Bern University of Applied Sciences

# CHVote – Sixteen Best Practices and Lessons Learned

*R. Haenni, E. Dubuis, R. E. Koenig, P. Locher*

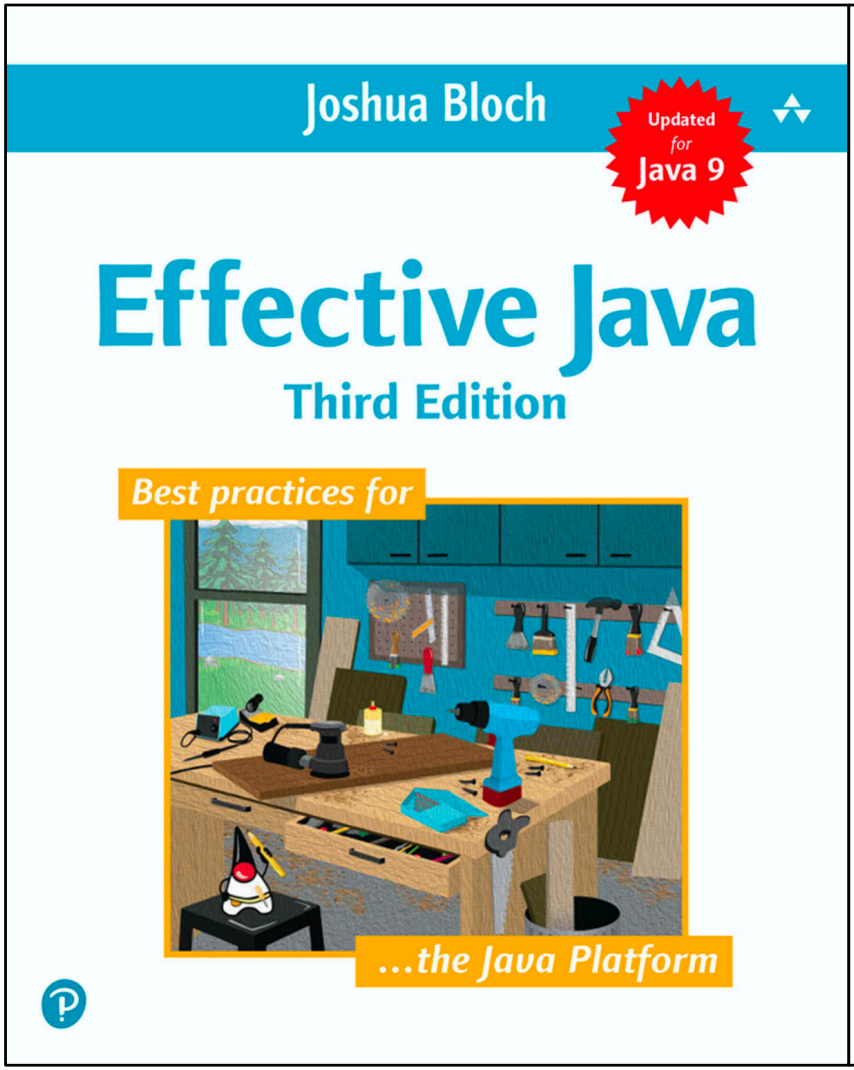
E-Vote-ID 2020, October 8th, 2020

# Paper Overview

| | |
|---|---|
| Item 1: | Modeling the Electoral Systems |
| Item 2: | Modeling the Electorate |
| Item 3: | Cryptographic Building Blocks |
| Item 4: | Cryptographic Parameters |
| Item 5: | Parties and Communication |
| Item 6: | Protocol Structure and Communication Diagrams |
| Item 7: | Pseudo-Code Algorithms |
| Item 8: | Usability and Performance |
| Item 9: | Mathematical Library |
| Item 10: | Naming Conventions |
| Item 11: | Implementation of Pseudo-Code Algorithms |
| Item 12: | Parameter Validity Checks |
| Item 13: | Implementation of Protocol Parties |
| Item 14: | Cryptographically Relevant Code |
| Item 15: | Transparency |
| Item 16: | Verifier |

