Bern University of Applied Sciences

CHVote and OpenCHVote

What went wrong? What went well?

Rolf Haenni, BFH

Cyber Security Winter School, June 15th, 2021

Outline

- Introduction
- Phase 1 & 2: The Geneva System (Pre-CHVote)
- Phase 3: CHVote 1.0 and CHVote 2.0
- Phase 4: CHVote Protocol and OpenCHVote
- Conclusion

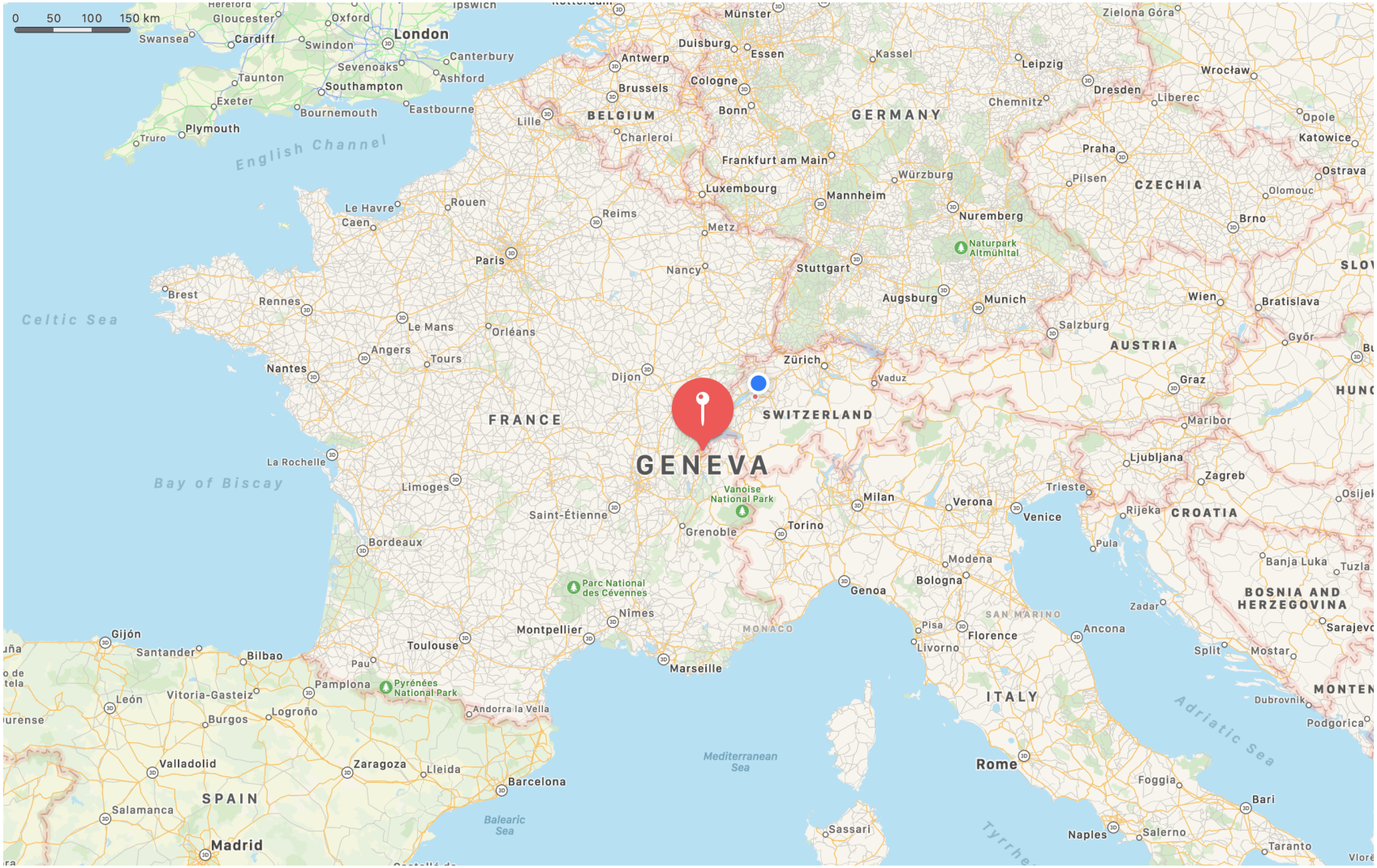
Outline

What does CH stand for?

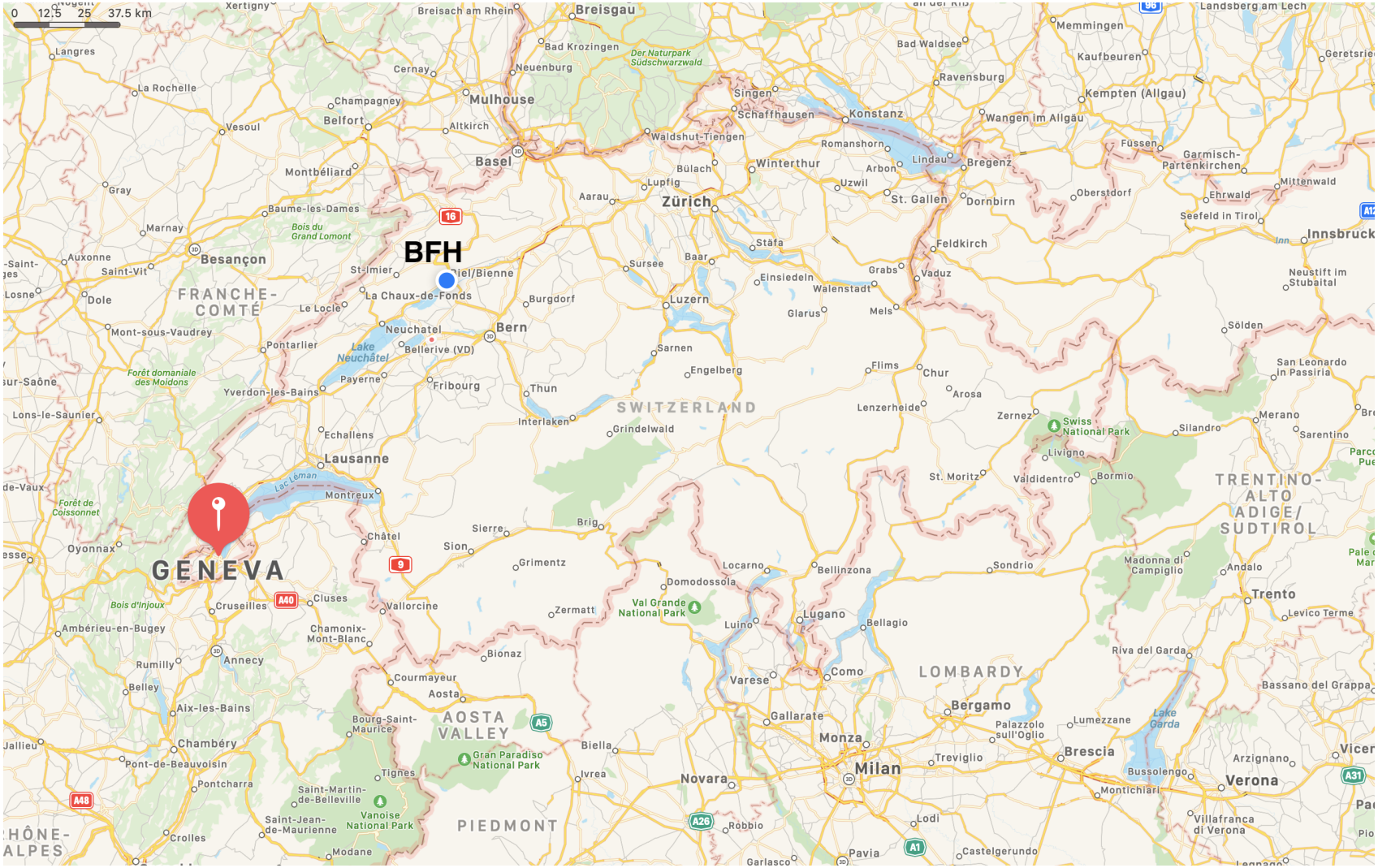
CH = Confederatio Helvetica

(Latin for Swiss Confederation)

Where does CHVote come from?



Where does CHVote come from?



Where does CHVote come from?

Why was CHVote launched?

Reason 1: Swiss direct democracy

- Up to four election days per year
  - Elections
  - Mandatory referendums
  - Optional referendums (>50k signatures)
  - Popular initiatives (>100k signatures)
- Different political levels
  - Federal
  - Cantonal
  - Municipal
- Up to 10 different election topics per election day

Why was CHVote launched?

Reason 2: Swiss citizens living abroad

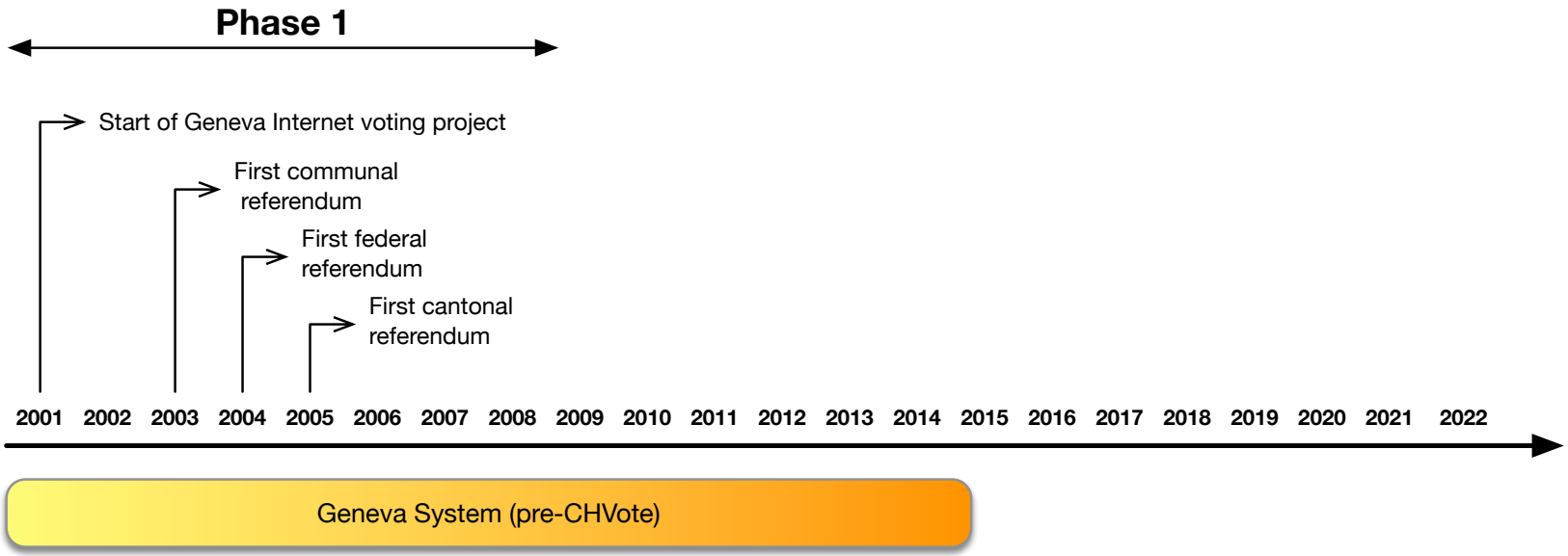
- Population of Switzerland $\approx$ 8.87 millions
- Swiss citizens living in Switzerland $\approx$ 6.66 millions
- Swiss citizens living abroad $\approx$ 0.77 millions (or $\approx$ 10.4%)
- Eligible to vote on cantonal and federal topics
- Politically strong Organisation of the Swiss Abroad (OSA)

Why was CHVote launched?

