

- Paper ballots.

What's the problem with that?

- Paper ballots.
- Must vote from **specific places** at **specific times**.

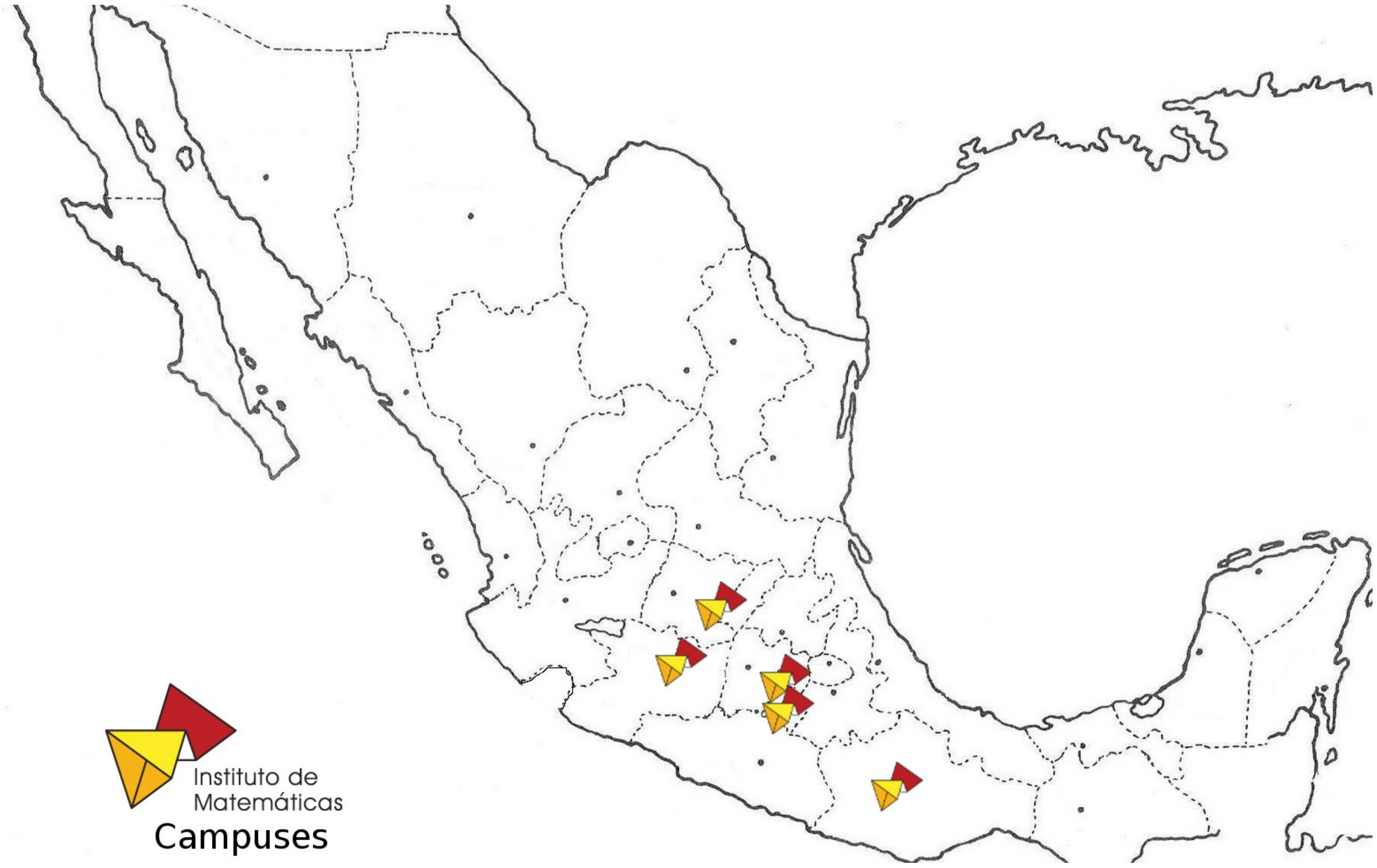
What's the problem with that?

- Paper ballots.
- Must vote from **specific places** at **specific times**.
- Cost of keeping watch over the polling stations and ballot boxes.

What's the problem with that?

- Paper ballots.
- Must vote from **specific places** at **specific times**.
- Cost of keeping watch over the polling stations and ballot boxes.
- Votes might need to be moved from multiple voting locations to a central place for counting.

What's the problem with that?



Enter Plone!

So we want to do online elections... **why over Plone?**

Enter Plone!

So we want to do online elections... **why over Plone?**

- Plone is already used for a lot of the information infrastructure at IMATE.

Enter Plone!

So we want to do online elections... **why over Plone?**

- Plone is already used for a lot of the information infrastructure at IMATE.
- Voter authentication.

Enter Plone!

So we want to do online elections... **why over Plone?**

- Plone is already used for a lot of the information infrastructure at IMATE.
- Voter authentication.
- Notifications: published election information, email.

Enter Plone!

So we want to do online elections... **why over Plone?**

- Plone is already used for a lot of the information infrastructure at IMATE.
- Voter authentication.
- Notifications: published election information, email.
- Voter and candidate list selections based on metadata (e.g. Faculty/Staff Directory)

Enter Plone!

So we want to do online elections... **why over Plone?**

- Plone is already used for a lot of the information infrastructure at IMATE.
- Voter authentication.
- Notifications: published election information, email.
- Voter and candidate list selections based on metadata (e.g. Faculty/Staff Directory)

```
Example:
  "Only tenured faculty as of X months previous to the
  election, who are not members of another committee and
  were not members of this committee for the past Y years
  may be candidates."
```

Not just IMATE

Multiple organizations can benefit from online elections:

- Universities
- Companies
- Online communities

Not just IMATE

Multiple organizations can benefit from online elections:

- Universities
- Companies
- Online communities

Voting on:

- People
- Issues or policies

Not just IMATE

Multiple organizations can benefit from online elections:

- Universities
- Companies
- Online communities

Voting on:

- People
- Issues or policies

We want an open source, general, extensible election system.

Election means security

- An election is **not a poll**.
- Voters require guarantees that their votes are **correctly counted** and **private**.
- **No** single **trusted party**

Security goals

- **Correctness**:
  The votes are recorded and counted as cast by the voters.

- **Privacy**:
  No one but the voter can know how she voted.

Outline

- I. Online elections
- II. Previous work
- III. The PloneVote system
- IV. PloneVote internals
- V. Conclusions

The cautionary tale of ATVotaciones

Alexander Zapata, **Iván Cervantes**.
ATVotaciones, 2008-2009

- Developed at IMATE.
- Based on: **Kiniry et al**. *KOA Remote Voting System*. 2006.
- Integrated with Plone 2 & 3.
- Uses Faculty/Staff metadata.
- Uses Plone authentication.
- Intended to be a secure election system...

The cautionary tale of ATVotaciones

Alexander Zapata, **Iván Cervantes**.
ATVotaciones, 2008-2009

- Developed at IMATE.
- Based on: **Kiniry et al**. *KOA Remote Voting System*. 2006.
- Integrated with Plone 2 & 3.
- Uses Faculty/Staff metadata.
- Uses Plone authentication.
- Intended to be a secure election system...

but fails to fulfill our security requirements without a trusted party.

The cautionary tale of ATVotaciones

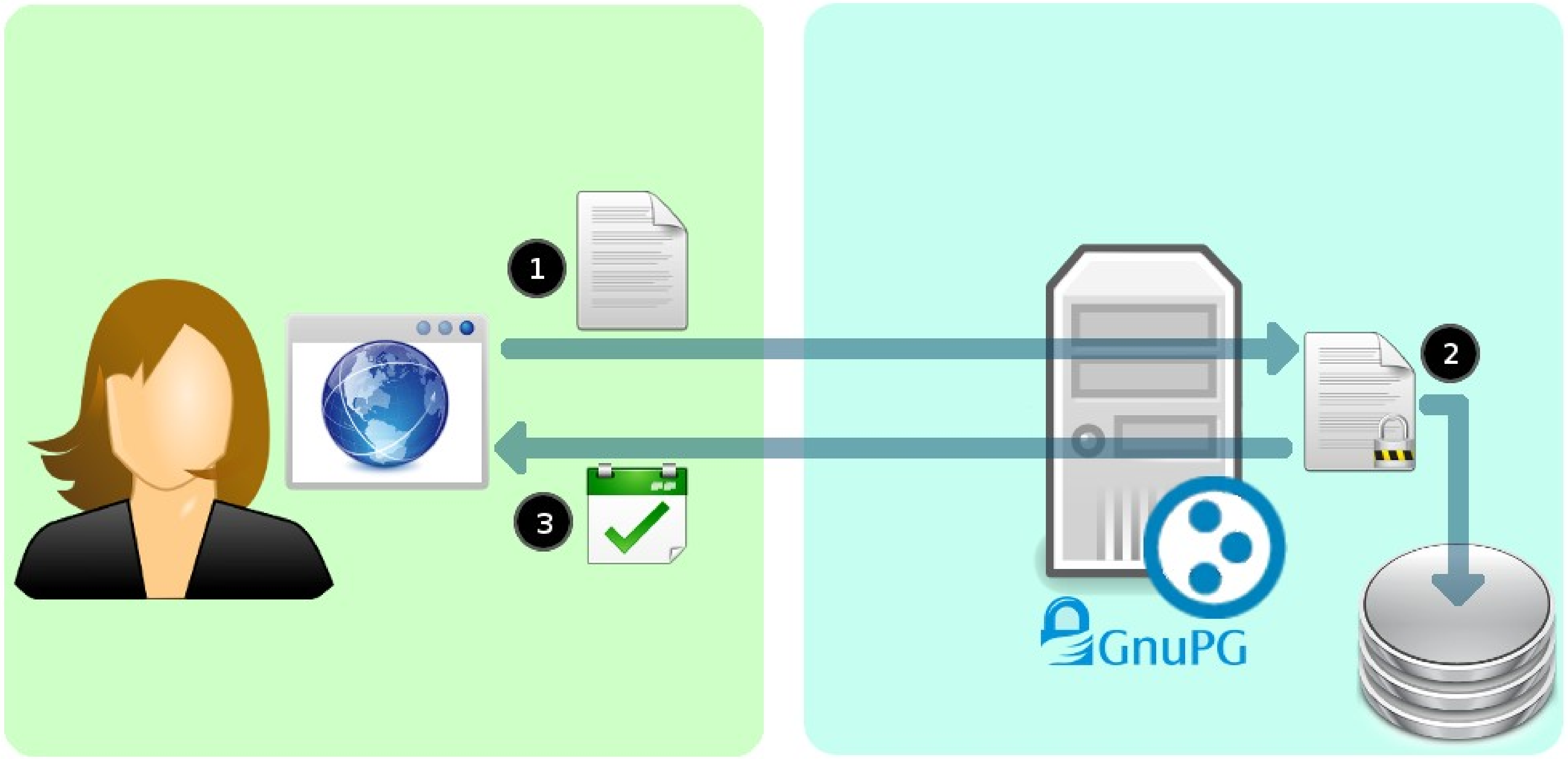
Alexander Zapata, **Iván Cervantes**.
ATVotaciones, 2008-2009

- Developed at IMATE.
- Based on: **Kiniry et al**. *KOA Remote Voting System*. 2006.
- Integrated with Plone 2 & 3.
- Uses Faculty/Staff metadata.
- Uses Plone authentication.
- Intended to be a secure election system...

but fails to fulfill our security requirements without a trusted party.

