Swiss Post E-Voting

• Presentation by Ella Kummer

• Supervisor : Louis-Henri Merino
• Professor Bryan Ford
• DEDIS Laboratory
Plan of the presentation

1. Quick overview of the Swiss post e-voting system
2. Goal of the project
3. Summary of the reviews
3. Interesting reviews
4. Conclusion
SECURITY OBJECTIVES

- **Individual verifiability**: Allows the voter to check that the vote was correctly transmitted and registered by the server by comparing a verification code that they receive with their voting documents to the verification code displayed online when they go to the ballot box.

- **Universal verifiability**: Allow voters or auditors to check that the election outcome corresponds to the registered votes using advanced cryptographic techniques such as non-interactive zero-knowledge proofs and verifiable mix-nets.

- **Vote secrecy**: Do not reveal a voter’s vote to anyone. It preserves the privacy of the voter by encrypting votes end-to-end and splitting the decryption key among multiple entities.
DECOMPOSITION OF THE SYSTEM:

This application contains several microservices. Each microservice is responsible for one part of the voting process, i.e., authentication, election information, vote verification, etc. It is considered untrustworthy.

A library that provides key sharing and encryption capabilities to the voting protocol. It prevents incorrect, unsafe or insecure usage of cryptography algorithms and providers. It has a single-entry point that is configurable.

An open-source server side library which implements cryptographic algorithms used as building blocks for the voting protocol. Focuses on the verifiable mix net and non-interactive zero-knowledge proofs.

The Control Components compose a system in which they work together as a group.

There are two types of control components: the Return Codes Control Component (CCR) and the Mixing Control Component (CCM).

They generate the return codes, shuffle the encrypted votes, and decrypt them at the end of the election while guaranteeing the integrity of the voting protocol.

At least one of them must be trustworthy while three of them might be under an adversaries’ control.
PHASES

The cryptographic protocol divides the Swiss Post Voting System’s "runtime" into three parts:

1. **Configuration Phase**: Generates the voter’s codes that are subsequently sent to the voter by postal mail and generates the election public key that is used for encrypting the votes.

2. **Voting Phase**: First authenticates the voter. Then the voter can select the desired voting options and ensure individual verifiability, thus the vote can get confirmed.

3. **Tally Phase**: The voting server and the mixing control components decrypt the votes and compute the election result while protecting vote secrecy and guaranteeing universal verifiability.
GOAL OF THE PROJECT

• Review the source code and the documentation to look for potential vulnerabilities and security issues.

• Methodology:
  • Target a part of the system *(e.g. building block)* or part of the protocol *(e.g. phase)* for specific reasons *(e.g. complexity, newness)*
  • Analyse the documentation
    • Correct results
    • Correct security (also related to scheme used)
  • Analyse the implementation
SUMMARY OF THE REVIEWS
REVIEWED CODE

CRYPTOGRAPHIC IMPLEMENTATION:

• Multi-recipient ElGamal scheme
• Digital signatures (RSA-PSS)
• Randomness generation
• Hashing (SHA-256)
GenKeysCCRj:
computes key pairs to later encrypt or derive new keys.

GenEncLongCodeShares:
creates shares of the long return codes.

SetupTallyCCMj:
computes the election key pair for each control component.
**REVIEWED CODE**

**VOTING PHASE – CONFIRM VOTE IMPLEMENTATION**

**CreateConfirmMessage**: create a confirmation key.

**CreateLVCCSharej**: derives codes for the next step.

**ExtractVCC**: extracts a code to allow the voter to check that it is identical to the code printed on their voting card.
REVIEWED CODE
TALLY PHASE IMPLEMENTATION

**GenVerifiableShuffle** :
shuffles and re-encryts the votes and proved a proof of the shuffle.

**GenVerifiableDecryptions** :
provides a verifiable partial decryption of a list of encrypted votes.

**VerifyShuffle** :
verifies the correctness of the shuffle argument.

**VerifyDecryptions** :
verifies the correctness of the decryption.
FINDS

✓ Necessary checks on inputs/data
✓ Groups operations
✓ Collision resistance for hash functions
✓ No insecure libraries
✓ Signed data
✓ Encryptions