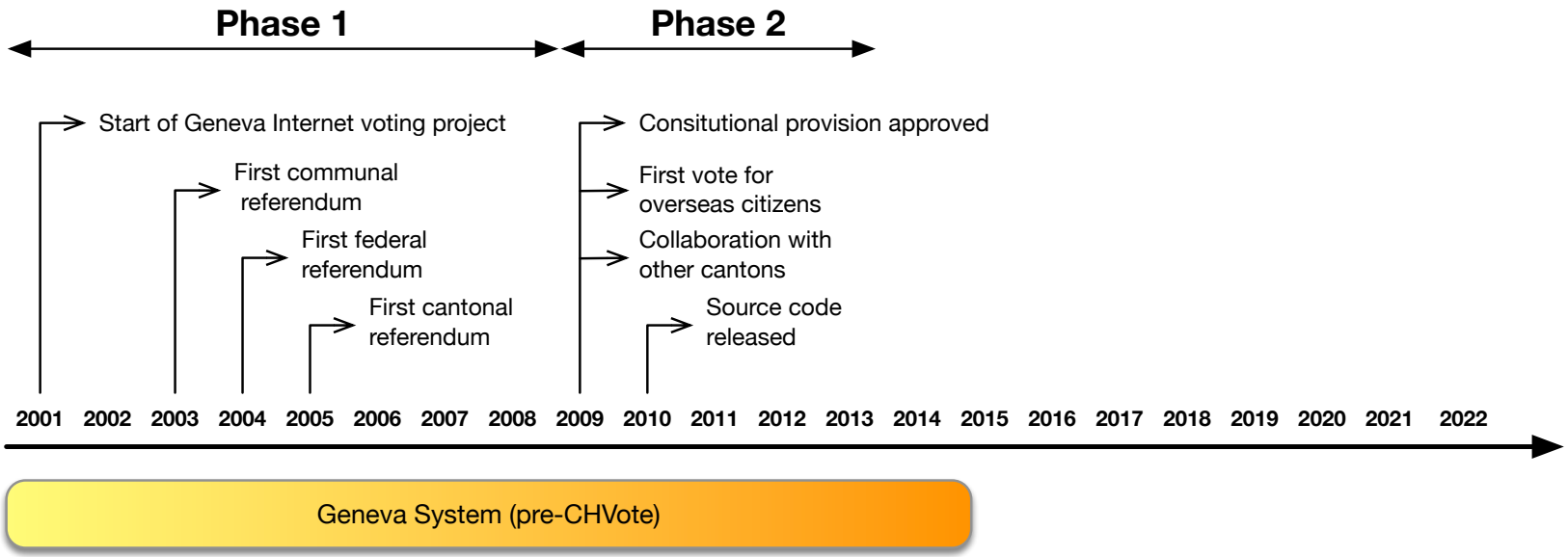
Reason 3: Federal Council

- Strong belief in opportunities (despite the manifold risks)
  - Simplified and less error-prone voting process
  - Better usability for voters with disabilities
  - Faster and more accurate tallying
  - Increased turnout
  - Pioneering role of Switzerland
  - Market opportunities by knowledge advantage
- Reports published in 2002, 2004, 2006, 2013
- Incentives for cantons to launch pilots
- "Federal Chancellery Ordinance on Electronic Voting", 2013 (updated in 2018)

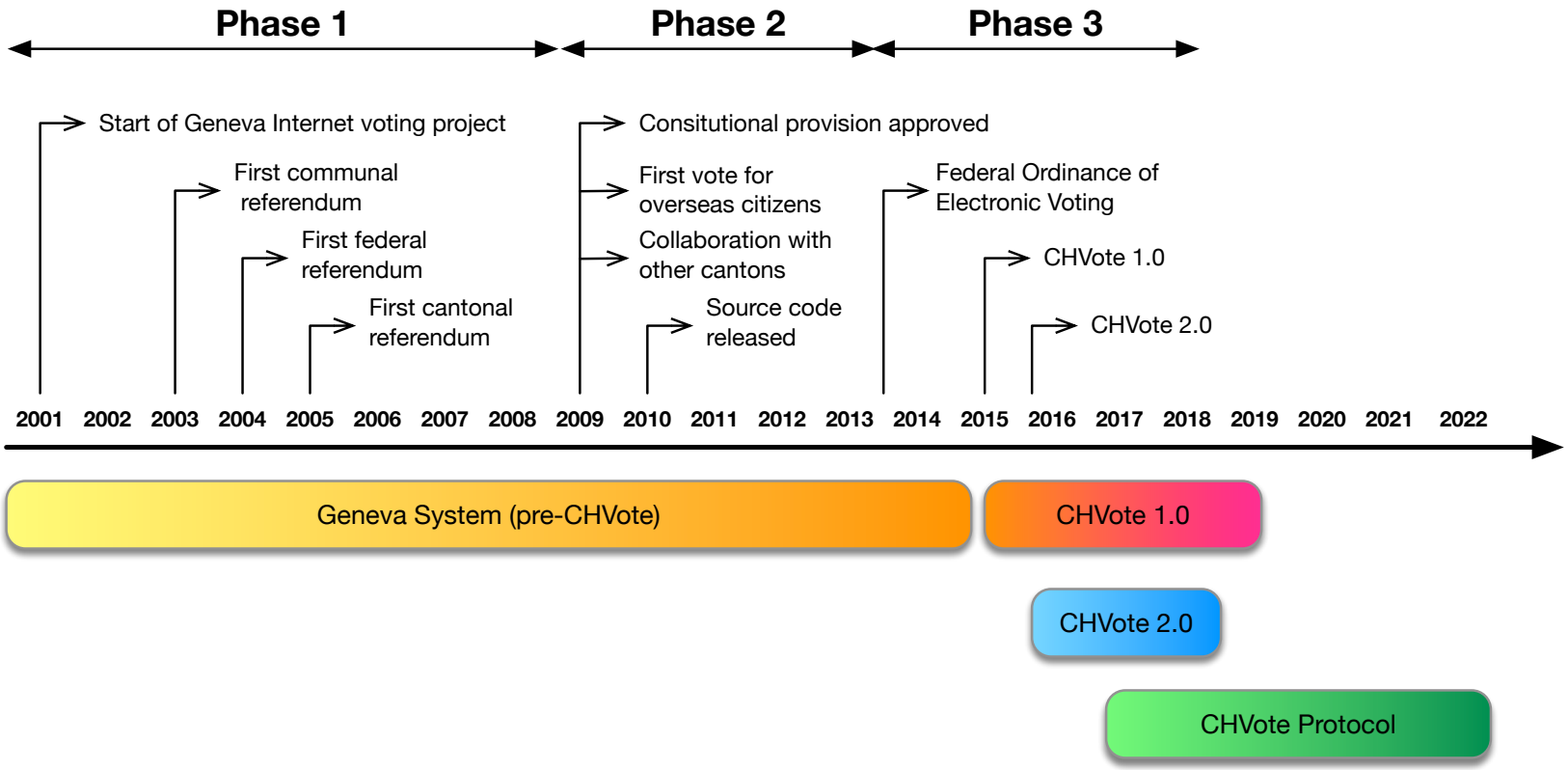
When was CHVote launched?



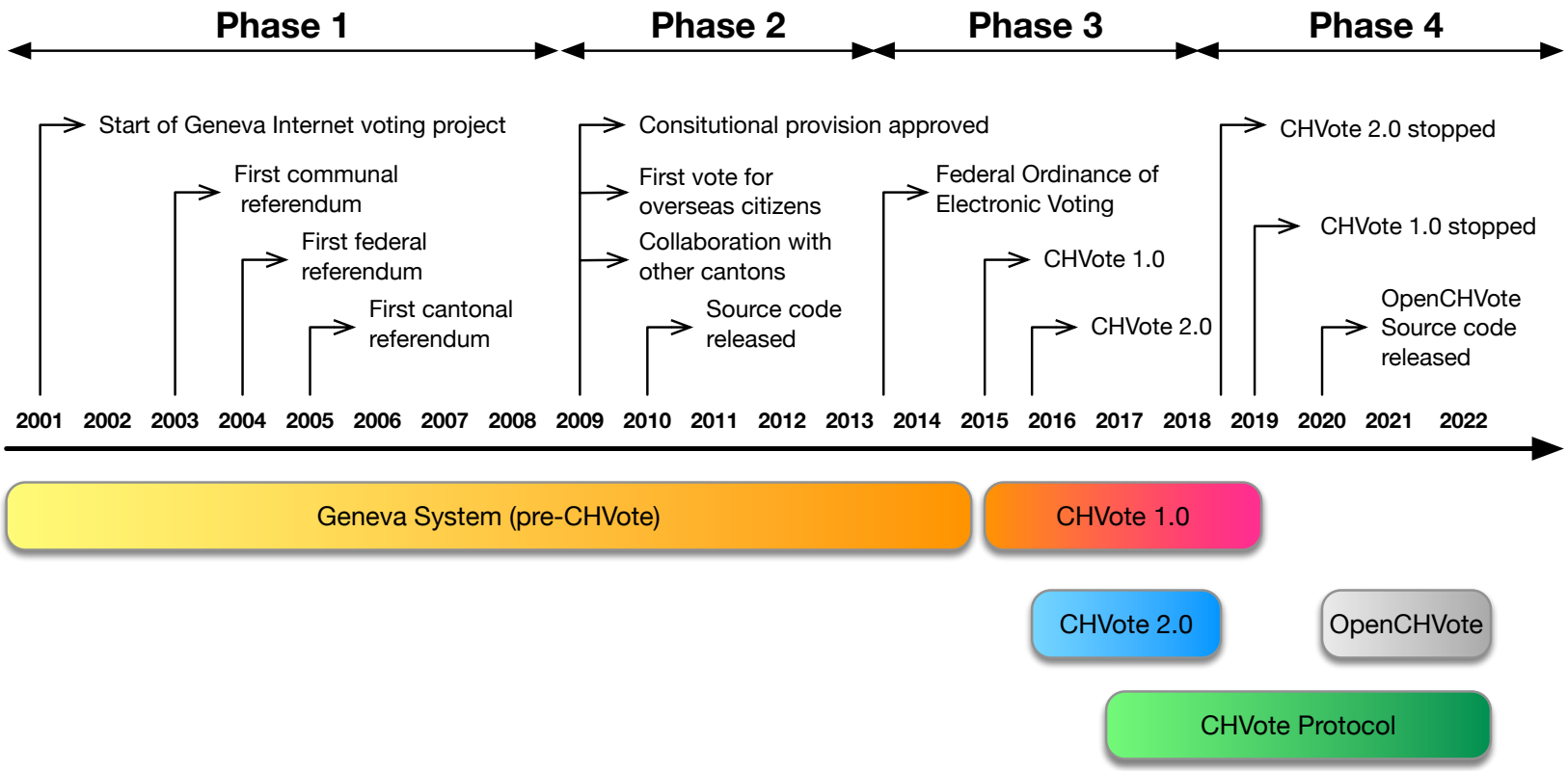
When was CHVote launched?