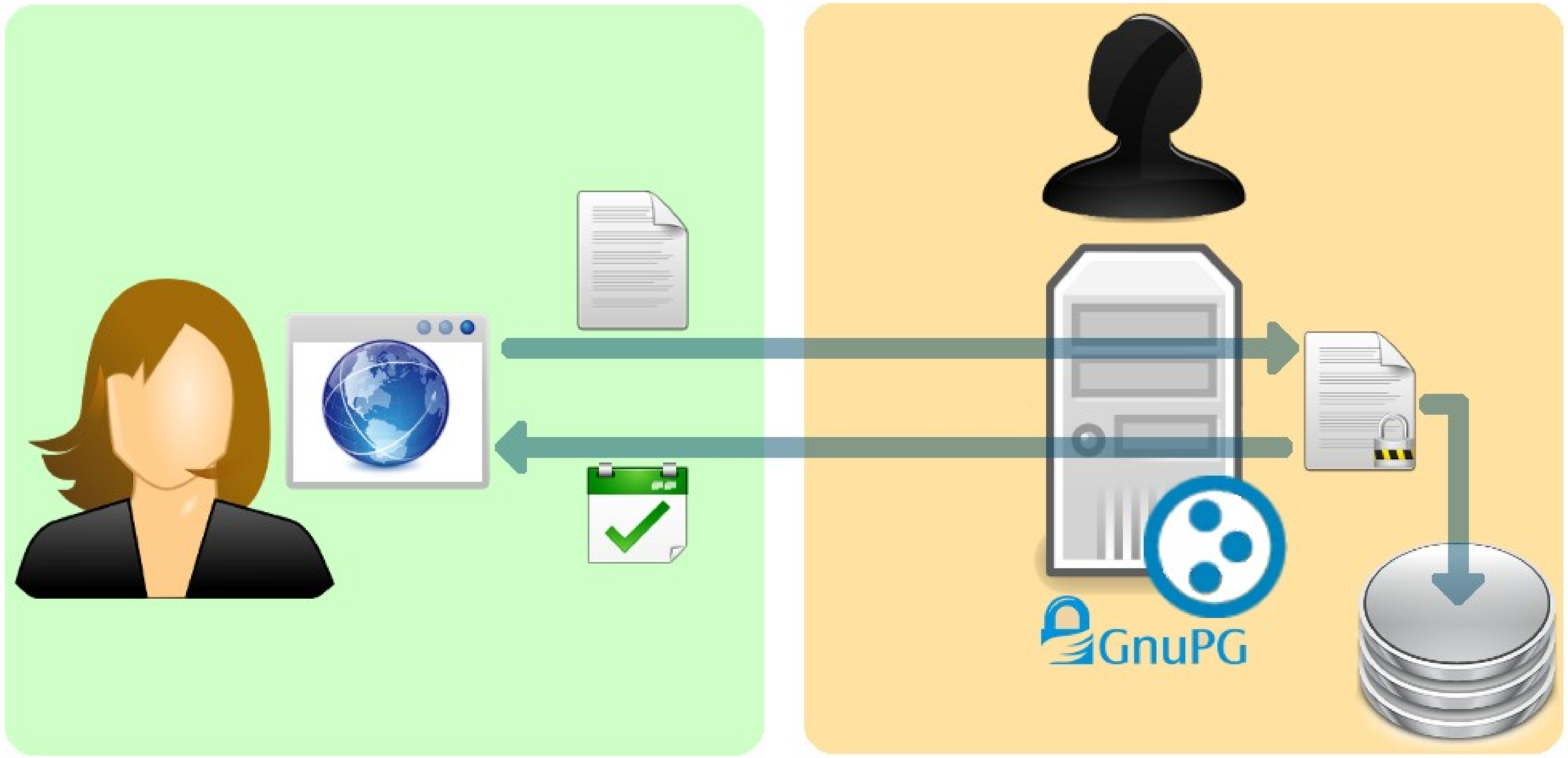
Getting security in online elections right is hard!

The cautionary tale of ATVotaciones



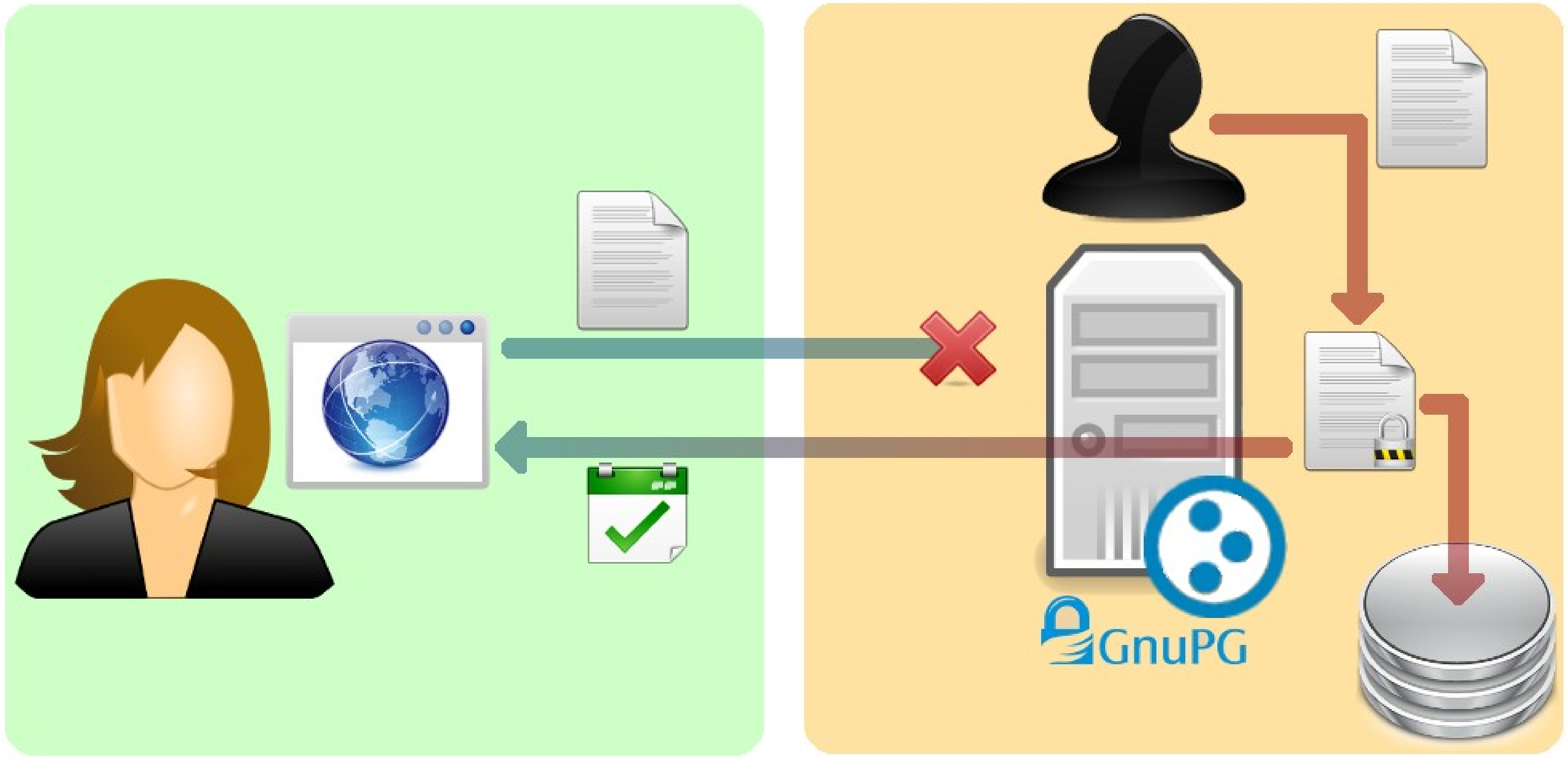
1. User casts vote in plain text on server.
2. Server encrypts and stores vote.
3. Server emits receipt.

The cautionary tale of ATVotaciones



Server Compromised: **No Privacy**

The cautionary tale of ATVotaciones



Server Compromised: **No Correctness**

Previous academic work

There is a lot of academic work on secure online elections:

- **Clarkson, Chong, Myers**.
  “Civitas: Toward a Secure Voting System”. IEEE SP 2007.
- **Ben Adida**.
  “Helios: Web-based Open-Audit Voting”. USENIX 2008.
- **Josh Benaloh**.
  “Simple Verifiable Elections”. Microsoft Research 2006.

… and many more.

Previous academic work

- These systems use **specialized cryptographic primitives**.

Previous academic work

- These systems use **specialized cryptographic primitives**.