○ Java best practices
○ Inconsistency between the documentation and the theory
FINDS

• Secure implementation of the cryptographic schemes
  ✓ El Gamal encryption
  ✓ Schnorr protocol for zero-knowledge proof (using Fiat-Shamir trick)
  ✓ Pedersen commitment scheme
  ✓ Bayer-Groth mixnet
FINDS

• Rely on java for
  ▪ Randomness generation
  ▪ Primality tests
  ▪ Digital signature
  ▪ Partly for hashing

~ security
INTERESTING DETAILED REVIEWS
RANDOMNESS GENERATION
- UPDATE

Class SecureRandom

java.lang.Object
java.util.Random
java.security.SecureRandom

: provides a cryptographically strong random number generator (RNG)
public BigInteger genRandomInteger(final BigInteger upperBound) {
    checkNotNull(upperBound);
    checkArgument(upperBound.compareTo(BigInteger.ZERO) > 0, "The upper bound must a be a positive integer greater than 0.");
    final BigInteger m = upperBound;

    final int bitlength = m.bitLength();

    BigInteger r;
    do {
        // This constructor internally masks the excess generated bits.
        r = new BigInteger(bitlength, secureRandom);
    } while (r.compareTo(m) >= 0);

    return r;
}

public BigInteger genRandomIntegerUpperBounded(BigInteger upperBound) {
    checkNotNull(upperBound);
    checkArgument(upperBound.compareTo(BigInteger.ZERO) > 0);

    int length = upperBound.bitLength();

    BigInteger random;
    do {
        random = genRandomIntegerByBits(length);
    } while (random.compareTo(upperBound) >= 0);

    return random;
RANDOMNESS GENERATION
- UPDATE

- Some cryptographic primitives are implemented both in the crypto-primitives and the cryptolib (for instance the ElGamal encryption scheme). The implementations are functionally equivalent. We are continuously replacing the cryptolib implementation with the more robust crypto-primitives one.

https://gitlab.com/swisspost-evoting/e-voting/e-voting
MixDecOnline\(j\) function

The online Mixing control components CCM shuffle and re-encrypt the previous control component’s ciphertexts and perform partial decryption.

If the control component is the first to mix \((j = 1)\), the input list of ciphertexts corresponds to the cleansed encrypted votes.

---

Algorithm 6.2 MixDecOnline\(j\)

**Context:**
- Group modulus \(p \in \mathbb{P}\)
- Group cardinality \(q \in \mathbb{P}\) s.t. \(p = 2q + 1\)
- Group generator \(g \in G_1\)
- Election event ID \(ee \in (\mathbb{A}_{\text{mix}})^{\Delta^\ast}\)
- Ballot box ID \(bb \in (\mathbb{A}_{\text{mix}})^{\Delta^\ast}\)
- Control component index \(j \in [1, 3]\)
- Number of allowed write-ins + 1 for this specific ballot box \(\delta \in \mathbb{N}^\ast\)
- List of shuffled and decrypted ballot boxes \(\mathcal{L}_{\text{dec}}\)

**Input:**
- Partially decrypted votes \(c_{\text{dec}, j-1} \in (\mathbb{Z}_q)^\nu\)
- Remaining election public key \(\mathcal{E}_{\text{pub}, j-1} \in \mathcal{G}_1^\prime\)
- CCM\(j\) election key pair \((\mathcal{E}_{\text{pub}, j}, \mathcal{E}_{\text{sk}, j}) \in \mathcal{G}_1^\prime \times \mathcal{Z}_q^\nu\)

**Ensure:** \(\mathcal{E}_c \geq 1\)
- \(l = \delta\) \(\triangleright \) The algorithm runs with at least one vote
- \(0 < \delta \leq \mu\)