When was CHVote launched?



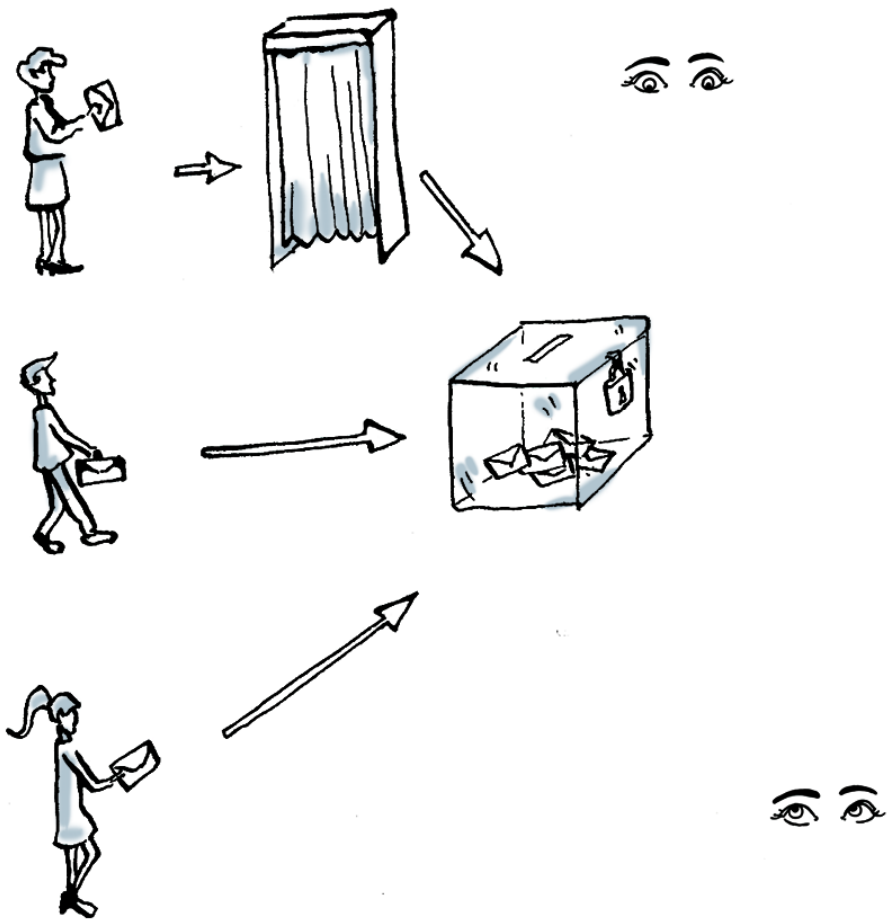
When was CHVote launched?



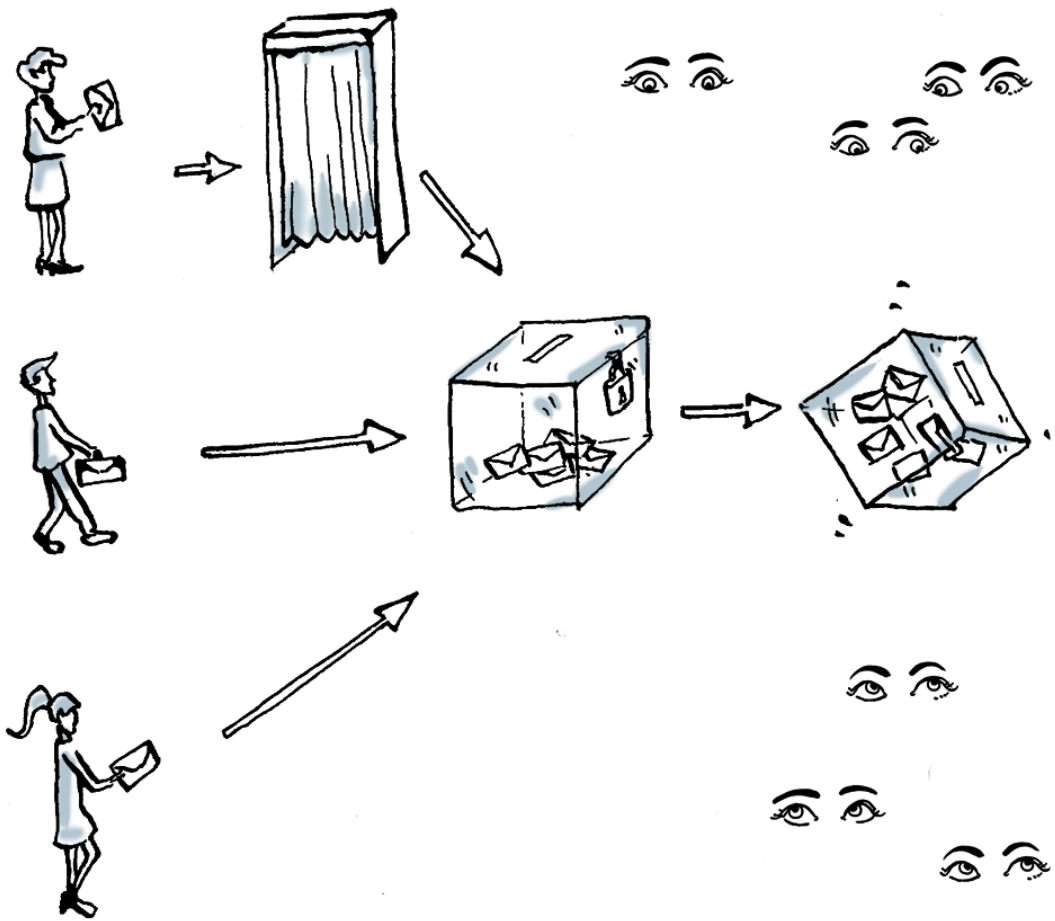
Outline

- Introduction
- Phase 1 & 2: The Geneva System (Pre-CHVote)
- Phase 3: CHVote 1.0 and CHVote 2.0
- Phase 4: CHVote Protocol and OpenCHVote
- Conclusion