- These systems use **complex protocols** and **code** running **on different domains**.

Previous academic work

- These systems use **specialized cryptographic primitives**.
- These systems use **complex protocols** and **code** running **on different domains**.
- These systems **work for our purposes** (they provide correctness and privacy without a trusteed party).

Civitas

Clarkson, Chong, Myers.
"Civitas: Toward a Secure Voting System". IEEE SP 2007.

- ✓ Working system
- ✓ First cryptographically secure online election system.
- ✓ Secure, scalable.
- ✗ Fairly complex.
- ✗ Written in Java.
- ✗ Not CMS-integrated.

Helios

Ben Adida.
“Helios: Web-based Open-Audit Voting”. USENIX 2008.

- ✓ Working system
- ✓ Secure, scalable.
- ✓ Simpler to use and set-up than Civitas.
- ✓ Written in python and JavaScript.
- ✗ Not CMS-integrated.
- ✗ Uses homomorphic encryption:
    - We cannot change the way votes are represented or counted without altering the security protocol.

SVE

Josh Benaloh.

“Simple Verifiable Elections”. Microsoft Research 2006.

- ✗ Academic paper.
  - Does not include a working implementation.
- ✗ Mechanical vote capturing.
- ✓ “Easy” to understand secure vote counting protocol.
- ✓ Vote representation independent.

Outline

- I. Online elections
- II. Previous work
- III. The PloneVote system
- IV. PloneVote internals
- V. Conclusions

PloneVote

PloneVote

What PloneVote aims to be...

PloneVote

What PloneVote aims to be...

- **First ever secure election system within a general purpose CMS**.

PloneVote

What PloneVote aims to be...

- **First ever secure election system within a general purpose CMS**.

- Designed from the start to be **Plone integrated**.

PloneVote

What PloneVote aims to be...

- **First ever secure election system within a general purpose CMS**.
- Designed from the start to be **Plone integrated**.
- **Based on state of the art academic work** (mostly SVE & Helios).

What PloneVote aims to be...

- **First ever secure election system within a general purpose CMS**.
- Designed from the start to be **Plone integrated**.
- **Based on state of the art academic work** (mostly SVE & Helios).
- Supporting multiple vote representation and vote counting schemes.

What PloneVote aims to be...

• **First ever secure election system within a general purpose CMS**.

• Designed from the start to be **Plone integrated**.

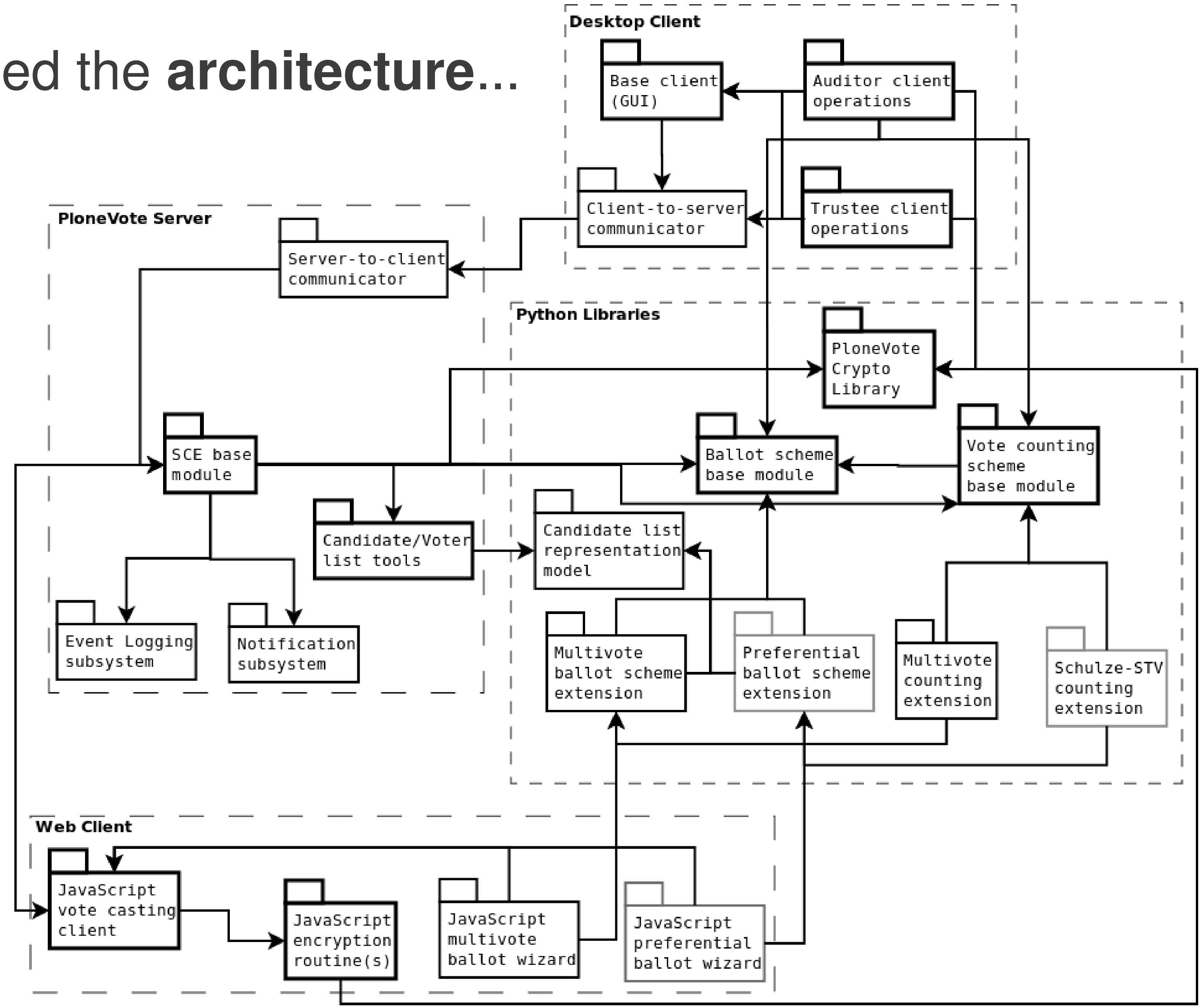
• **Based on state of the art academic work** (mostly SVE & Helios).

• Supporting multiple vote representation and vote counting schemes.

• Allowing **voting** directly **from any modern browser**.

Where are we?

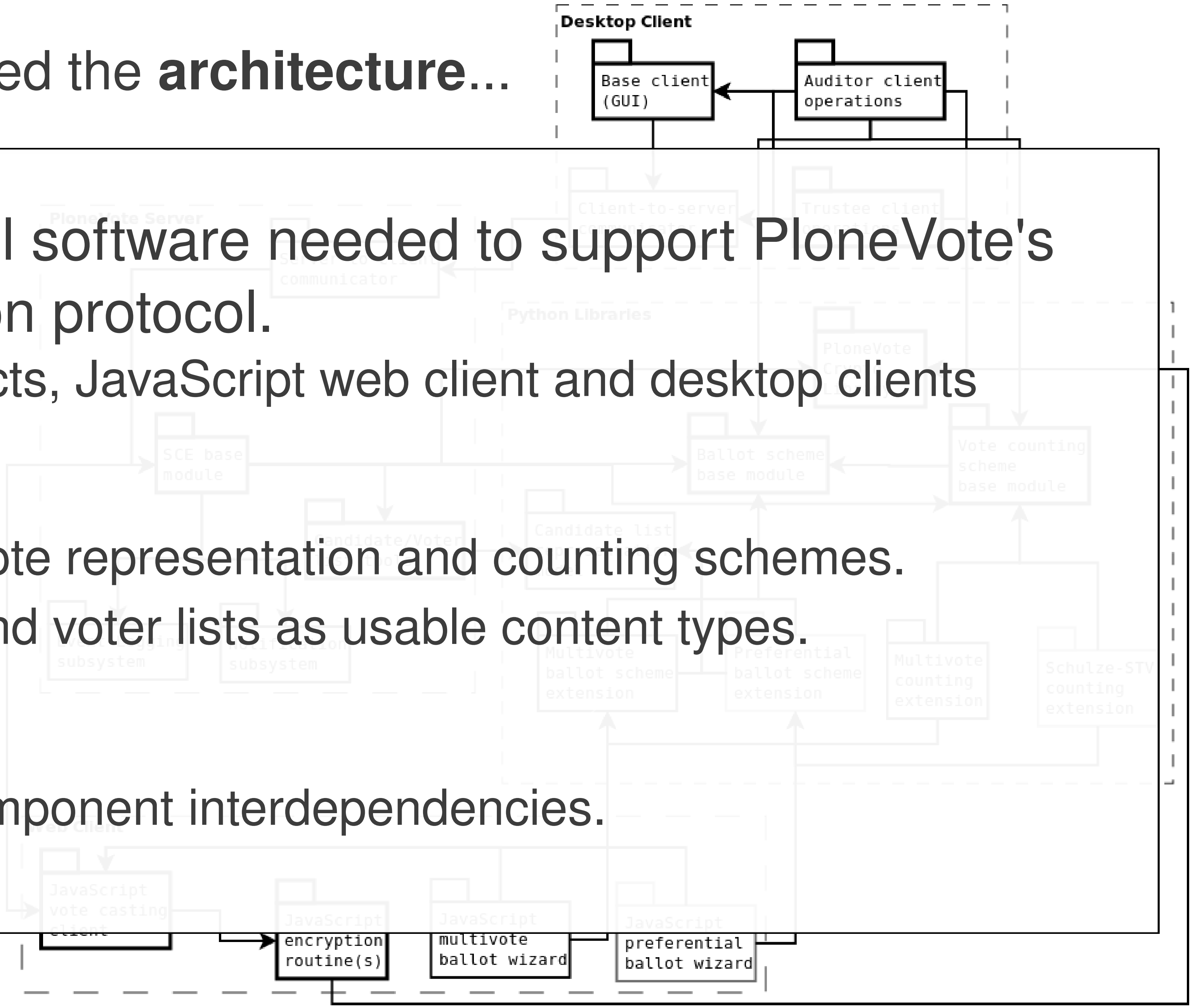
We have designed the **architecture**...