**Ensure:** \(bb \notin \mathcal{L}_{\text{dec}}\)

**Operations:**
1. \(\mathcal{L}_{\text{dec}} \leftarrow \text{"MixDecOnline"}, \text{IntegerToString}(j)\)
2. If \(\mathcal{E}_c > 1\) then
3. \((c_{\text{dec}, j}, \mathcal{E}_{\text{sk}, j}) \leftarrow \text{GenVerifiableShuffle}(c_{\text{dec}, j-1}, \mathcal{E}_{\text{pub}, j-1})\) \(\triangleright\) See crypto primitives specification
4. \((c_{\text{dec}, j}, \mathcal{E}_{\text{sk}, j}) \leftarrow \text{GenVerifiableDecryption}(c_{\text{dec}, j}, \mathcal{E}_{\text{pub}, j}, \mathcal{L}_{\text{dec}}, \mathcal{L}_{\text{sk}, j})\) \(\triangleright\) See crypto primitives specification
5. Else
6. \((c_{\text{dec}, j}, \mathcal{E}_{\text{sk}, j}) \leftarrow \text{GenVerifiableDecryption}(c_{\text{dec}, j-1}, \mathcal{E}_{\text{pub}, j}, \mathcal{L}_{\text{dec}}, \mathcal{L}_{\text{sk}, j})\)
7. End if
8. \(\mathcal{E}_{\text{pub}, j} \leftarrow \text{CompressPublicKey}(\mathcal{E}_{\text{pub}, j}, \delta)\) \(\triangleright\) See crypto primitives specification
9. \(\mathcal{E}_{\text{pub}, j} \leftarrow \mathcal{E}_{\text{pub}, j-1} \mod p\)
10. \(\mathcal{L}_{\text{dec}} \leftarrow \mathcal{L}_{\text{dec}} \cup bb\)

**Output:**
- Shuffled votes \(c_{\text{dec}, j} \in (\mathbb{Z}_q)^\nu\)
- Shuffle proof \(\pi_{\text{shuf}, j}\)
- Partially decrypted votes \(c_{\text{dec}, j} \in (\mathbb{Z}_q)^\nu\)
- Decryption proofs \(\pi_{\text{dec}, j} \in (\mathbb{Z}_q \times \mathbb{Z}_q)^\nu\)
- Remaining election public key \(\mathcal{E}_{\text{pub}, j} \in \mathcal{G}_1^\prime\)
The algorithm accepts the case where only one vote is submitted to decryption.

This could be considered as a security issue as this case does not preserve the voter anonymity.

As there is only one vote, no shuffling can be performed to break the link between the ciphertexts and the plaintexts and thus the voter.
MixDecOnline function

• This is a special case not likely to happen as during a vote there is a very low chance of having only one voter.

• However, for the sake of security it should be acknowledge. Either it needs to be modified*, or they need to make clear that they allow this case which removes some anonymity.

* e.g. enforce to receive at least two ciphertexts to ensure 2-anonymity and have only 50% of chance of linking the voter to their vote.
MixDecOnline function

✓ Thankfully, as the verifier is trustworthy, it is not possible for an active attacker to try to reduce the list of ciphertexts in order to use this condition as an oracle to ask for the decryption of a specific ciphertext.

During the shuffling (and decryption) proof, the original vector of unshuffled ciphertexts is used as input, a difference between the lists’ sizes would be directly detected by verifier.
“The Swiss Post Voting System uses the RSA-PSS signature scheme with 2048-bits key length “
“The Signature class is used to provide applications the functionality of a digital signature algorithm.

The signature algorithm can be, among others, the NIST standard DSA, using DSA and SHA-256.

These algorithms are described in the Signature section (*) of the Java Cryptography Architecture Standard Algorithm Name Documentation.”

https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#Signature

The signature algorithm with SHA-* and the RSA encryption algorithm as defined in the OSI Interoperability Workshop, using the padding conventions described in PKCS #1 (Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1)
CONCLUSION
CONCLUSION

• The system is still under development and some future work is planned

• Through the work done in this project, we have a high level of confidence on the security of the Swiss post e-voting system