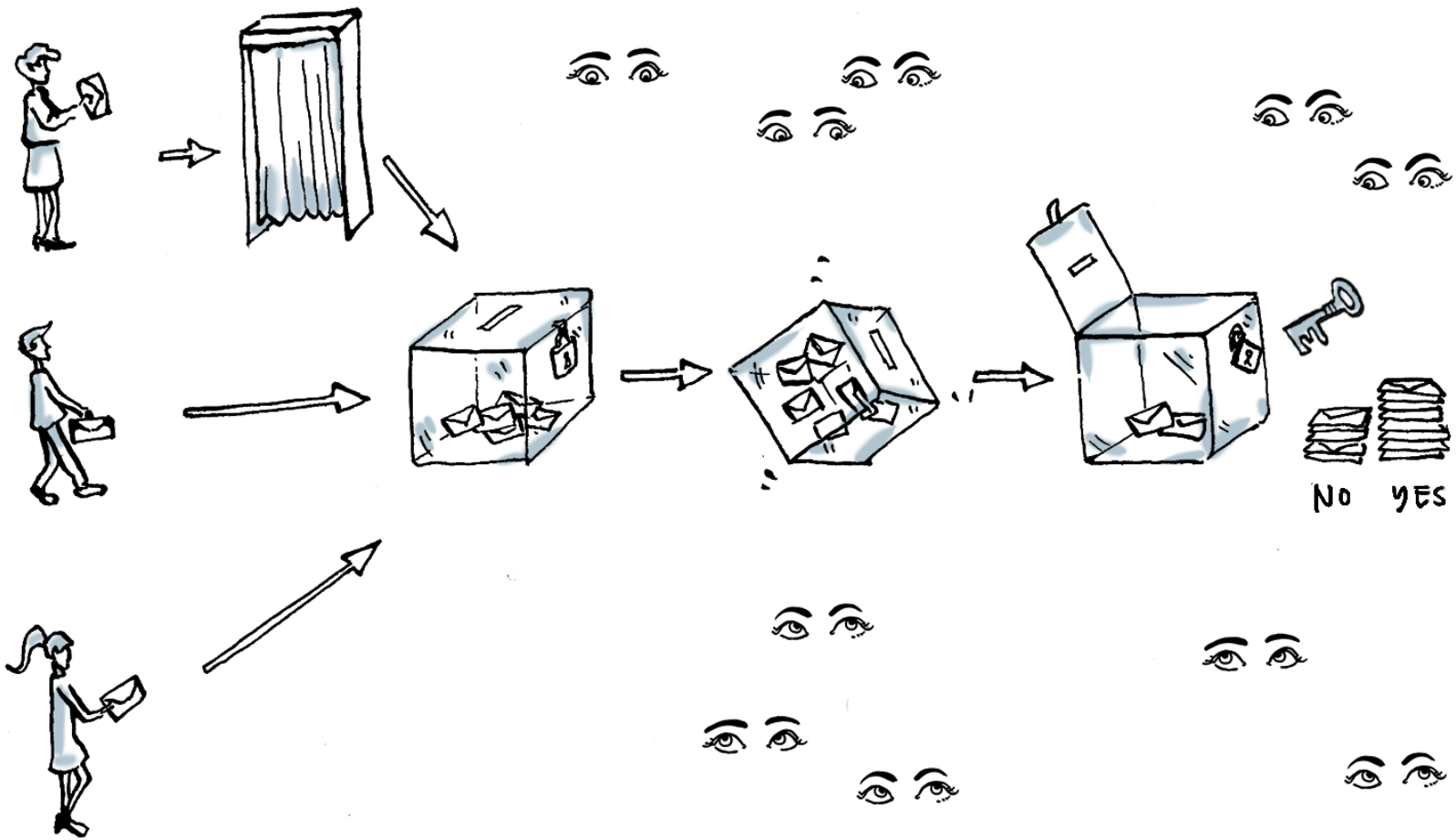
Traditional Paper-Based Voting



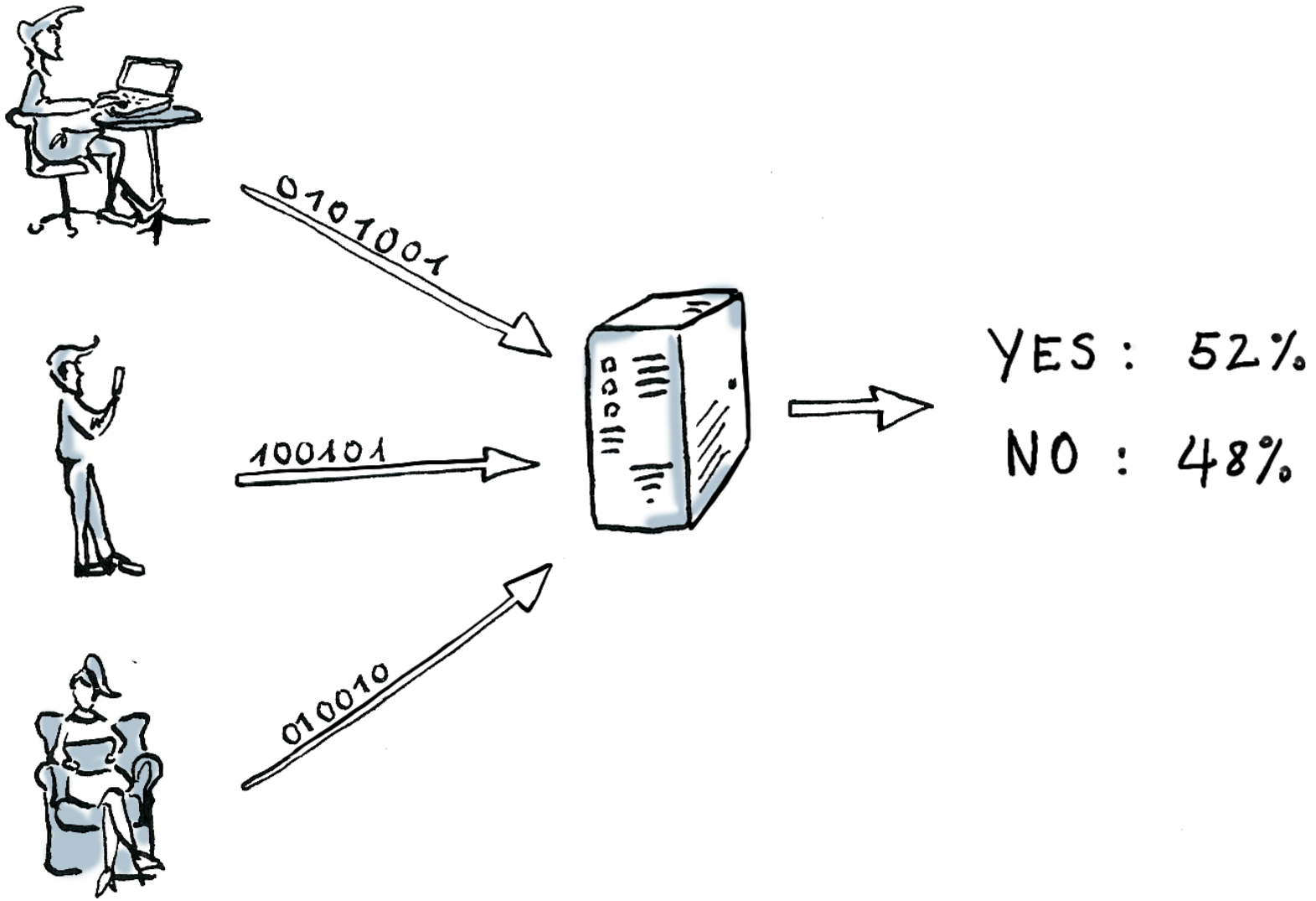
Traditional Paper-Based Voting



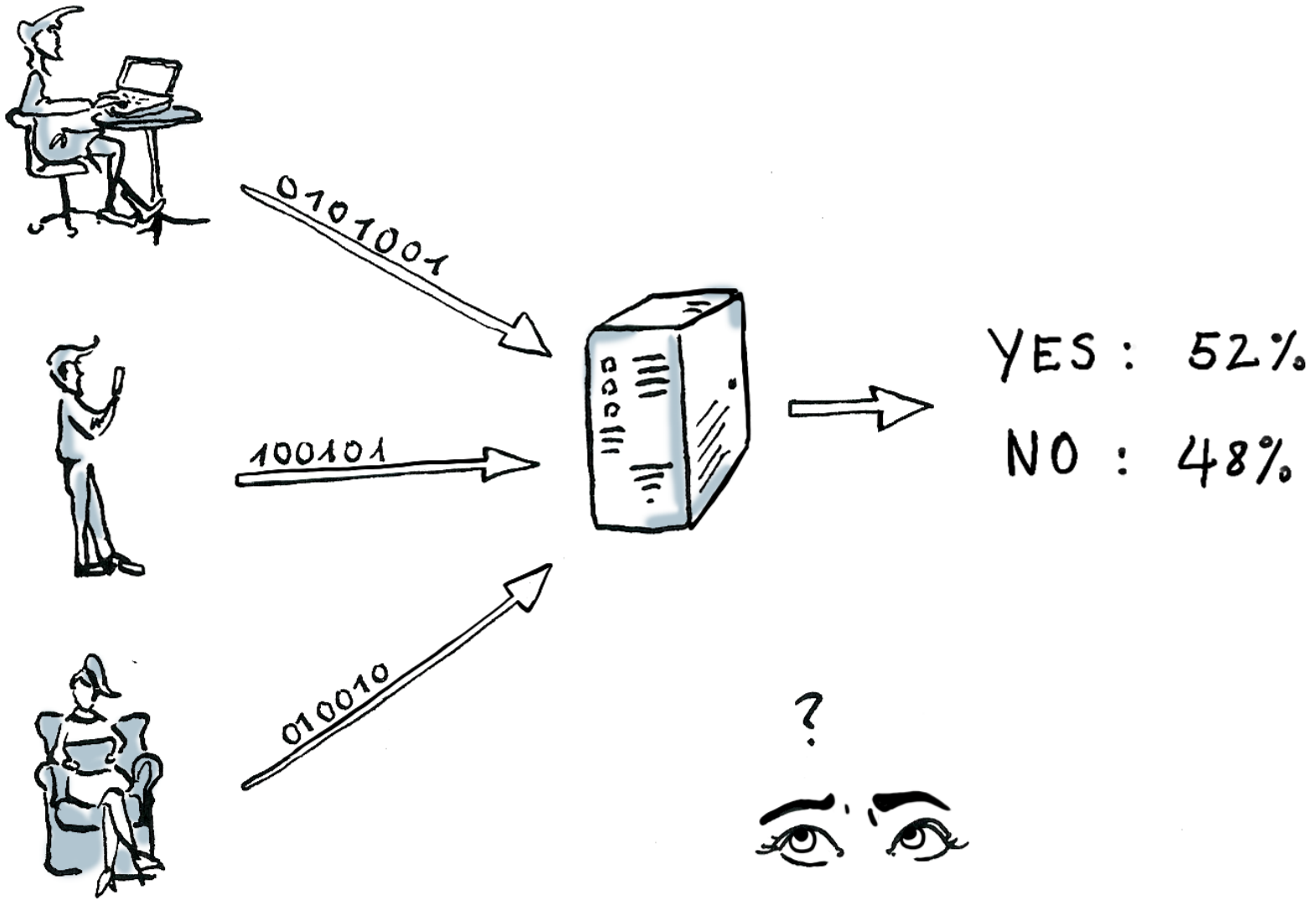
Traditional Paper-Based Voting



Blackbox Voting System



Blackbox Voting System



Flawed Security Concept

- Focus on providing functionality (with some security added)
- Strong trust assumptions (fully trusted server and client)
- Homemade cryptography (e.g. “Double Envelope Method”)
- Lack of proper security concepts
  - Security definitions
  - Adversary model
  - Cryptographic protocol
  - Formal security proofs
- Lack of transparency
  - Documentation
  - Source code (until 2010)
  - Evaluation reports
- No individual or universal verifiability

Strong Trust Assumptions

Question: *So you could possibly see all cleartext votes before the tally?*

Answer: Yes, but don't worry, we are adults!

Lack of Transparency

Question: *Did you perform any penetration tests?*

Answer: Yes.

Question: *What were the result?*

Answer: Sorry, we can't tell!

Outline

- Introduction
- Phase 1 & 2: The Geneva System (Pre-CHVote)
- Phase 3: CHVote 1.0 and CHVote 2.0
- Phase 4: CHVote Protocol and OpenCHVote
- Conclusion

3rd Vote Electronique Report

The introduction of verifiability is central to the new security requirements.

3rd Vote Electronique Report
Swiss Federal Council, 2013

Ordinance on Electronic Voting (VEleS)

- Effective since December 2013
- Enhanced security requirements
  - End-to-end encryption
  - Verifiability
  - Distributed trust
  - Transparency
- Step-wise extension
  - Step 1: Individual verifiability ($\leq$50% electorate)
  - Step 2: Individual and universal verifiability ($\leq$100% electorate)
- Updated in July 2018 (article on disclosing the source code)
- Next update expected in 2022

VEleS: Individual Verifiability

Voters must be able to ascertain whether their vote has been manipulated or intercepted on the user platform or during transmission. [. . .] Voters must receive proof that the server system has registered the vote as it was entered by the voter on the user platform.