Desktop Client
Base client (GUI)
Auditor client operations
Client-to-server communicator
Trustee client operations
PloneVote Server
Server-to-client communicator
SCE base module
Candidate/Voter list tools
Event Logging subsystem
Notification subsystem
Python Libraries
PloneVote Crypto Library
Ballot scheme base module
Vote counting scheme base module
Candidate list representation model
Multivote ballot scheme extension
Preferential ballot scheme extension
Multivote counting extension
Schulze-STV counting extension
Web Client
JavaScript vote casting client
JavaScript encryption routine(s)
JavaScript multivote ballot wizard
JavaScript preferential ballot wizard

Where are we?

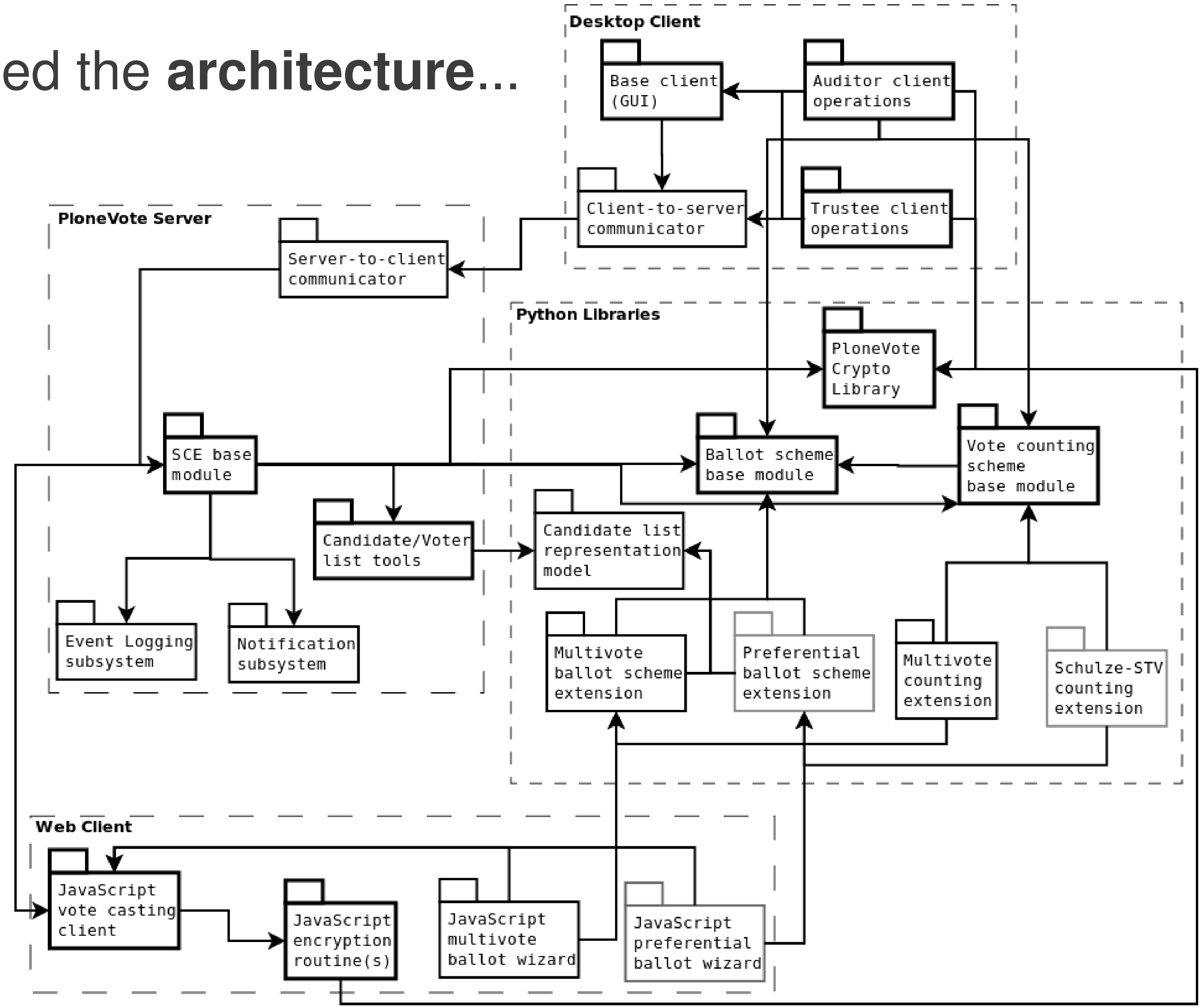
We have designed the **architecture**...



- Describes all software needed to support PloneVote's secure election protocol.
  - Plone products, JavaScript web client and desktop clients
- Modular
  - Extensible vote representation and counting schemes.
  - Candidate and voter lists as usable content types.
- Layered
  - No cyclic component interdependencies.

Where are we?

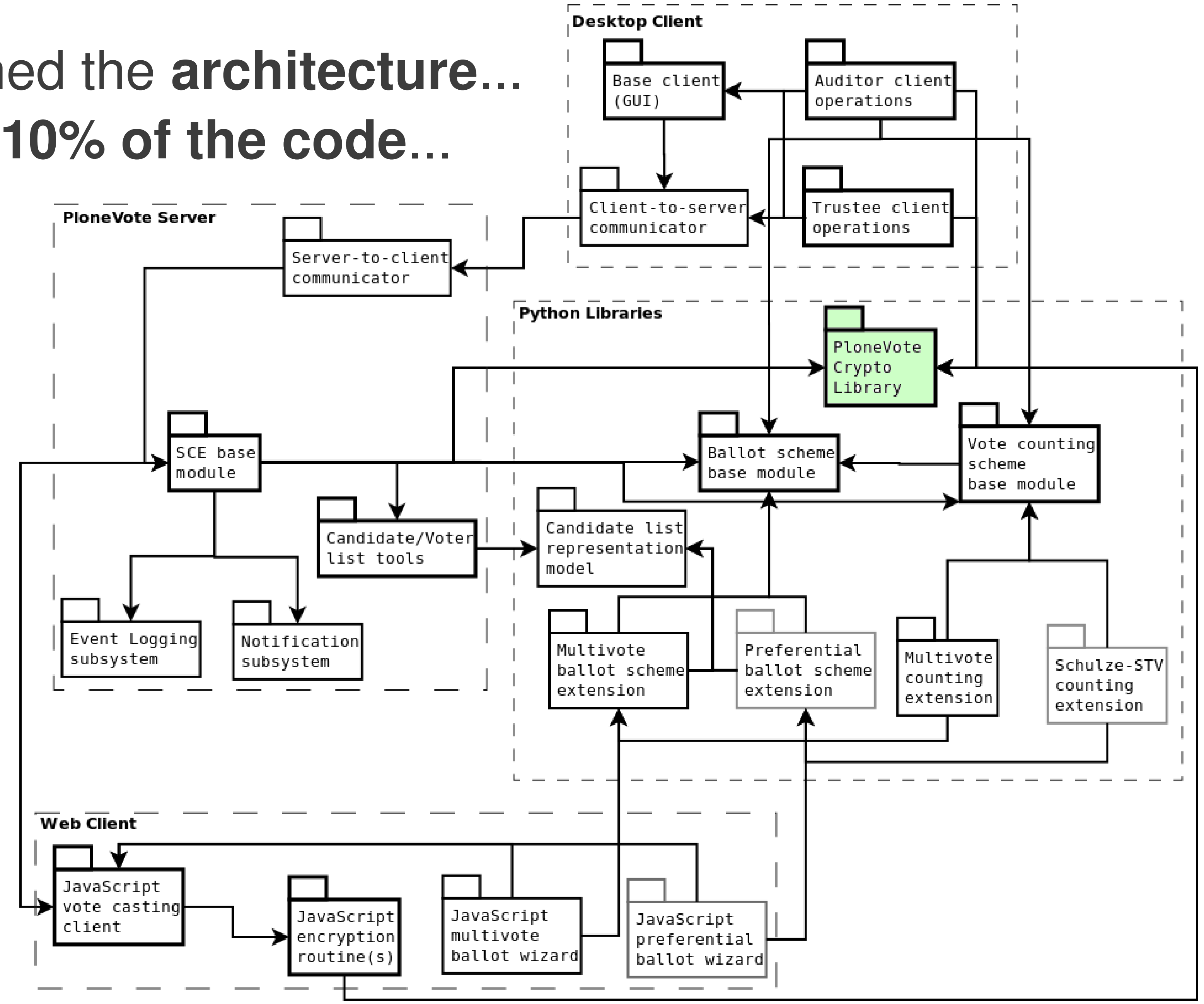
We have designed the **architecture**...


Desktop Client
Base client (GUI)
Auditor client operations
Client-to-server communicator
Trustee client operations
PloneVote Server
Server-to-client communicator
SCE base module
Candidate/Voter list tools
Event Logging subsystem
Notification subsystem
Python Libraries
PloneVote Crypto Library
Ballot scheme base module
Vote counting scheme base module
Candidate list representation model
Multivote ballot scheme extension
Preferential ballot scheme extension
Multivote counting extension
Schulze-STV counting extension
Web Client
JavaScript vote casting client
JavaScript encryption routine(s)
JavaScript multivote ballot wizard
JavaScript preferential ballot wizard

Where are we?

We have designed the **architecture**...

...written about **10% of the code**...



Desktop Client
Base client (GUI)
Auditor client operations
Client-to-server communicator
Trustee client operations
PloneVote Server
Server-to-client communicator
Python Libraries
PloneVote Crypto Library
SCE base module
Ballot scheme base module
Vote counting scheme base module
Candidate/Voter list tools
Candidate list representation model
Event Logging subsystem
Notification subsystem
Multivote ballot scheme extension
Preferential ballot scheme extension
Multivote counting extension
Schulze-STV counting extension
Web Client
JavaScript vote casting client
JavaScript encryption routine(s)
JavaScript multivote ballot wizard
JavaScript preferential ballot wizard

Where are we?