# Paper Overview

| | |
|---|---|
| Item 1: | Modeling the Electoral Systems |
| Item 2: | Modeling the Electorate |
| Item 3: | Cryptographic Building Blocks |
| Item 4: | Cryptographic Parameters |
| Item 5: | Parties and Communication |
| Item 6: | Protocol Structure and Communication Diagrams |
| Item 7: | Pseudo-Code Algorithms |
| Item 8: | Usability and Performance |
| Item 9: | Mathematical Library |
| Item 10: | Naming Conventions |
| Item 11: | Implementation of Pseudo-Code Algorithms |
| Item 12: | Parameter Validity Checks |
| Item 13: | Implementation of Protocol Parties |
| Item 14: | Cryptographically Relevant Code |
| Item 15: | Transparency |
| Item 16: | Verifier |




# Contents

Bern University of Applied Sciences

# Item 17: Software Announcement

# Item 17: Software Announcement

To continue the CHVote project independently of the support from the State of Geneva, a new funding from *eGovernment Switzerland* has been acquired by the Bern University of Applied Sciences in August 2019. The main goal of this project was to release a final stable version of the specification document and to update the cryptographic core of the protocol based on the code released by the State of Geneva. As a first project deliverable, the current Version 3.0 of the specification document has been released in December 2019 [9]. At the time of writing this paper, the developed *OpenCHVote* software is not yet complete. Since the project is in its final stage, the code is expected to be released soon under a non-proprietary license.[3] The general purpose of the project is to make the achievements available to others for pursuing it further.

[1] For further details about the reasons for abandoning the project, we refer to the State Council's press statement at https://www.ge.ch/document/12832/telecharger.
[2] See https://chvote2.gitlab.io
[3] See https://gitlab.com/openchvote

# Item 17: Software Announcement

To continue the CHVote project independently of the support from the State of Geneva, a new funding from *eGovernment Switzerland* has been acquired by the Bern University of Applied Sciences in August 2019. The main goal of this project was to release a final stable version of the specification document and to update the cryptographic core of the protocol based on the code released by the State of Geneva. As a first project deliverable, the current Version 3.0 of the specification document has been released in December 2019 [9]. At the time of writing this paper, the developed *OpenCHVote* software is not yet complete. Since the project is in its final stage, the code is expected to be released soon under a non-proprietary license.[3] The general purpose of the project is to make the achievements available to others for pursuing it further.

[1] For further details about the reasons for abandoning the project, we refer to the State Council's press statement at https://www.ge.ch/document/12832/telecharger.
[2] See https://chvote2.gitlab.io
[3] See https://gitlab.com/openchvote

## Lesson Learned

Don't announce releasing software before your are ready to release it

# Item 17: Software Announcement

but . . .

# Item 17: Software Announcement

- OpenCHVote v1.0 has been released yesterday on

  `https://gitlab.com/openchvote`

- Along with an updated CHVote Protocol Specification v3.1

  `https://eprint.iacr.org/2017/325`

- GNU Affero General Public License Version 3

- Funded by the State of Geneva (2016–2019) and the Federal Chancellery (2019–2020)

# Item 17: Software Announcement

## 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

# Item 17: Software Announcement

## Missing Features

- Software for performing universal verification (the Verifier)
- JavaScript implementation of the voting client
- Reliable and secure infrastructure services
- Other infrastructure software components

# Items 7 & 11:
# Pseudo-Code Algorithms

# Item 7: Pseudo-Code Algorithms

| 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** | | |
| $pk \leftarrow \mathsf{GetPublicKey}(\mathbf{pk})$ | | |
| $(\cdot, \cdot, \cdot, \cdot, \mathbf{n}, \cdot, \cdot, w_v) \leftarrow VP_v$ | | |
| $(\alpha, \mathbf{r}) \leftarrow \mathsf{GenBallot}(X_v, \mathbf{s}, pk, \mathbf{n}, w_v)$ | | |
| | $\xrightarrow{v, \alpha}$ | |

# Item 7: Pseudo-Code Algorithms

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

# Item 7: Pseudo-Code Algorithms

**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** $(\alpha, \mathbf{r})$ // $\alpha \in \mathbb{G}_{\hat{q}} \times (\mathbb{G}_q^2)^k \times (\mathbb{Z}_{2^\tau} \times (\mathbb{Z}_{\hat{q}} \times \mathbb{G}_q \times \mathbb{Z}_q))$, $\mathbf{r} \in \mathbb{Z}_q^k$