Federal Chancellery Ordinance on Electronic Voting
VEleS, Art.4, 2013

Cast-as-Intended Verifiability

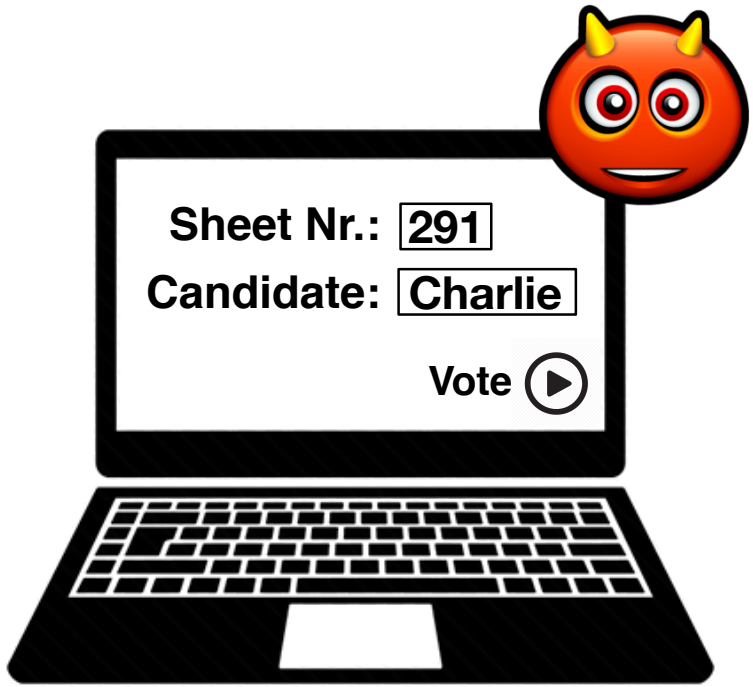
Code Sheet Nr. 291

Candidates	Codes
Alice	7449
Bob	8243
Charlie	9123

Cast-as-Intended Verifiability

Code Sheet Nr. 291

Candidates	Codes
Alice	7449
Bob	8243
Charlie	9123



Cast-as-Intended Verifiability

Code Sheet Nr. 291

Candidates	Codes
Alice	7449
Bob	8243
Charlie	9123

CHVote 1.0

- CHVote 1.0 is an extension of the Geneva System
- Goal: Cast-as-intended verifiability (1st extension step of VEleS)
- Simplistic solution:
  - Fully trusted server and printer
  - Upon receipt, encrypted ballots are decrypted by the server
  - Decrypted ballots are used to select the correct code from code database
  - Decrypted ballots are "forgotten"
  - Encrypted ballots are stored in ballot database
- First used in March 2015, last used in 2019
- Source code available at https://github.com/republique-et-canton-de-geneve/chvote-1-0

Liste de codes pour la carte n° 5874-8863-1400-8743			
Votation fédérale			
Question 1 Acceptez-vous l'arrêté fédéral du 20 juin 2013 portant règlement du financement et de l'aménagement de l'infrastructure ferroviaire (Contre-projet direct à l'initiative populaire "Pour les transports publics", qui a été retirée) ?	Oui A2B4	Non J5B9	Blanc Z8H5
Question 2 Acceptez-vous l'initiative populaire "Financer l'avortement est une affaire privée - Alléger l'assurance-maladie en radiant les coûts de l'interruption de grossesse de l'assurance de base" ?	Oui P8H3	Non X2A7	Blanc Q3L7
Votation cantonale			
Question 1 Acceptez-vous l'initiative 143 «Pour une véritable politique d'accueil de la Petite enfance» ?	Oui U6T4	Non P3D6	Blanc S6C2
Question 2 Acceptez-vous la loi constitutionnelle modifiant la constitution de la République et canton de Genève (Contreprojet à l'IN 143) (A 2 00 – 10895), du 15 décembre 2011 ?	Oui N4F2	Non M2A3	Blanc Q9L5
Question 3 **Question subsidiaire**: Si l'initiative (IN 143 «Pour une véritable politique d'accueil de la Petite enfance») et le contreprojet sont acceptés, lequel des deux a-t-il votre préférence ? Initiative 143 ? Contreprojet ?	IN K9W9	CP T3S6	Blanc Y2V4

Codes de vérification

1) Consultez les codes de vérification fournis dans votre matériel de vote
2) Vérifiez que les codes pour chaque question soient les mêmes entre cette page web et ceux de votre matériel de vote

Où trouver les codes ?

VOTATION FÉDÉRALE		VOS CHOIX	VOS CODES
1	Acceptez-vous l'initiative populaire «**Pour une économie durable et fondée sur une gestion efficiente des ressources (économie verte)**»?	NON	M9F2
2	Acceptez-vous l'initiative populaire «**AVSplus: pour une AVS forte**»?	NON	L3M8
3	Acceptez-vous la loi fédérale du 25 septembre 2015 sur le **renseignement** (LRens)?	NON	X3T6

VOTATION CANTONALE		VOS CHOIX	VOS CODES
1	Acceptez-vous la loi constitutionnelle modifiant la constitution de la République et canton de Genève (Cst-GE) *(Elections au système majoritaire)* (A 2 00 – 11757), du 26 février 2016?	NON	V3Q3

VEleS: Universal Verifiability

Auditors receive proof that the result has been ascertained correctly. They must evaluate the proof in a observable procedure. To do this, they must use technical aids that are independent of and isolated from the rest of the system.

Federal Chancellery Ordinance on Electronic Voting
VEleS, Art.5, 2013

CHVote 2.0

- Relaunched from scratch in 2016
- Expert dialogue initiated at project startup
- Collaboration with academia (BFH, EPFL, LORIA, University of Bristol)
- Mindset of maximal transparency
  - PoC source code released since October 2016
  - Cryptographic protocol published since April 2017
  - Security proofs published in October 2018
- Stopped in November 2018 for financial reasons (without ever using it in practice)
- Source code and documentation published in 2019 at https://chvote2.gitlab.io

State of Geneva official release of the CHVote 2.0 project

The State of Geneva hereby releases all the public material that have been created during the CHVote 2.0 applications development project. This includes the source code of the applications, the technical documentations, the functional requirements and the release notes.

Although the project has been discontinued, the released material allows any party to build and run an evoting system allowing to complete the whole process for a votation: prepare and start a ballot, cast votes, and finally decrypt the ballot box and get the results. It provides individual and universal verifiability using the security model including independent control components and insiders threats as required by the Federal Chancellery ordinance.

The source code is published under the AGPL v3 licence so that any piece of the work can be reused or studied by any third party having an interest in electronic voting, whether they are academic partners, researchers, students, citizens or any other public or private organization.

Please note that this release is not part of the current electronic voting service. Refer to the released source code and documentation of the CHVote 1.0 system instead.

Outline

- Introduction
- Phase 1 & 2: The Geneva System (Pre-CHVote)
- Phase 3: CHVote 1.0 and CHVote 2.0
- Phase 4: CHVote Protocol and OpenCHVote
- Conclusion

CHVote Protocol

- Designed by BFH for CHVote 2.0
- Available since 2017 on Cryptology ePrint Archive (2017/325)
- Current Version 3.2 (December 2020)
- Comprehensive, self-contained document (212 pages)
  - Cryptographic primitives
  - Protocol parties
  - Communication channels
  - Adversary model and security parameters
  - Election parameters
  - Protocol messages
  - Pseudo-code algorithms (≈ 100)
- Security proofs published separately (D. Berhard, V. Cortier, P. Gaudry, M. Turuani, B. Warinschi)

Cryptographic Ingredients

- ElGamal encryption
- Non-interactive zero-knowledge proofs
- Oblivious transfer (Chu-Tzeng protocol)
- Schnorr identification and signatures
- Shamir secret sharing
- Re-encryption mix-net (Wikström)
- Distributed decryption
- Key-encapsulation mechanism (hybrid encryption)

Transmission of Verification Codes

Code Sheet Nr. 291

Candidates	Codes
Alice	7449
Bob	8243
Charlie	9123

- The voting server does not learn the voter's selection
- The voting client does not learn codes different from the voter's selection

Oblivious Transfer

- In cryptography, an oblivious transfer (OT) is a protocol between a sender and a receiver
  - The sender has n messages $\mathbf{m} = (m_1, \ldots, m_n)$
  - The receiver selects k indices $\mathbf{s} = (s_1, \ldots, s_k)$, $s_i \in \{1, \ldots, n\}$
  - Executing the protocol reveals $\mathbf{m_s} = (m_{s_1}, \ldots, m_{s_k})$ to the receiver
- Properties of OT protocols
  - Receiver privacy: the sender learns nothing about $\mathbf{s}$
  - Sender privacy: the receiver learns nothing more than $\mathbf{m_s}$
- In the efficient OT-protocol by Chu and Tzeng, the receiver's query consists of ElGamal encryptions

OT-Protocol by Chu and Tzeng

Receiver
selects $\mathbf{s} = (s_1, \ldots, s_k)$

for $j = 1, \ldots, k$
– pick random $r_j \in_R \mathbb{Z}_q$
– compute $a_j = \Gamma(s_j) \cdot g^{r_j}$

$\mathbf{a}=(a_1,\ldots,a_k)$ ⟶

Sender
knows $\mathbf{m} = (m_1, \ldots, m_n)$

pick random $r \in_R \mathbb{Z}_q$
for $j = 1, \ldots, k$
– compute $b_j = a_j^r$
for $i = 1, \ldots, n$
– compute $k_i = H(\Gamma(i)^r)$
– compute $c_i = m_i \oplus k_i$
compute $d = g^r$

⟵ $\mathbf{b}=(b_1,\ldots,b_k)$, $\mathbf{c}=(c_1,\ldots,c_n)$, d

for $j = 1, \ldots, k$
– compute $k_j = H(b_j \cdot d^{-r_j})$
– compute $m_{s_j} = c_{s_j} \oplus k_j$
let $\mathbf{m_s} = (m_{s_1}, \ldots, m_{s_k})$

Example of Protocol Description

Voting Client		Election Authority $j \in \{1, \ldots, s\}$
knows $v, X_v, \mathbf{s}, VP_v$		knows $EP, pk, \hat{\mathbf{x}}, pk_j, \pi_j, \mathbf{P}_j, B_j, F_j$
	$\xrightarrow{v}$	
		$VP_v \leftarrow \mathsf{GetVotingParameters}(v, EP)$
	$\xleftarrow{[pk_j, \pi_j]_{VP_v}}$	
$\mathbf{pk} \leftarrow (pk_1, \ldots, pk_s)$		
$\boldsymbol{\pi} \leftarrow (\pi_1, \ldots, \pi_s)$		
if $\neg\mathsf{CheckKeyPairProofs}(\boldsymbol{\pi}, \mathbf{pk})$ **abort**		
$pk \leftarrow \mathsf{GetPublicKey}(\mathbf{pk})$		
$(\cdot, \cdot, \cdot, \cdot, \mathbf{n}, \cdot, \cdot, w_v) \leftarrow VP_v$		
$(\alpha, \mathbf{r}) \leftarrow \mathsf{GenBallot}(X_v, \mathbf{s}, pk, \mathbf{n}, w_v)$		
	$\xrightarrow{v, \alpha}$	

Example of Protocol Description

Voting Client		Election Authority $j \in \{1, \ldots, s\}$
knows $v, X_v, \mathbf{s}, VP_v$		knows $EP, pk, \hat{\mathbf{x}}, pk_j, \pi_j, \mathbf{P}_j, B_j, F_j$
	$\xrightarrow{v}$	
		$VP_v \leftarrow \mathsf{GetVotingParameters}(v, EP)$
	$\xleftarrow{[pk_j, \pi_j]_{VP_v}}$	
$\mathbf{pk} \leftarrow (pk_1, \ldots, pk_s)$ $\boldsymbol{\pi} \leftarrow (\pi_1, \ldots, \pi_s)$ **if** $\neg\mathsf{CheckKeyPairProofs}(\boldsymbol{\pi}, \mathbf{pk})$ **abort** $pk \leftarrow \mathsf{GetPublicKey}(\mathbf{pk})$ $(\cdot, \cdot, \cdot, \cdot, \mathbf{n}, \cdot, \cdot, w_v) \leftarrow VP_v$ $(\alpha, \mathbf{r}) \leftarrow \mathsf{GenBallot}(X_v, \mathbf{s}, pk, \mathbf{n}, w_v)$		
	$\xrightarrow{v, \alpha}$	