We have designed the **architecture**...
…written about **10% of the code**...

$\mathbb{Z}_p^*$

Pedersen

$$k_{di} = 2\left(\sum_{j=1}^{n} s_{ji}\ mod\ q\right)$$

$$z = \prod_{i\in\{i_1,\dots,i_k\}} \gamma^{k_{di}\left(\prod_{l\in(i_1,\dots,i_k)\wedge l\neq i}\left(\frac{l}{l-i}\right)\right)}$$

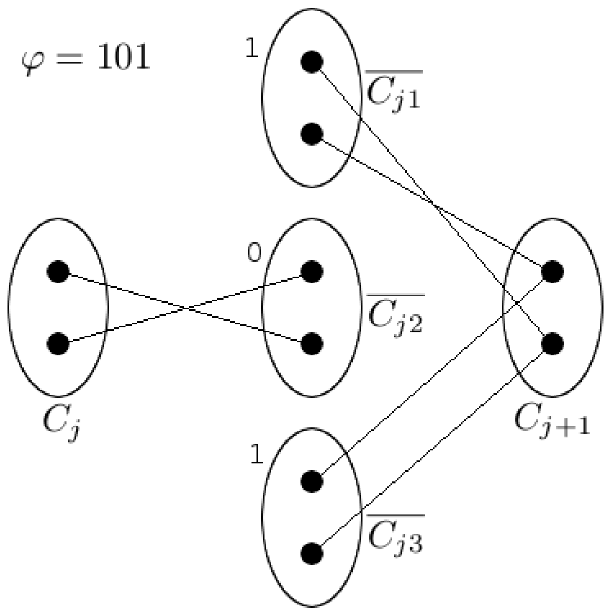
IDA-CPA

$$g^J = g^{t+\varphi r}\ (mod\ p-1)$$

$\varphi = 101$

$$O\left(\sum_{1\leq i\leq k} e_i\left(lg\ n + \sqrt{p_i}\right)\right)$$

Fiat-Shamir heuristic



EGCryptosystem.new()

$$c' = \left(g^{r'}\gamma\ mod\ p, (k_e)^{r'}\delta\ mod\ p\right)$$

get_prime()

$$g^{2s_{ji}} = \prod_{l=0}^{k-1}(C_{jl})^{2i^l}\ mod\ p$$

$$v_j = \prod_{l=1}^{n} g^{2s_{lj}}\ mod\ p$$

...and implemented **90% of the math** ;)

Where are we?

PloneVoteCryptoLib

- A cryptographic library written in pure python.
- The core of PloneVote.
- Implements all the specialized cryptography required by the PloneVote system:
    - ElGamal encryption
    - Distributed threshold encryption key generation
    - Distributed verifiable threshold decryption
    - Reencryption and verifiable vote mixing
    - Proofs verification (mixing and partial decryption)
- 100% operational.

(This was my bachelor's thesis, advisor: S. Rajsbaum.)

PloneVoteCryptoLib

- **First implementation** of these cryptographic primitives **with a library approach**.

- **First python implementation of mixnets**.

- Designed specifically to support the needs of PloneVote.

PloneVoteCryptoLib

Intuitive API for complex cryptographic operations:

```
vote_collection.shuffle_with_proof()
```

PloneVoteCryptoLib

Intuitive API for complex cryptographic operations:

```
vote_collection.shuffle_with_proof()
```

Versus:

- Perform ElGammal reencryption for each vote.
- Randomly permute the votes (using a secure source of randomness)
- Generate 128 alternate mixes of the votes.
- Generate a 128 bit challenge using the Chaum-Pedersen heuristic.
- Use the challenge and mixing mappings to provide a zero-knowledge proof of ciphertext collection equivalence.
- … and mind your group algebra!

PloneVoteCryptoLib

Intuitive API for complex cryptographic operations:

```
vote_collection.shuffle_with_proof()
```

With:

- Task monitoring services: custom progress bars, etc.
- Simple and detailed configuration:

```
params.SECURITY_LEVEL = \
        params.SECURITY_LEVELS_ENUM.HIGH
```

or

```
params.CUSTOM_{FOO} = my_value
```

- Custom object serialization and included XML serialization.
- And more!

What do we want to do now?

- We want to document and open source what we have.
- We want to continue implementing the full system.
- We want **your help!**

Outline

- I. Online elections
- II. Previous work
- III. The PloneVote system
- IV. PloneVote internals
- V. Conclusions

PloneVote users

There are three user roles in a PloneVote election:

- **Voter**:
  - The person who votes in the election.
- **Trustee**:
  - Members of the election commission.
  - Appointed before the election starts.
  - Oversees the election and **protects voter privacy**.
- **Auditor**:
  - **Verifies election correctness**.
  - Designated auditors selected before the election starts.
  - Any other user can optionally audit the election as well.

Two stages of voting

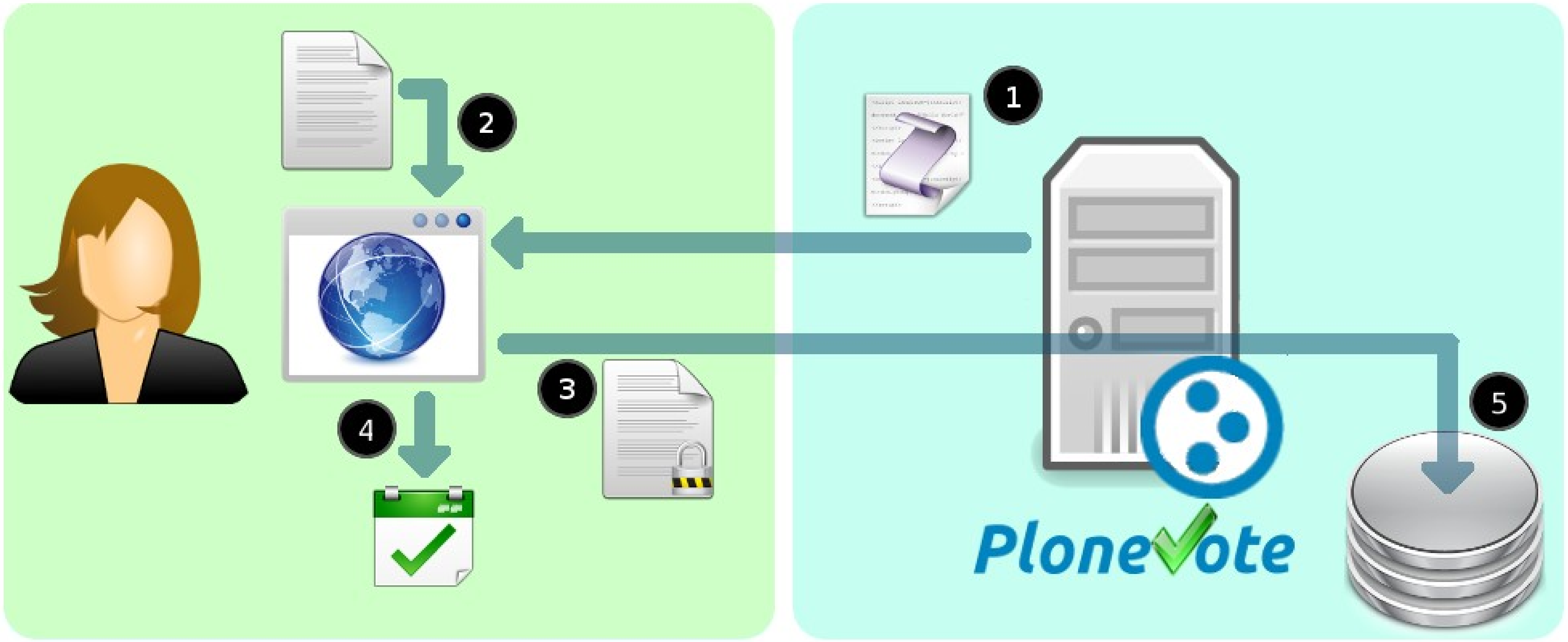
Vote Capture:

Each **voter** casts their own **vote**, which is captured by the server and recorded as part of a **captured ballot set**.

Vote Counting:

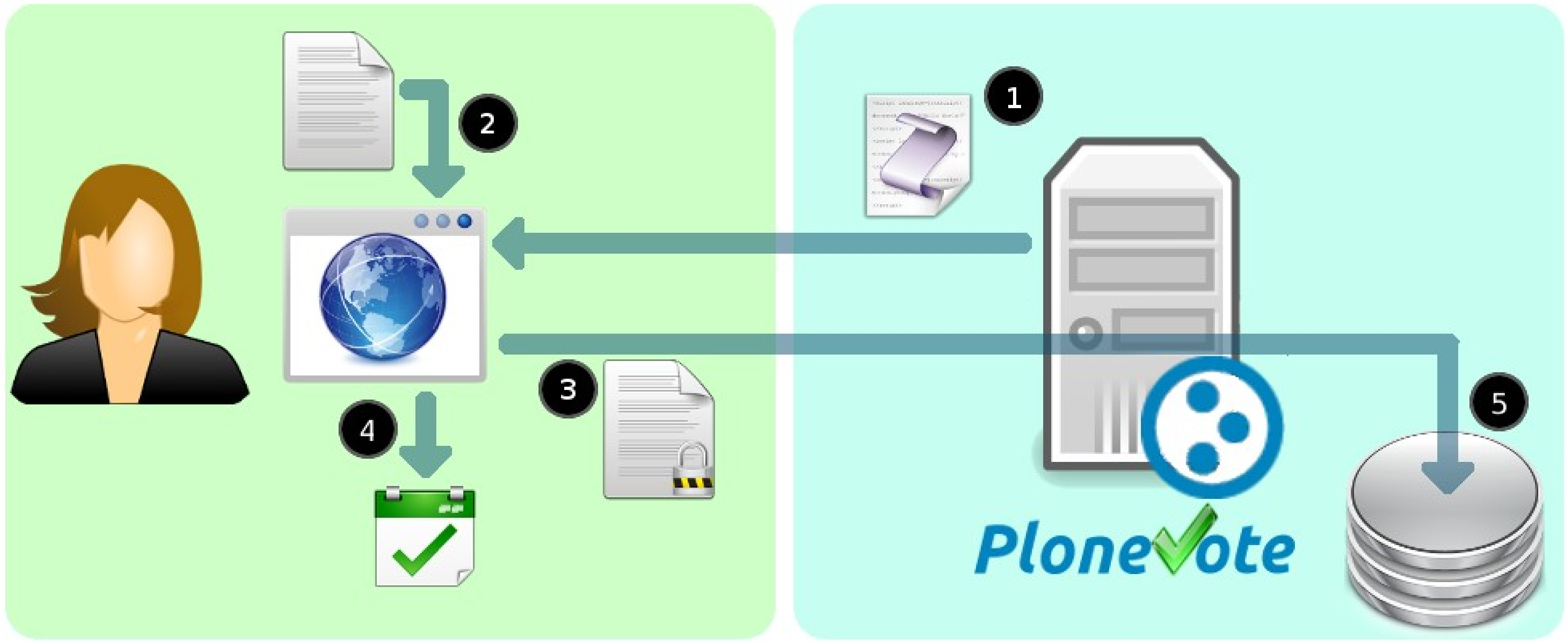
The **trustees** *transform* the votes to ensure privacy, *decrypt* them, *count* them and *publish* the **election results** together with the info required to *verify* them.