# Item 7: Pseudo-Code Algorithms

**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** $(\alpha, \mathbf{r})$ // $\alpha \in \mathbb{G}_{\hat{q}} \times (\mathbb{G}_q^2)^k \times (\mathbb{Z}_{2^\tau} \times (\mathbb{Z}_{\hat{q}} \times \mathbb{G}_q \times \mathbb{Z}_q))$, $\mathbf{r} \in \mathbb{Z}_q^k$

# Item 11: Implementation of Algorithms

```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);
    }
}
```

# Item 11: Implementation of Algorithms

```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);
    }
}
```

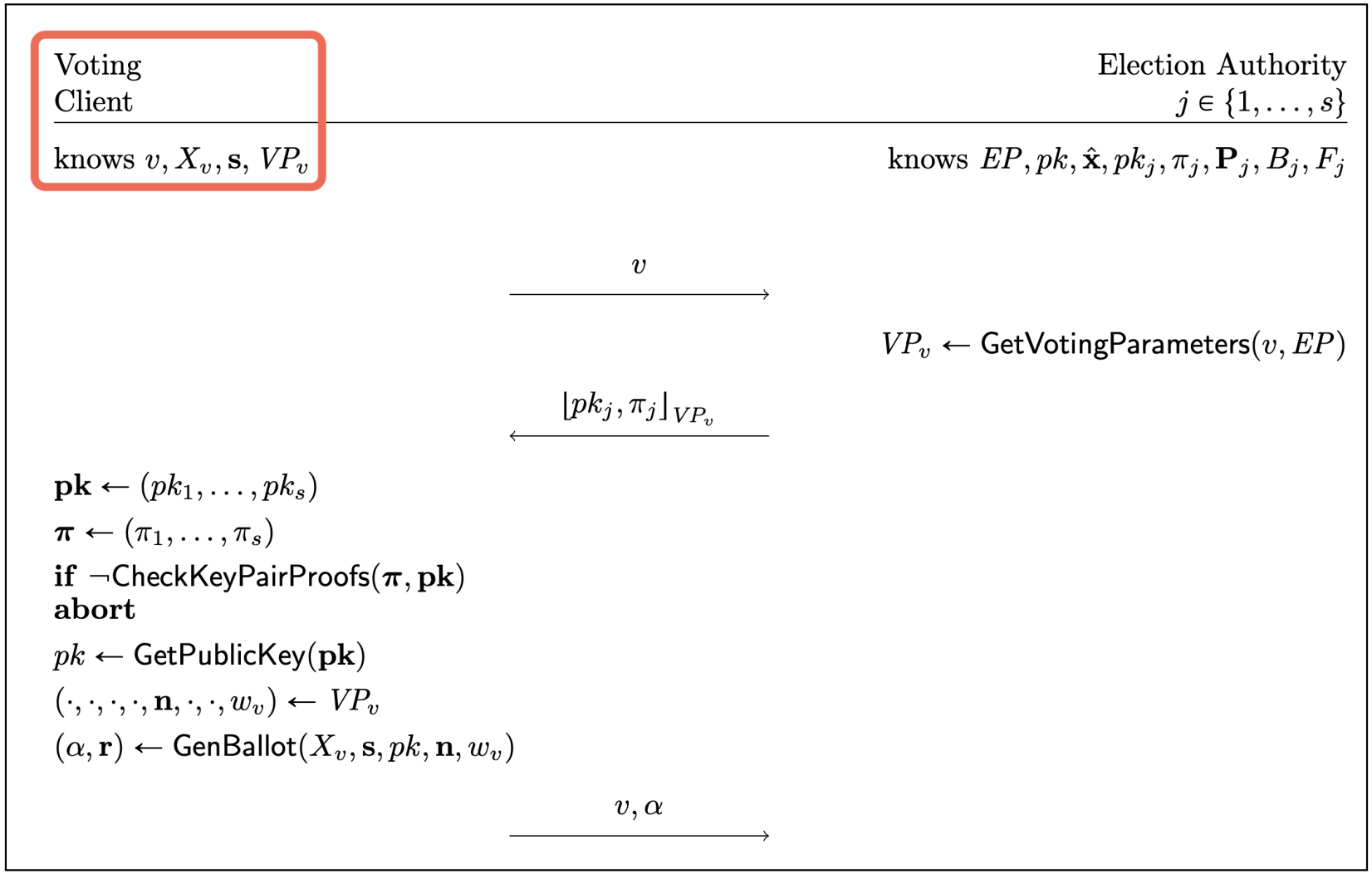
# Item 11: Implementation of Algorithms

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

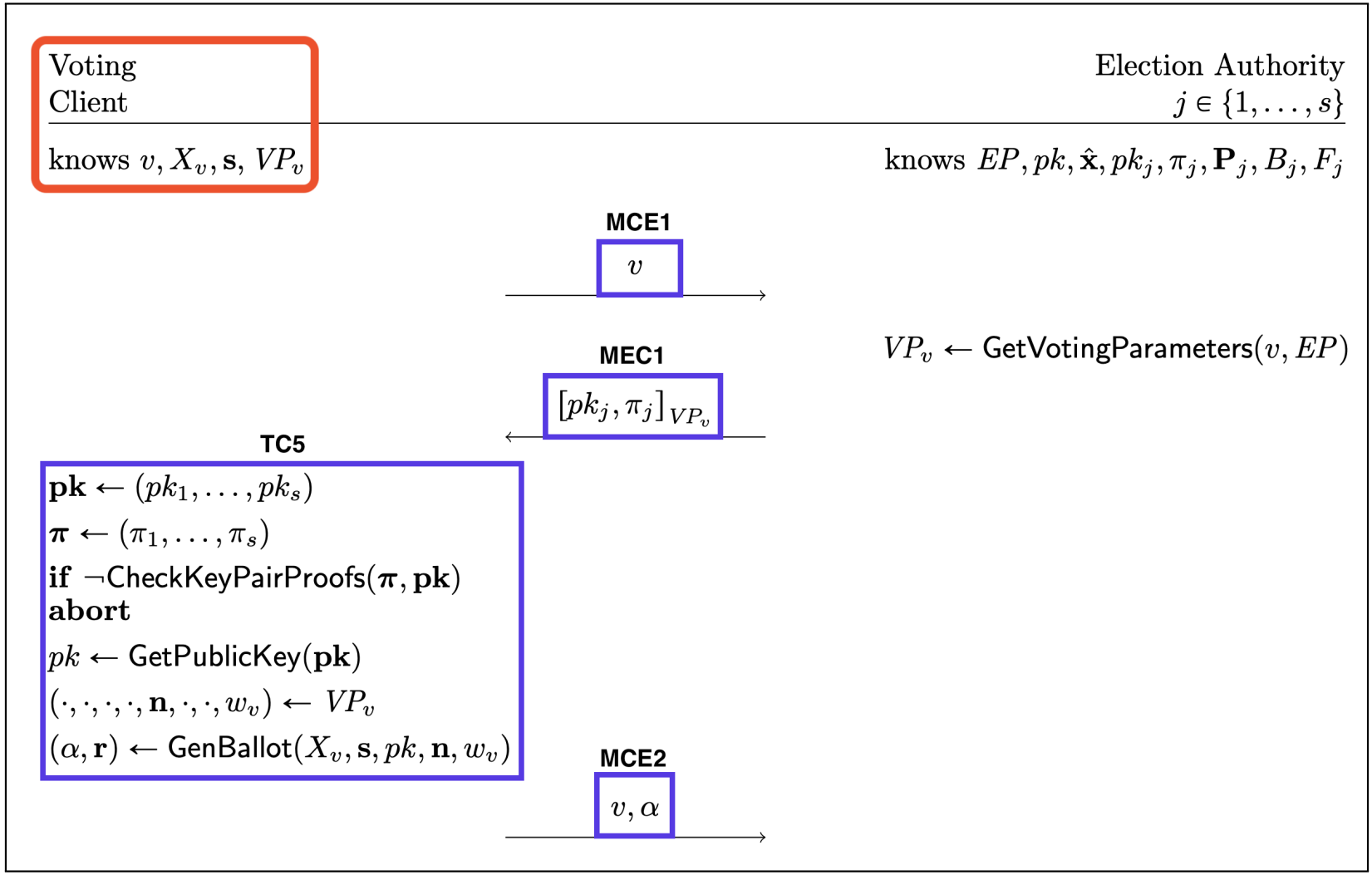
```
var x = ToInteger.run(X, params.A_X);
var x_hat = Mod.pow(params.g_hat, x, params.p_hat);
var bold_p = GetPrimes.run(n + w, params);
var bold_m = GetEncodedSelections.run(bold_s, bold_p);
var m = Math.prod(bold_m.map(QuadraticResidue::getValue));
```