Example of Pseudo-Code Algorithm

Algorithm: GenBallot$(X, \mathbf{s}, pk, \mathbf{n}, w)$

Input: Voting code $X \in A_X^{\ell_X}$
Selection $\mathbf{s} = (s_1, \ldots, s_k)$, $1 \leqslant s_1 < \cdots < s_k \leqslant n$
Encryption key $pk \in \mathbb{G}_q$
Number of candidates $\mathbf{n} = (n_1, \ldots, n_t)$, $n_j \in \mathbb{N}^+$, $n = \sum_{j=1}^{t} n_j$
Counting circle $w \in \mathbb{N}^+$

$x \leftarrow$ ToInteger(X, A_x) // see Alg. 4.8
$\hat{x} \leftarrow \hat{g}^x \bmod \hat{p}$
$\mathbf{p} \leftarrow$ GetPrimes$(n + w)$ // $\mathbf{p} = (p_0, \ldots, p_{n+w})$, see Alg. 8.1
$\mathbf{m} \leftarrow$ GetEncodedSelections$(\mathbf{s}, \mathbf{p})$ // $\mathbf{m} = (m_1, \ldots, m_k)$, see Alg. 8.24
$m \leftarrow \prod_{j=1}^{k} m_j$
if $p_{n+w} \cdot m \geqslant p$ **then**
  return $\perp$ // $\mathbf{s}$, $\mathbf{n}$, and w are incompatible with p
$(\mathbf{a}, \mathbf{r}) \leftarrow$ GenQuery$(\mathbf{m}, pk)$ // $\mathbf{a} = (a_1, \ldots, a_k)$, $\mathbf{r} = (r_1, \ldots, r_k)$, see Alg. 8.25
$r \leftarrow \sum_{j=1}^{k} r_j \bmod q$
$\pi \leftarrow$ GenBallotProof$(x, m, r, \hat{x}, \mathbf{a}, pk)$ // see Alg. 8.26
$\alpha \leftarrow (\hat{x}, \mathbf{a}, \pi)$
return $(\alpha, \mathbf{r})$ // $\alpha \in \mathbb{G}_{\hat{q}} \times (\mathbb{G}_q^2)^k \times (\mathbb{Z}_{2^\tau} \times (\mathbb{Z}_{\hat{q}} \times \mathbb{G}_q \times \mathbb{Z}_q))$, $\mathbf{r} \in \mathbb{Z}_q^k$

Example of Pseudo-Code Algorithm

Algorithm: GenBallot$(X, \mathbf{s}, pk, \mathbf{n}, w)$

Input: Voting code $X \in A_X^{\ell_X}$
Selection $\mathbf{s} = (s_1, \ldots, s_k)$, $1 \leqslant s_1 < \cdots < s_k \leqslant n$
Encryption key $pk \in \mathbb{G}_q$
Number of candidates $\mathbf{n} = (n_1, \ldots, n_t)$, $n_j \in \mathbb{N}^+$, $n = \sum_{j=1}^{t} n_j$
Counting circle $w \in \mathbb{N}^+$

$x \leftarrow$ ToInteger(X, A_x) // see Alg. 4.8
$\hat{x} \leftarrow \hat{g}^x \bmod \hat{p}$
$\mathbf{p} \leftarrow$ GetPrimes$(n + w)$ // $\mathbf{p} = (p_0, \ldots, p_{n+w})$, see Alg. 8.1
$\mathbf{m} \leftarrow$ GetEncodedSelections$(\mathbf{s}, \mathbf{p})$ // $\mathbf{m} = (m_1, \ldots, m_k)$, see Alg. 8.24
$m \leftarrow \prod_{j=1}^{k} m_j$
if $p_{n+w} \cdot m \geqslant p$ **then**
  return $\perp$ // $\mathbf{s}$, $\mathbf{n}$, and w are incompatible with p
$(\mathbf{a}, \mathbf{r}) \leftarrow$ GenQuery$(\mathbf{m}, pk)$ // $\mathbf{a} = (a_1, \ldots, a_k)$, $\mathbf{r} = (r_1, \ldots, r_k)$, see Alg. 8.25
$r \leftarrow \sum_{j=1}^{k} r_j \bmod q$
$\pi \leftarrow$ GenBallotProof$(x, m, r, \hat{x}, \mathbf{a}, pk)$ // see Alg. 8.26
$\alpha \leftarrow (\hat{x}, \mathbf{a}, \pi)$
return $(\alpha, \mathbf{r})$ // $\alpha \in \mathbb{G}_{\hat{q}} \times (\mathbb{G}_q^2)^k \times (\mathbb{Z}_{2^\tau} \times (\mathbb{Z}_{\hat{q}} \times \mathbb{G}_q \times \mathbb{Z}_q))$, $\mathbf{r} \in \mathbb{Z}_q^k$

OpenCHVote

- OpenCHVote 1.0 has been released in October 2020 on Gitlab
- Along with an updated CHVote Protocol Specification
- GNU Affero General Public License Version 3
- Funded by the State of Geneva (2016–2019) and the Federal Chancellery (2019–2020)
- Main project goals:
  - Cover all cryptographically relevant parts
  - Exclude all cryptographically irrelevant parts
  - Eliminate gap between specification and code as far as possible
  - Easy to install, execute, and test

OpenCHVote: Key Features

- Complete parameterizable CHVote protocol run
- Simultaneous execution of multiple protocol runs
- Two protocol variants: *plain* and *writein*
- Optimized performance
- Clean infrastructure interfaces to cryptographically irrelevant components
- No compulsory dependencies to third-party libraries/frameworks
- Straightforward installation (Maven)

Example of Algorithm Implementation

```java
public class GenBallot {

    public static Pair<Ballot, Vector<BigInteger>>
        run(String X, IntVector bold_s, QuadraticResidue pk, IntVector bold_n, int w, Parameters params)

        // PREPARATION
        Precondition.checkNotNull(X, bold_s, pk, bold_n, params);
        int k = bold_s.getLength();
        int t = bold_n.getLength();
        int n = Math.intSum(bold_n);
        Precondition.check(params.GG_q.contains(pk));
        Precondition.check(IntSet.NN_plus.contains(w));
        Precondition.check(Set.String(params.A_X, params.ell_X).contains(X));
        Precondition.check(Set.IntVector(IntSet.NN_plus, t).contains(bold_n));
        Precondition.check(Set.IntVector(IntSet.NN_plus(n), k).contains(bold_s));
        Precondition.check(bold_s.isSorted());

        // ALGORITHM
        var x = ToInteger.run(X, params.A_X);
        var x_hat = Mod.pow(params.g_hat, x, params.p_hat);
        var bold_p = GetPrimes.run(n + w, params);
        var bold_m = GetEncodedSelections.run(bold_s, bold_p);
        var m = Math.prod(bold_m.map(QuadraticResidue::getValue));
        if (bold_p.getValue(n + w).getValue().multiply(m).compareTo(params.p) >= 0)
            throw new AlgorithmException(GenBallot.class, AlgorithmException.Type.INCOMPATIBLE_MATRIX);
        var pair = GenQuery.run(bold_m, pk, params);
        var bold_a = pair.getFirst();
        var bold_r = pair.getSecond();
        var r = Mod.sum(bold_r, params.q);
        var pi = GenBallotProof.run(x, Mod.prod(bold_m), r, x_hat, bold_a, pk, params);
        var alpha = new Ballot(x_hat, bold_a, pi);
        return new Pair<>(alpha, bold_r);
    }
}
```

Example of Algorithm Implementation

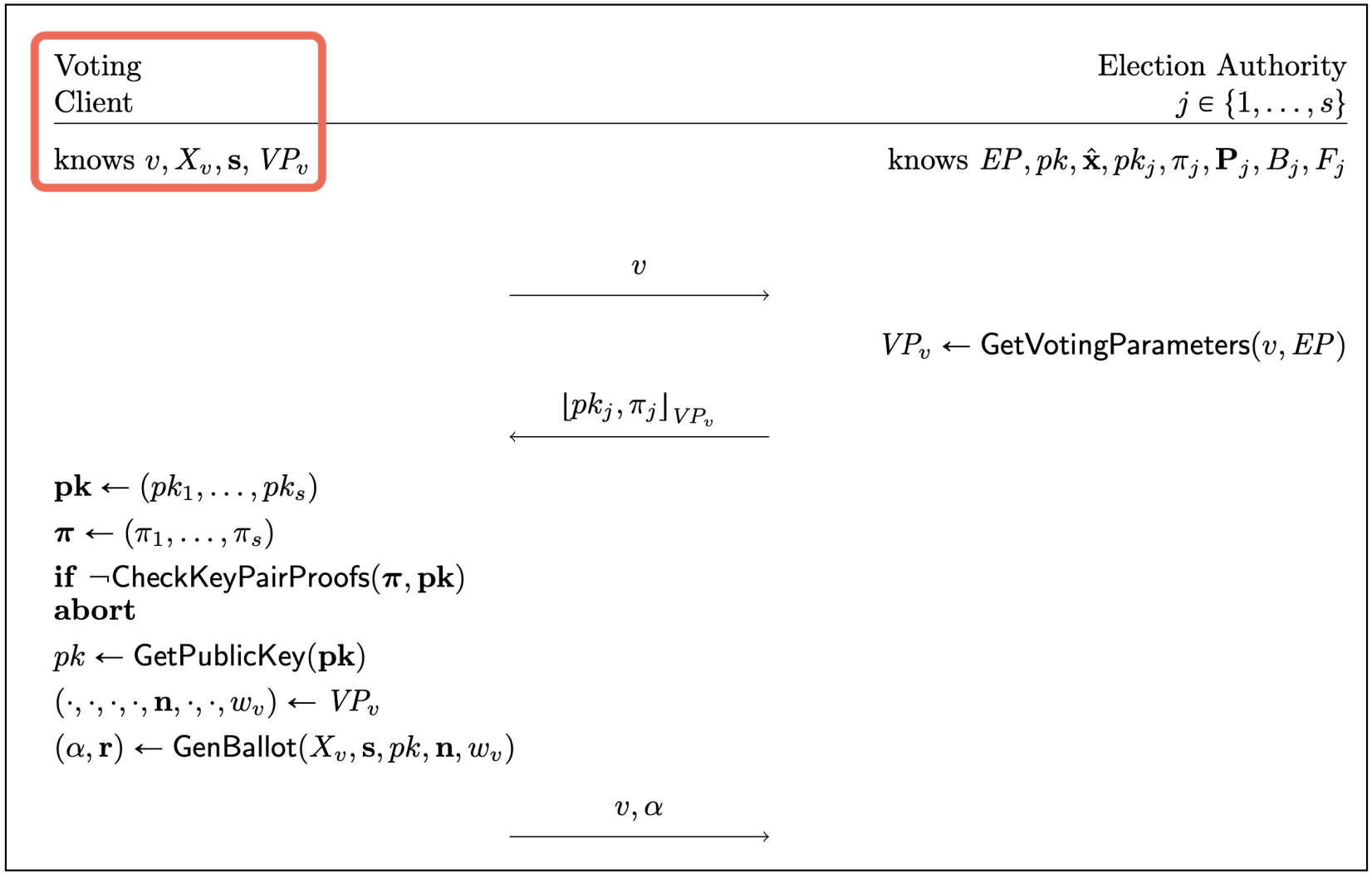
```java
public class GenBallot {

    public static Pair<Ballot, Vector<BigInteger>>
        run(String X, IntVector bold_s, QuadraticResidue pk, IntVector bold_n, int w, Parameters params)

        // PREPARATION
        Precondition.checkNotNull(X, bold_s, pk, bold_n, params);
        int k = bold_s.getLength();
        int t = bold_n.getLength();
        int n = Math.intSum(bold_n);
        Precondition.check(params.GG_q.contains(pk));
        Precondition.check(IntSet.NN_plus.contains(w));
        Precondition.check(Set.String(params.A_X, params.ell_X).contains(X));
        Precondition.check(Set.IntVector(IntSet.NN_plus, t).contains(bold_n));
        Precondition.check(Set.IntVector(IntSet.NN_plus(n), k).contains(bold_s));
        Precondition.check(bold_s.isSorted());

        // ALGORITHM
        var x = ToInteger.run(X, params.A_X);
        var x_hat = Mod.pow(params.g_hat, x, params.p_hat);
        var bold_p = GetPrimes.run(n + w, params);
        var bold_m = GetEncodedSelections.run(bold_s, bold_p);
        var m = Math.prod(bold_m.map(QuadraticResidue::getValue));
        if (bold_p.getValue(n + w).getValue().multiply(m).compareTo(params.p) >= 0)
            throw new AlgorithmException(GenBallot.class, AlgorithmException.Type.INCOMPATIBLE_MATRIX);
        var pair = GenQuery.run(bold_m, pk, params);
        var bold_a = pair.getFirst();
        var bold_r = pair.getSecond();
        var r = Mod.sum(bold_r, params.q);
        var pi = GenBallotProof.run(x, Mod.prod(bold_m), r, x_hat, bold_a, pk, params);
        var alpha = new Ballot(x_hat, bold_a, pi);
        return new Pair<>(alpha, bold_r);
    }
}
```