Vote capture



1. Server sends JavaScript voting client.
2. Voter captures preferences in JS client.
3. Client encrypts and sends vote to server.
4. Client emits receipt for voter
5. Server stores encrypted vote.

Vote capture



Ben Adida's *Helios* has shown that this is possible.

Vote counting

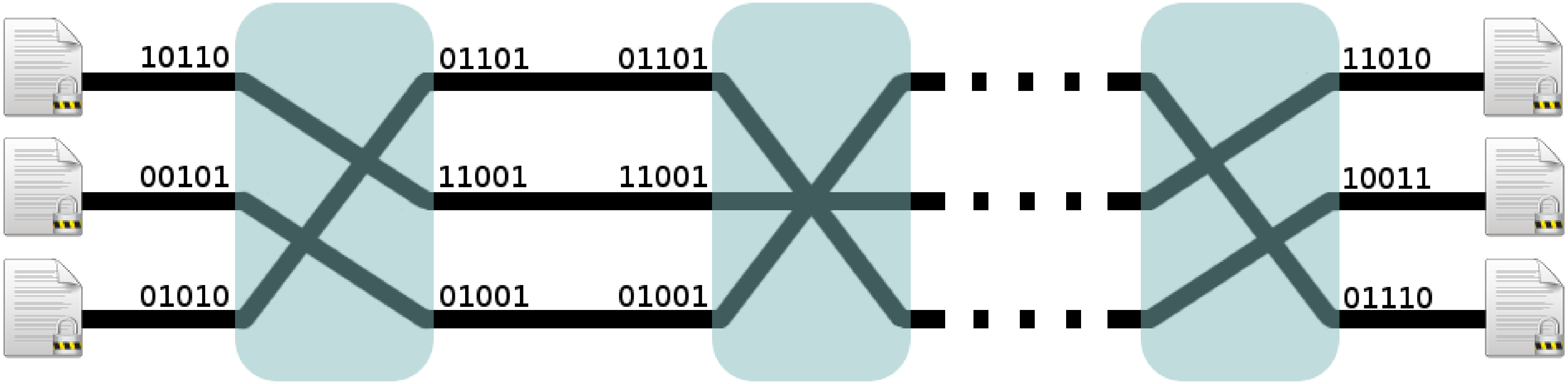
- Based on **Josh Benaloh's** *“Simple Verifiable Elections”*.

- Step 1:
  Each trustee cryptographically shuffles the votes.

- Step 2:
  The trustees cooperate to decrypt the votes.

Vote counting

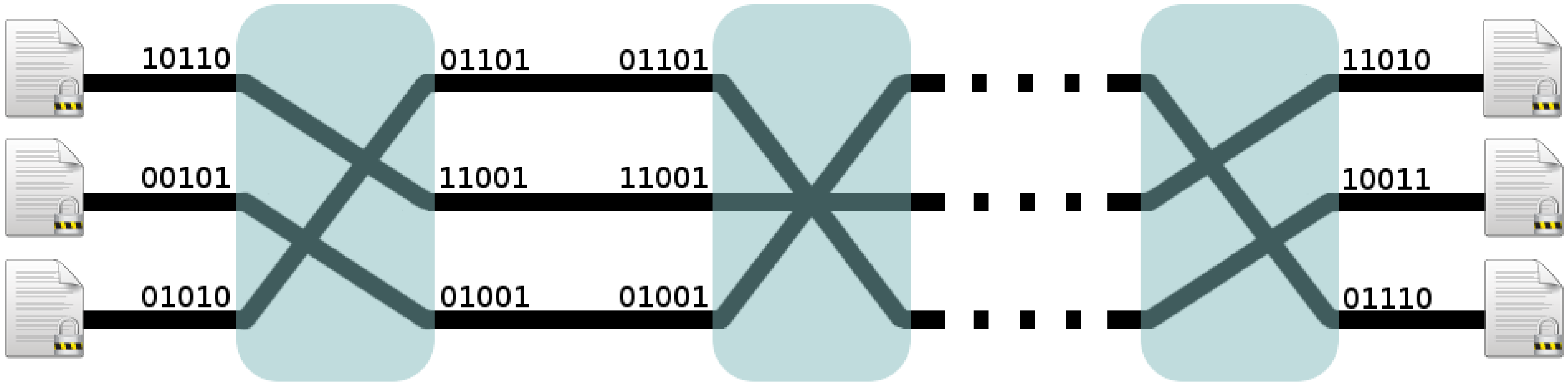
Step 1:
Each trustee cryptographically shuffles the votes



Vote counting

Step 1:
Each trustee cryptographically shuffles the votes

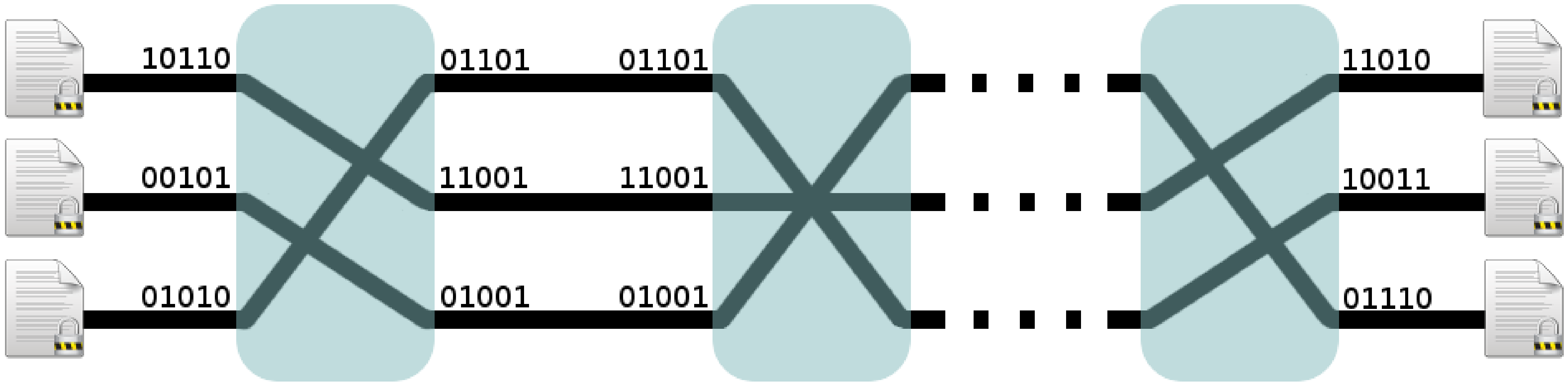
- Same set of votes before and after shuffling.



Vote counting

Step 1:
Each trustee cryptographically shuffles the votes

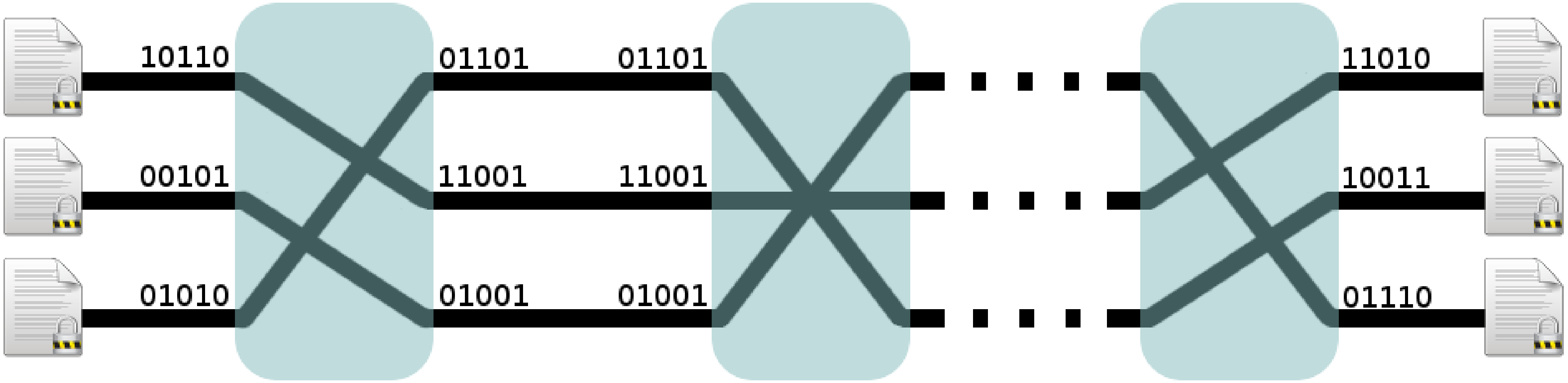
- Same set of votes before and after shuffling.
- It is impossible to determine which vote in the input corresponds to which vote in the output.