# Item 13: Protocol Parties

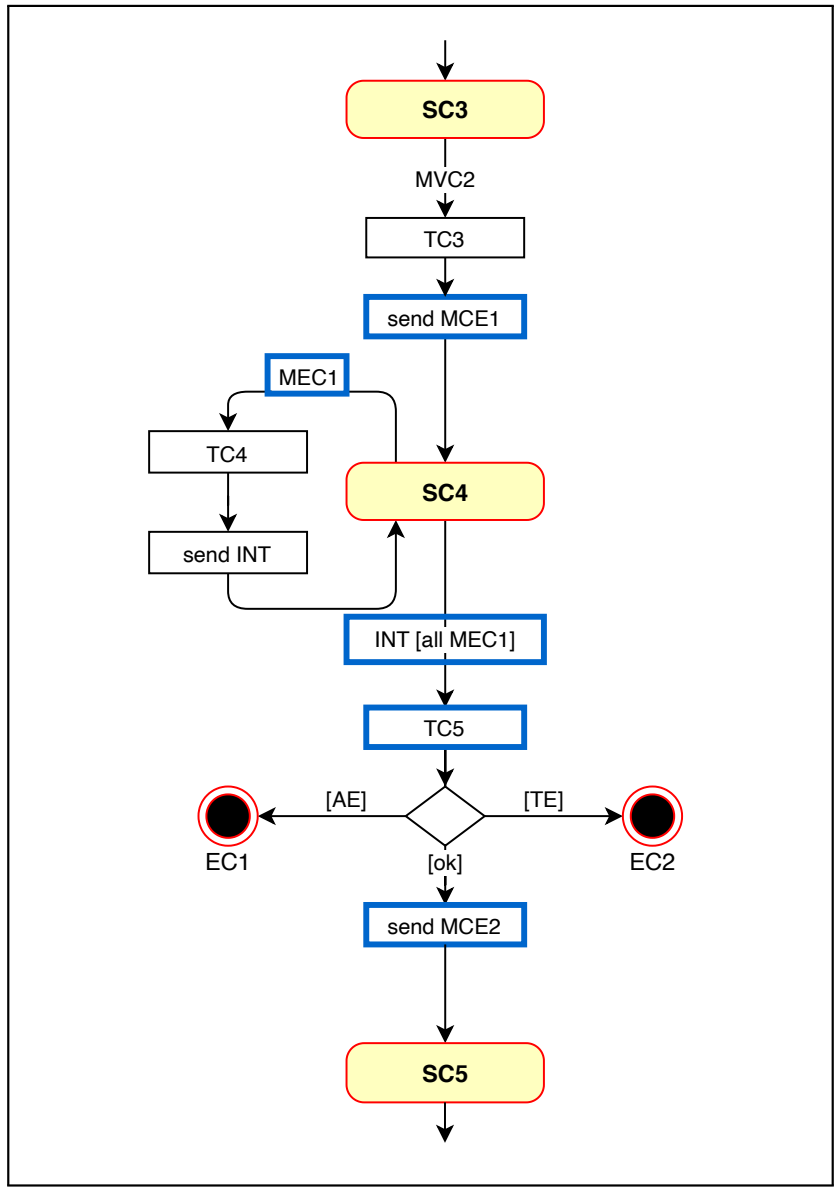
# Item 13: Protocol Parties



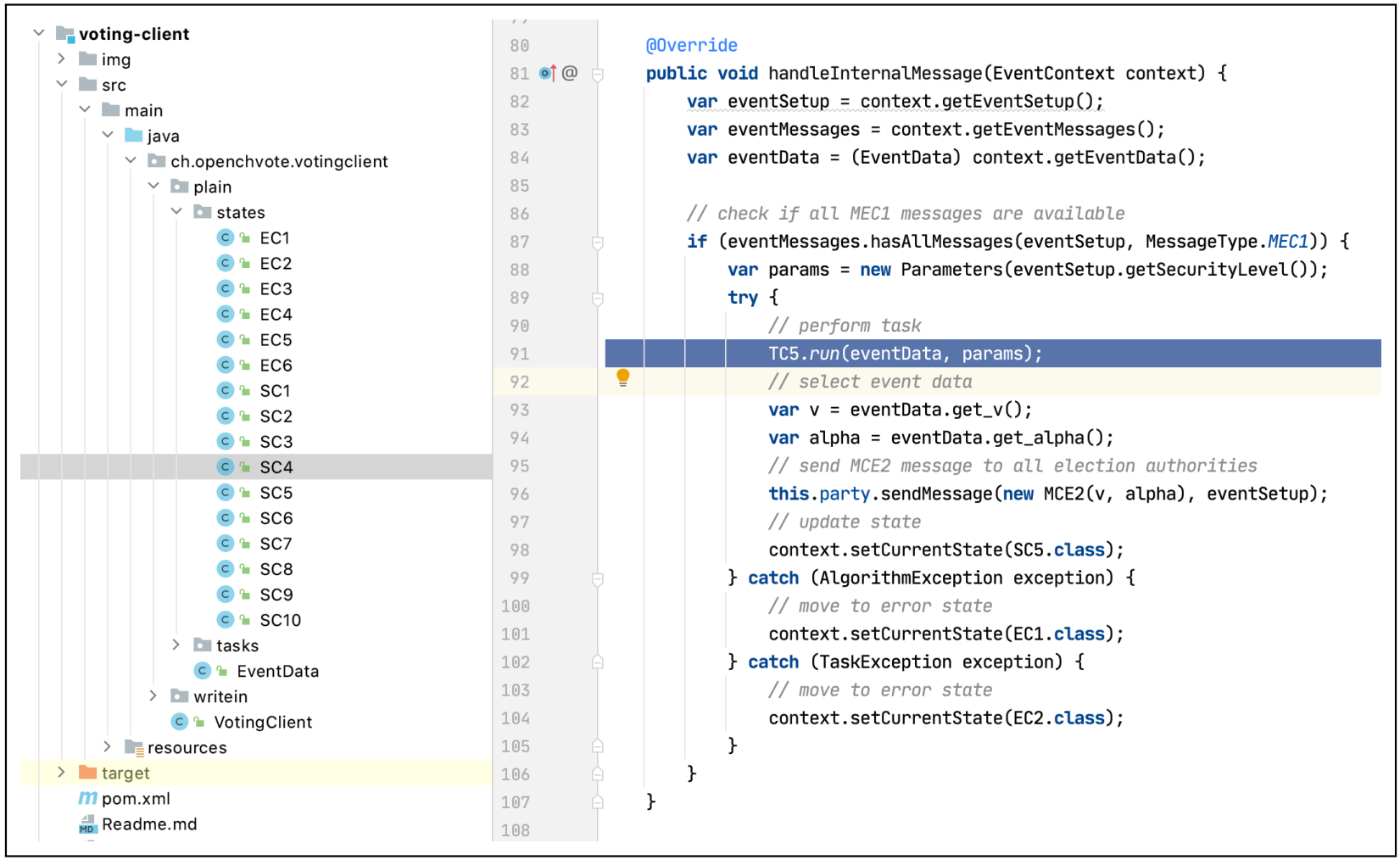
# Item 13: Protocol Parties



# Item 13: Protocol Parties



# Item 13: Protocol Parties



# Item 13: Protocol Parties

```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);
    }
```

# Item 13: Protocol Parties

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

```
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();
```