Pseudo-Code vs. Java Code

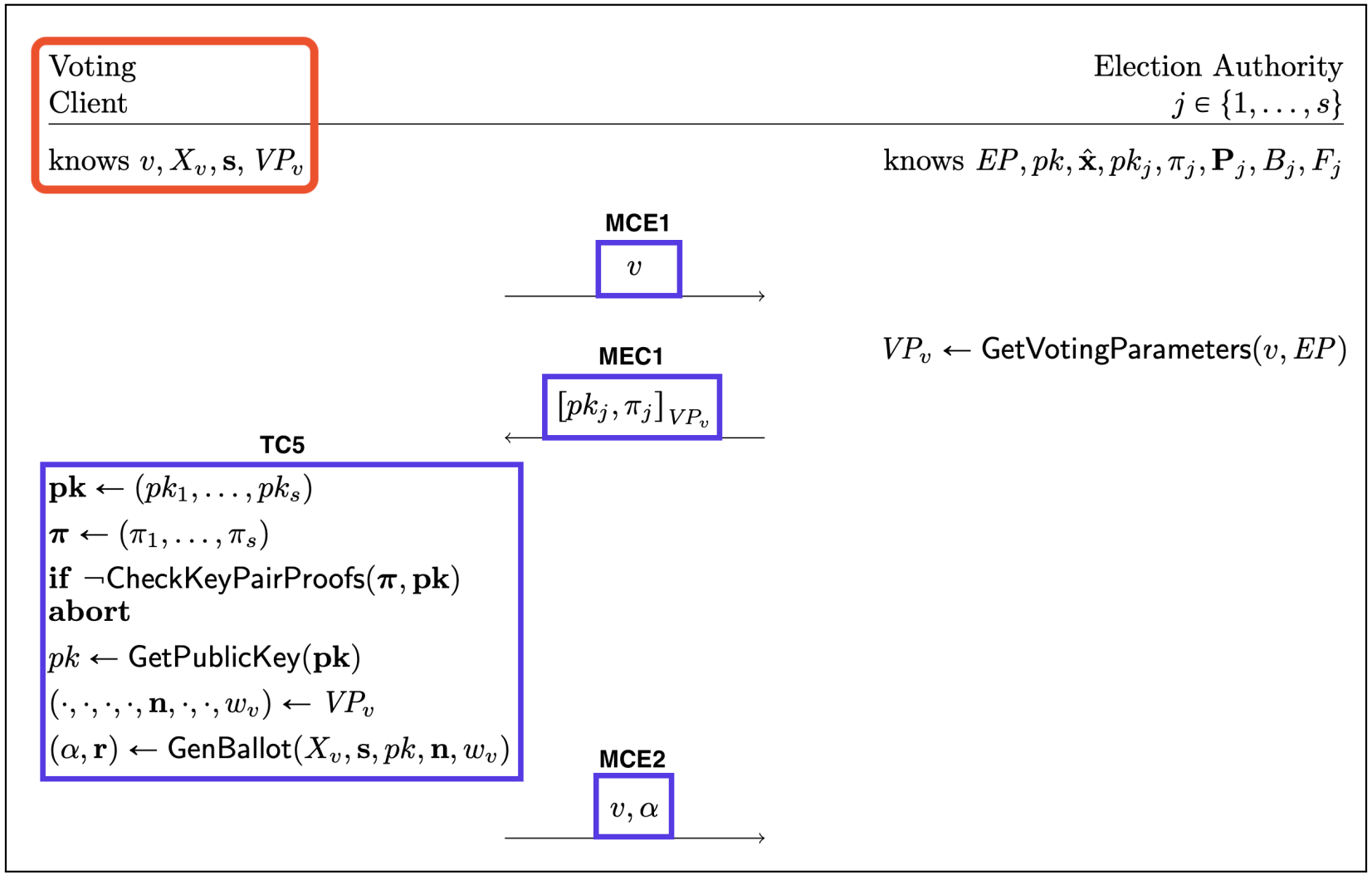
$$
\begin{aligned}
x &\leftarrow \mathsf{ToInteger}(X, A_x) \\
\hat{x} &\leftarrow \hat{g}^x \bmod \hat{p} \\
\mathbf{p} &\leftarrow \mathsf{GetPrimes}(n + w) \\
\mathbf{m} &\leftarrow \mathsf{GetEncodedSelections}(\mathbf{s}, \mathbf{p}) \\
m &\leftarrow \textstyle\prod_{j=1}^{k} m_j
\end{aligned}
$$

```
var x = ToInteger.run(X, params.A_X);
var x_hat = Mod.pow(params.g_hat, x, params.p_hat);
var bold_p = GetPrimes.run(n + w, params);
var bold_m = GetEncodedSelections.run(bold_s, bold_p);
var m = Math.prod(bold_m.map(QuadraticResidue::getValue));
```