Vote counting

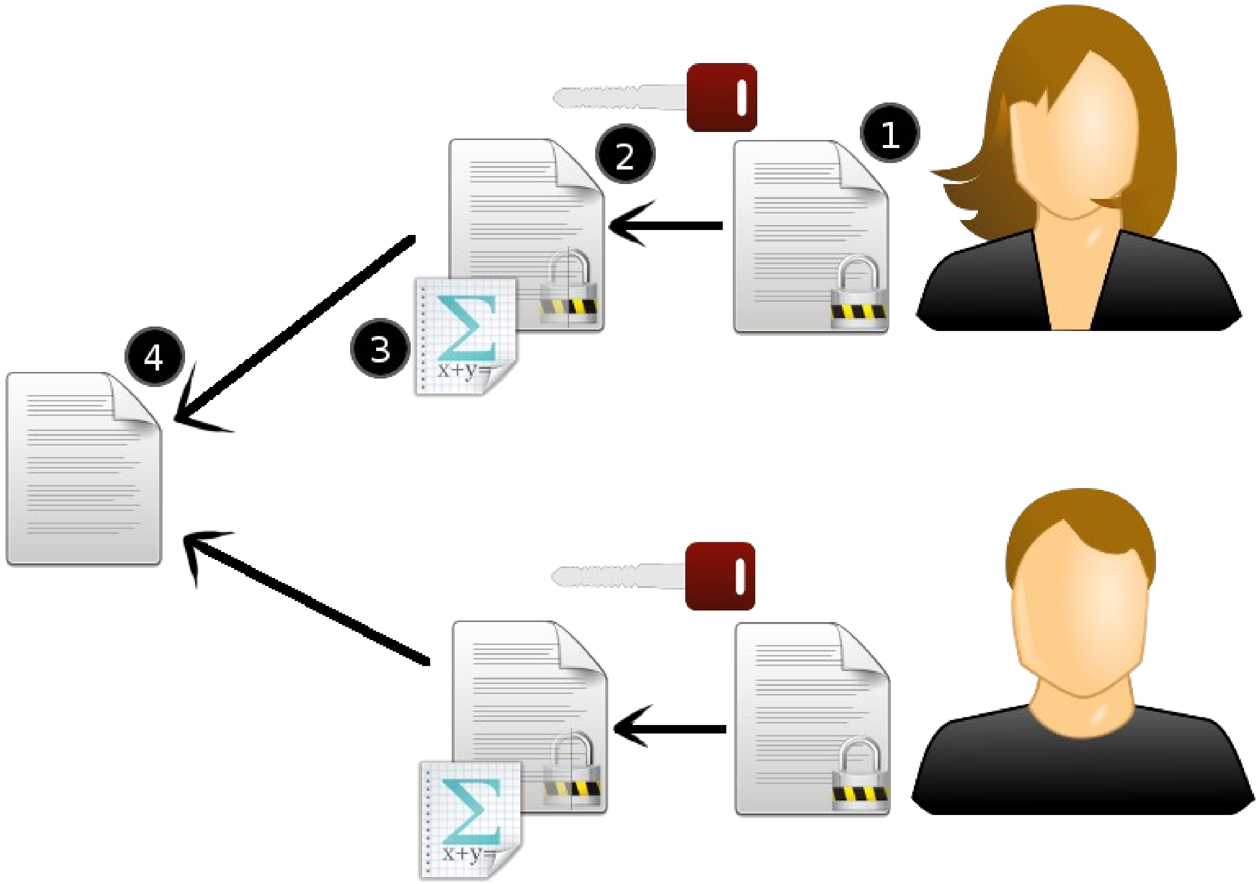
Step 1:
Each trustee cryptographically shuffles the votes

- Same set of votes before and after shuffling.
- It is impossible to determine which vote in the input corresponds to which vote in the output.
- Attached proof of correct shuffling.



Vote counting

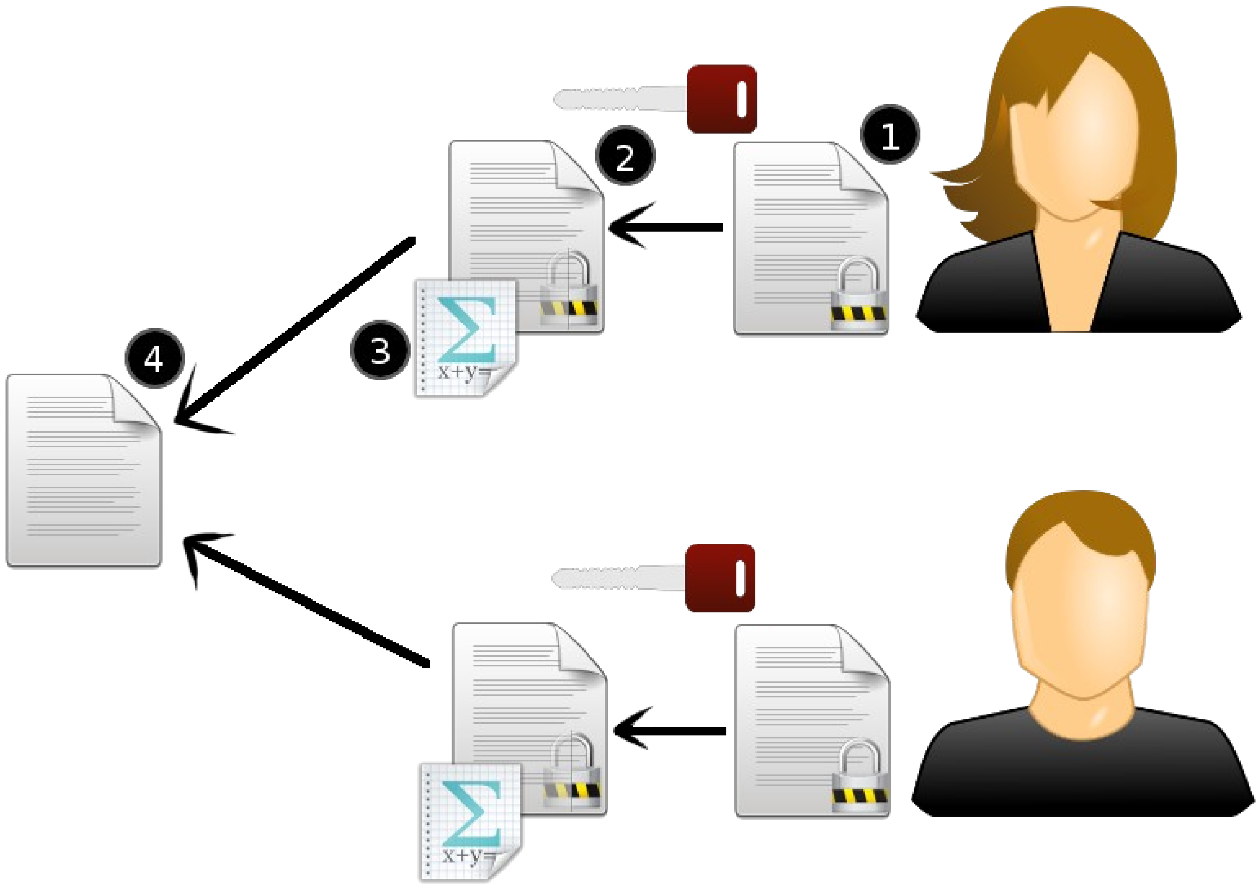
Step 2:
The trustees cooperate to decrypt the votes



Vote counting

Step 2:
The trustees cooperate to decrypt the votes

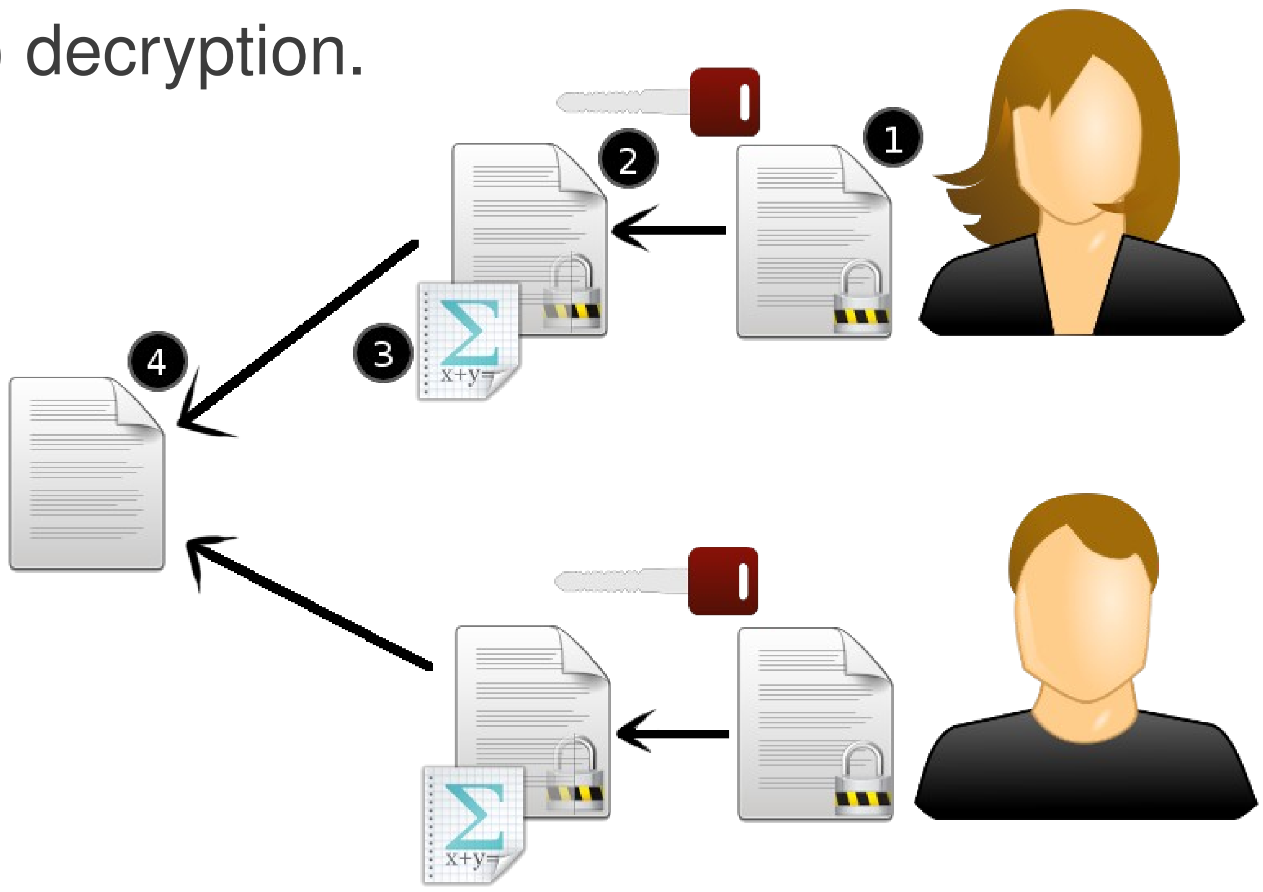
- Only the last shuffled set is decrypted.