# Conclusion

# Conclusion

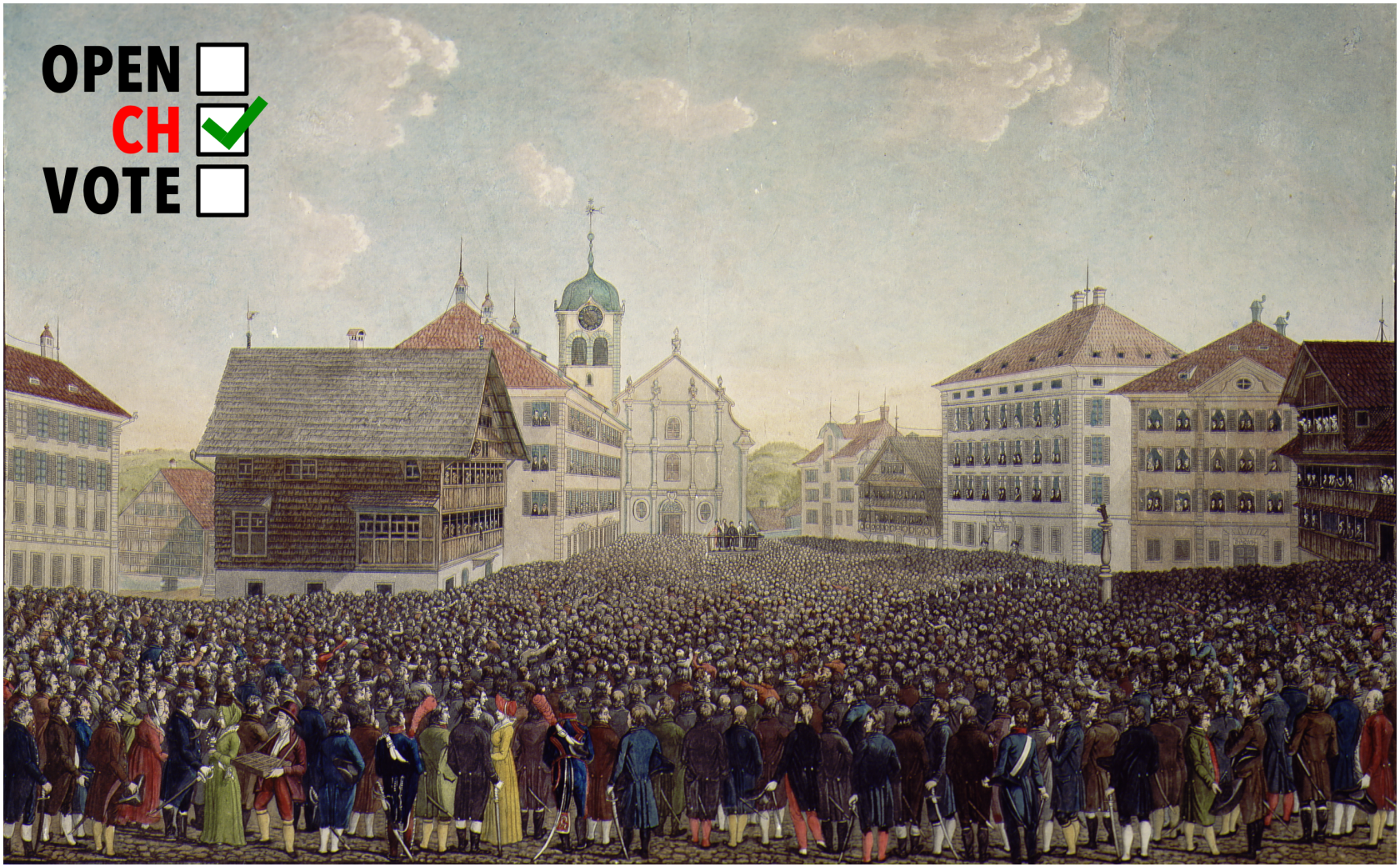
## Achievements

- CHVote: Consistent and comprehensive protocol specification
- OpenCHVote: Perfectly matching Java implementation
- Cryptographically relevant aspects fully covered
- Flexible software design
- Clean production-quality code

## Next Steps

- Public scrutiny
- Integration of third-party contributions
- Collaborations with third parties on extensions and open issues

# Questions?




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