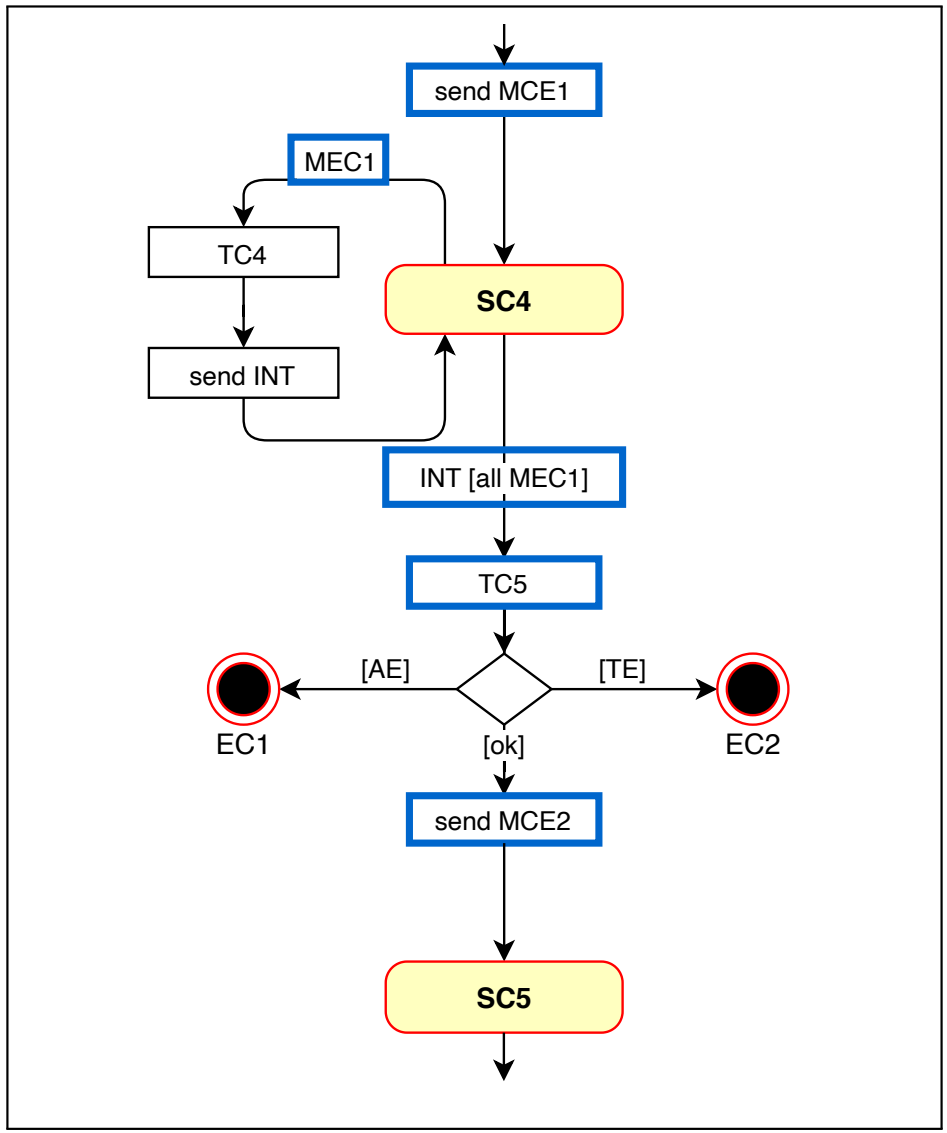
Example of Protocol Party



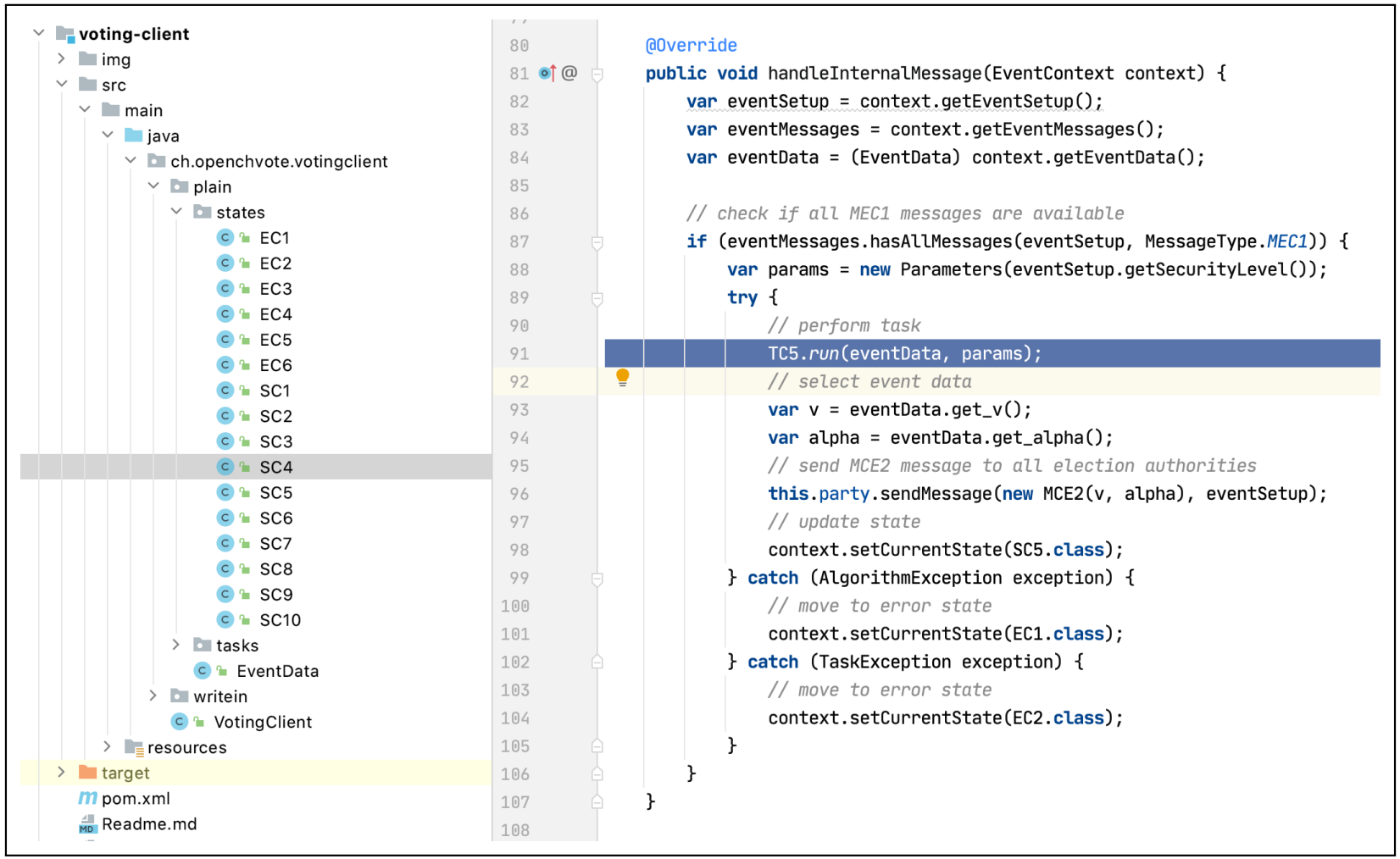
Example of Protocol Party



Example of Protocol Party



Example of Protocol Party



Example of Protocol Party

```java
public class TC5 {

    public static void run(EventData eventData, Parameters params) {

        // select event data
        var X_v = eventData.get_X_v();
        var bold_s = eventData.get_bold_s();
        var VP_v = eventData.get_VP_v();

        // perform main task
        var bold_pk = eventData.get_bold_pk();
        var bold_pi = eventData.get_bold_pi();
        if (!CheckKeyPairProofs.run(bold_pi, bold_pk, params)) {
            throw new TaskException(TC5.class, TaskException.Type.INVALID_ZKP_PROOF);
        }
        var pk = GetPublicKey.run(bold_pk, params);
        var bold_n = VP_v.get_bold_n();
        var w_v = VP_v.get_w_v();
        var pair = GenBallot.run(X_v, bold_s, pk, bold_n, w_v, params);
        var alpha = pair.getFirst();
        var bold_r = pair.getSecond();

        // update event data
        eventData.set_pk(pk);
        eventData.set_alpha(alpha);
        eventData.set_bold_r(bold_r);
    }
```

Example of Protocol Party

$$
\begin{aligned}
&\mathbf{pk} \leftarrow (pk_1, \ldots, pk_s) \\
&\boldsymbol{\pi} \leftarrow (\pi_1, \ldots, \pi_s) \\
&\textbf{if } \neg \mathsf{CheckKeyPairProofs}(\boldsymbol{\pi}, \mathbf{pk}) \\
&\textbf{abort} \\
&pk \leftarrow \mathsf{GetPublicKey}(\mathbf{pk}) \\
&(\cdot, \cdot, \cdot, \cdot, \mathbf{n}, \cdot, \cdot, w_v) \leftarrow VP_v \\
&(\alpha, \mathbf{r}) \leftarrow \mathsf{GenBallot}(X_v, \mathbf{s}, pk, \mathbf{n}, w_v)
\end{aligned}
$$

```
var bold_pk = eventData.get_bold_pk();
var bold_pi = eventData.get_bold_pi();
if (!CheckKeyPairProofs.run(bold_pi, bold_pk, params))
    throw new TaskException(TC5.class, TaskException.Type.INVALID_ZKP_PROOF);
var pk = GetPublicKey.run(bold_pk, params);
var bold_n = VP_v.get_bold_n();
var w_v = VP_v.get_w_v();
var pair = GenBallot.run(X_v, bold_s, pk, bold_n, w_v, params);
var alpha = pair.getFirst();
var bold_r = pair.getSecond();
```

Outline

- Introduction
- Phase 1 & 2: The Geneva System (Pre-CHVote)
- Phase 3: CHVote 1.0 and CHVote 2.0
- Phase 4: CHVote Protocol and OpenCHVote
- Conclusion

Conclusion

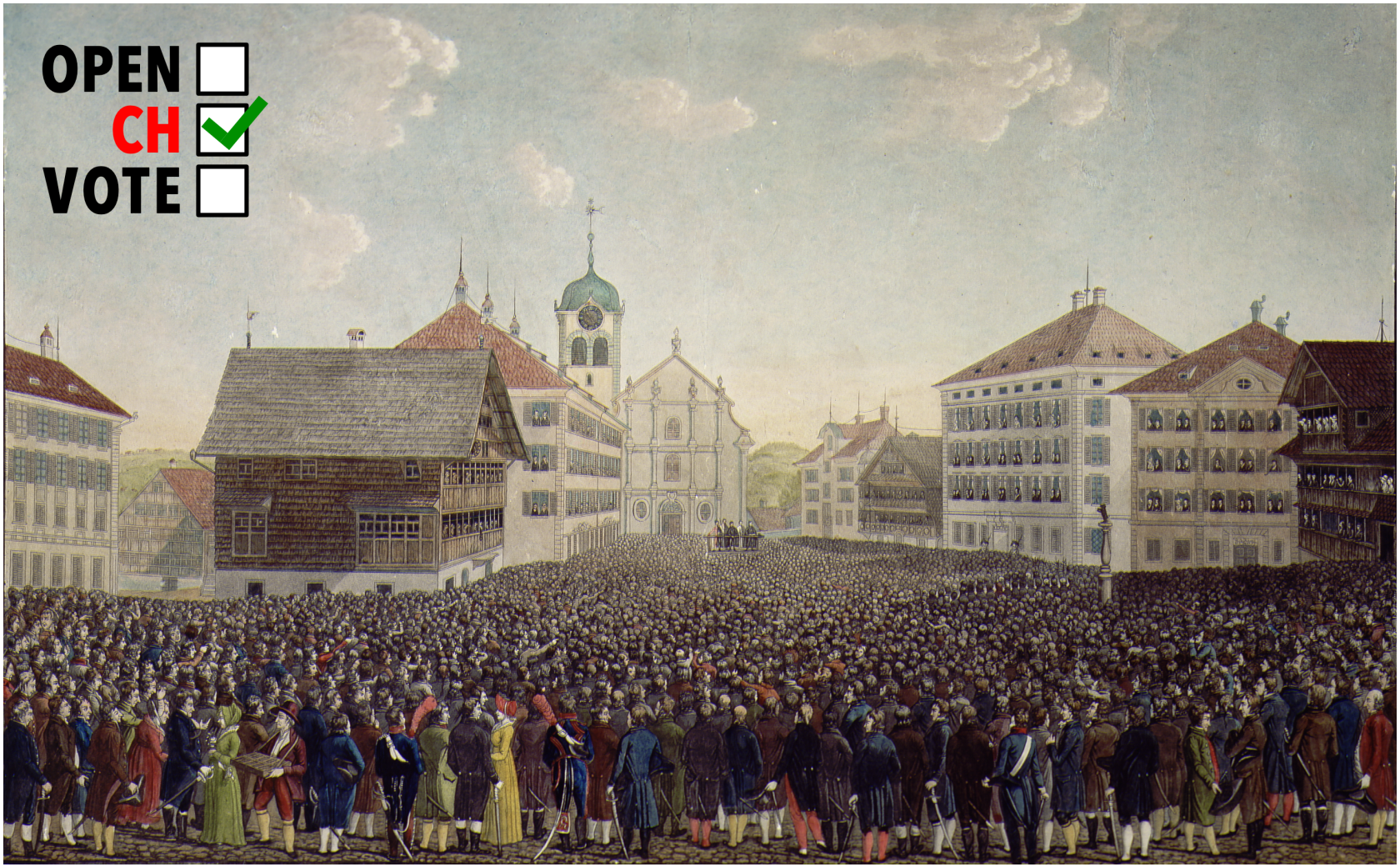
Phases 1–3

- Long journey of learning
- Right decision to discontinue CHVote 1.0
- Right mindset brought into CHVote 2.0
- Were the VEleS requirements too ambitious?

Phase 4

- CHVote and OpenCHVote are two shapes of the same thing
- As such, it may serve as an example for other projects
- What's next?

Questions?



Source: https://en.wikipedia.org/wiki/Landsgemeinde