Vote counting

Step 2:
The trustees cooperate to decrypt the votes

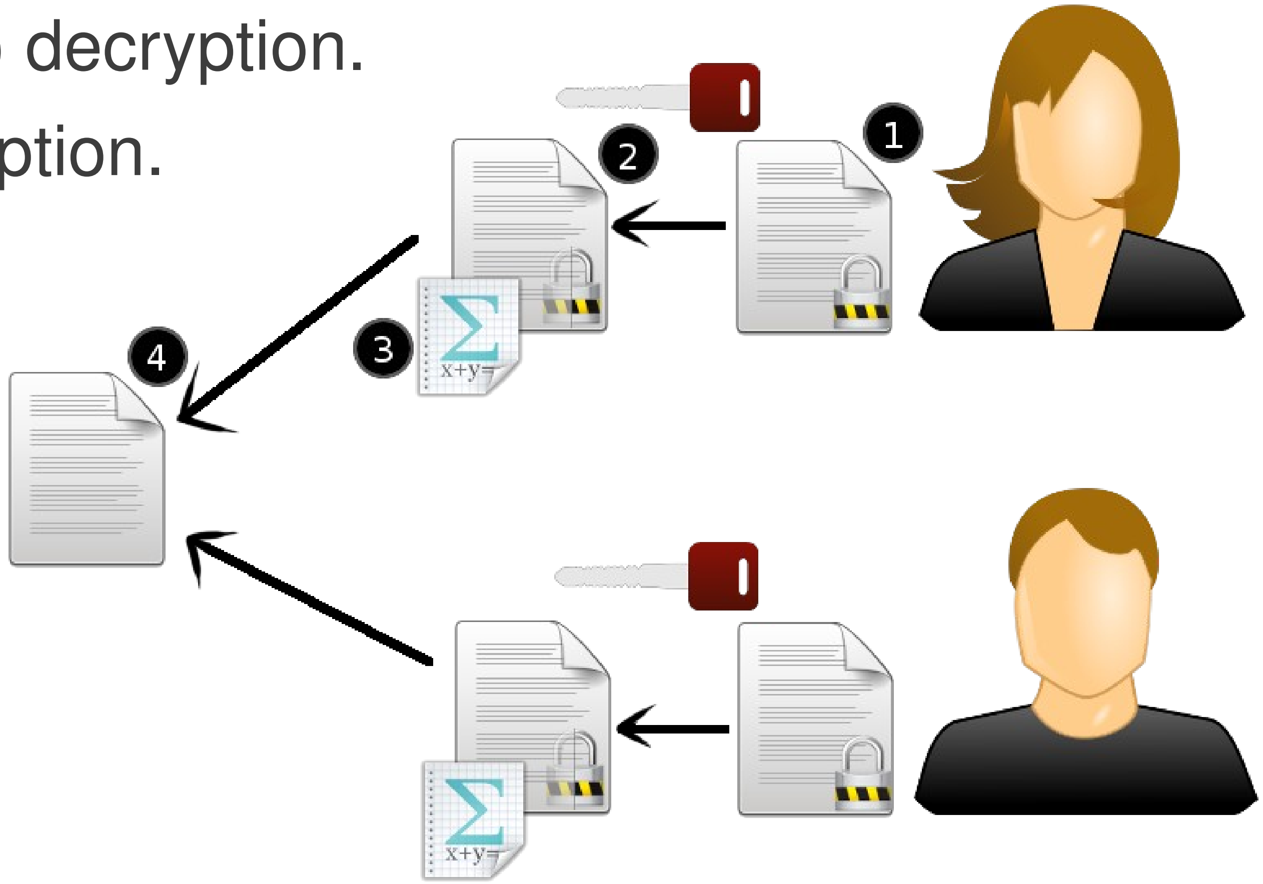
- Only the last shuffled set is decrypted.
- Multi-party (threshold) decryption.



Vote counting

Step 2:
The trustees cooperate to decrypt the votes

- Only the last shuffled set is decrypted.
- Multi-party (threshold) decryption.
- Proof of correct decryption.



Election verification

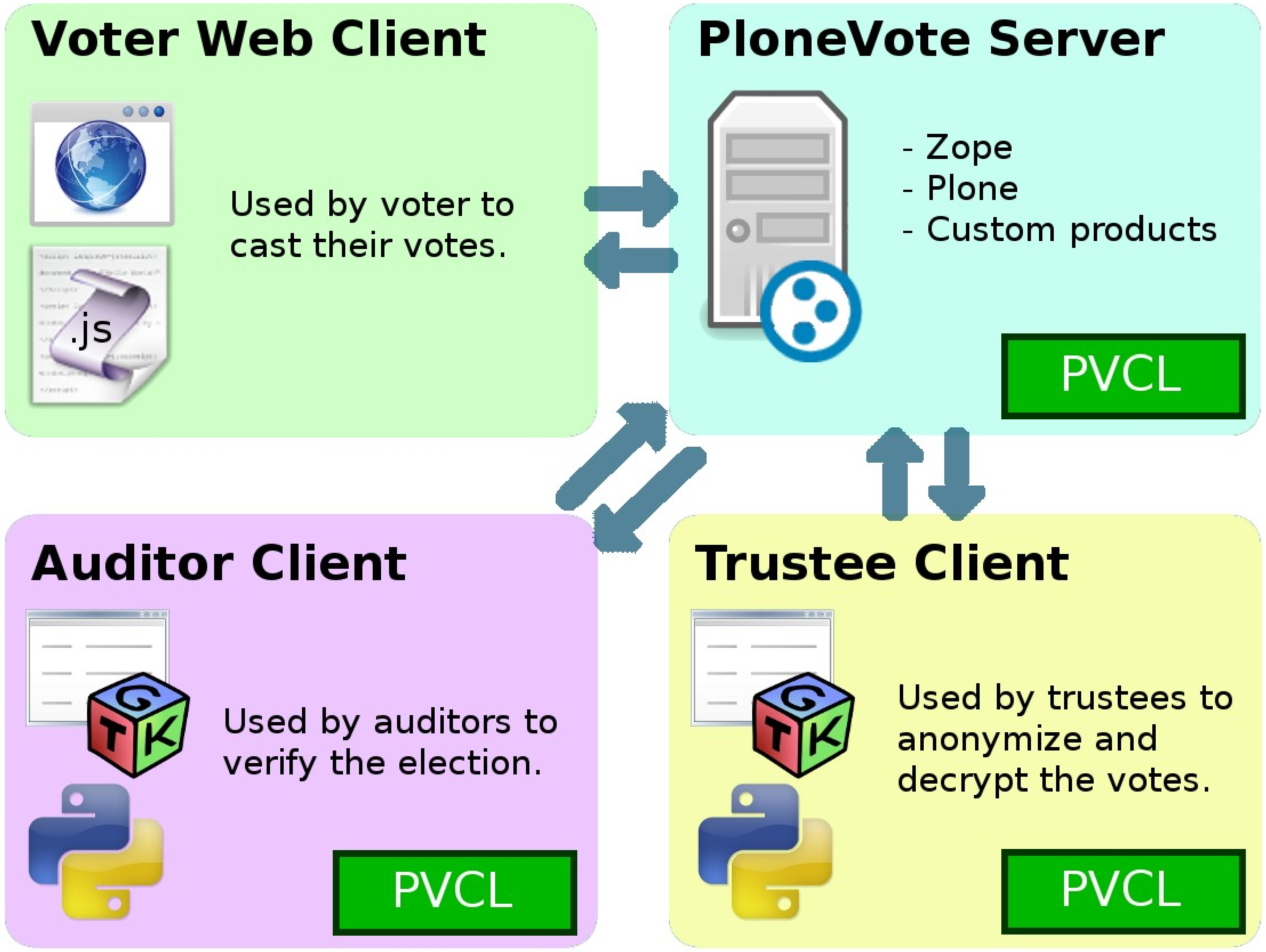
- **Each voter must verify that their own vote was captured.**
  - Checking her receipt.

Election verification

- **Each voter must verify that their own vote was captured.**
  - Checking her receipt.

- **Any auditor can verify that the captured votes were correctly counted.**
  - Checking the proofs of shuffling and decryption.

PloneVote components


Voter Web Client
.js
Used by voter to cast their votes.
PloneVote Server
- Zope
- Plone
- Custom products
PVCL
Auditor Client
Used by auditors to verify the election.
PVCL
Trustee Client
Used by trustees to anonymize and decrypt the votes.
PVCL

Outline

Conclusions

- **Secure online elections** are useful.

Conclusions

- **Secure online elections** are useful.
- There is value in **integrating** secure online elections **with Plone**.

Conclusions

- **Secure online elections** are useful.

- There is value in **integrating** secure online elections **with Plone**.

- **PloneVote** attempts to provide this integration, while supporting **state of the art security**.

Conclusions

- **Secure online elections** are useful.
- There is value in **integrating** secure online elections **with Plone**.
- **PloneVote** attempts to provide this integration, while supporting **state of the art security**.
- We have done some of the work, but there is still much to do... and **we want your help to do it!**

Thank you for your attention

Questions?