Poster: A Secure and Verifiable Electronic Voting System

Francisco Kajatt-Vaccari (student), Tamara Finogina (student), Panagiotis Karras (faculty)
Skolkovo Institute of Science and Technology
Skolkovo, Moscow, Russian Federation

***Abstract*—We propose an architecture for an electronic voting system that satisfies the following three requirements: Integrity, Verifiability, and Anonymity. We keep information about user votes and user identity in different tables, encrypt them, and hide connections between those tables. Encryption techniques used in the system include oblivious RAM and fully (or partly) homomorphic encryption.**

1. Introduction

Nowadays, the need to securely collect people's votes on different social and political questions has spawned interest in the development of secure means for carrying out elections and surveys over the internet [1], [2], [3].

Unfortunately, extant electronic voting systems focus on ensuring confidentiality and anonymity, yet do not address the challenge of verifiability. In some cases, a large subset of the voting population has to participate in an auditing process in order to ensure verifiability. In others, users are unable to check their votes after casting them.

For instance, the Helios system [1] aims to provide a complete solution for an electronic voting system. However, Helios ensures anonymity by reshuffling votes after they have been collected, then proving that only order has been altered. Yet, if the system is intercepted before reshuffling, anonymity is compromised; thus, anonymity is not ensured throughout the voting process. Besides, a compromised or corrupted Helios system can insert votes for users who decided not to cast one, and even impersonate a voter. Last, Helios can provide verification only when a large majority of the voters verify their vote [4].

An recent interesting proposal was presented in [2]; this system guarantees vote integrity and also allows a voters to verify that their vote has been correctly counted, while keeping each vote secret. However it still leaves some unaddressed questions. In particular: (i) a vote has a unique ID; thus, it remains anonymous only as long as this ID is secret; (ii) verification based on paper proof, which can be fabricated; (iii) a voter can verify only her own vote, without a possibility to check the integrity of the whole result.

We aim to design a system addressing all aforementioned deficiencies: guaranteeing anonymity in all phases and allowing any single voter to verify the integrity of all votes and the voting result. In more detail, we stipulate that an electronic voting system must comply with three main requirements: voter *anonymity*, vote *integrity* and result *verifiability*; voter anonymity implies that a certain vote should remain secret in the face of internal or external adversaries trying to intercept the communication; vote integrity implies that each vote is guaranteed to be, and can only be, counted uncorrupted in the final tally; result verifiability implies that any voter should be able to verify that their own vote has been counted and that the full result is correct.

2. Proposed Architecture

Our architecture features three distinct entities participating in the election process: The Voting Population who submit votes (Voters), a Central Electoral Commission (Commission) who oversees the election, and a Central Server (Server) that stores and calculates results upon request. The Server stores four tables: a *Voters* table, a *Votes* table accessible via Oblivious RAM, a WORM (write-once-read-many) *Hashes* table, and a *Public Keys (PK)* table. The Commission generates fully homomorphic keys [5] and makes the encryption key public. Each user holds a pair of asymmetric encryption keys to be used throughout the procedure (pPK, pSK). The first two tables are shown in Table 1 and 2.

TABLE 1: Voters

Field	Description
UID	A unique identifier for users.
encrypt(pPK, $v_0, \ldots, v_n$)	Encrypted vote by the voter using asymmetric public key; can be more than one field depending on how a vote is defined.
encrypt(pPK, index)	Encrypted index pointing to relevant entry in Votes Table, using voter's public key.
hash(.)	Hash of all previous fields.
encrypt(pSK, hash(.))	Encryption of hash using voter's secure key.

3. Voting Procedure

The Commission generates a pair of public and private fully homomorphic keys (fhPK and fhSK), shares the public one (fhPK), make a list of all eligible voters, and add their public keys in the *PK* table.

A user who registers in the voting system receives from the server an index for the *Voters* table, to be encrypted using his public key in the PK table. The server reserves entry slots in the *Votes* table and *Hashes* table, which both contain

TABLE 2: Votes

Field	Description
encrypt(fhPK, v_0)	Encrypted value for vote 0.
...	Fields reserved for more votes encrypted as above.
encrypt(fhPK, v_n)	Last vote field.
encrypt(fhPK, $data_0$)	Encrypted data associated to vote, e.g., voter's ZIP code.
...	Fields reserved for more vote-associated data.
encrypt(fhPK, $data_m$)	Last data field.
index	Index that points to record in Integrity Verification table.

vote-related information. Indices in the *Voters* table bear no connection to those in the *Votes* and *Hashes* tables. A vote, cast in the form $\{index, v_0, \ldots, v_n, data_0, \ldots, data_m\}$, is encrypted using the commission-provided public homomorphic key (fhPK), and then sent to the *Votes* table. We reiterate that the server can access this *Votes* table only via *Oblivious RAM*, so as to preserve anonymity even in the face of a collusion between Server and Commission. Such a collusion may allow for votes to be decrypted, yet, even then, the Server will be able to link a record in the *Votes* table to its corresponding record on the *Voters* table.

The server also hashes each received vote, puts the result in its reserved entry in the WORM *Hashes* table, and stores that index on the *Vote* table. Last, a voter enters all other vote-related data, which are encrypted using the voter's public key, hashed, and signed using the voters's private key to verify the user's participation, before being stored in the *Voters* table.

4. Verification Procedure

The Voters table can be used to check vote uniqueness by checking a voter's signature (Table:*Voters*, field: *Encryption of hash using voter's secure key.*). Voters can also check the integrity of their vote against potential collusion between commission and server. Using the privately signed hash we can verify the voter's identity. Every voter can also retrieve her vote from the Votes table, decrypt it with his pSK, encrypt it again with the commission's fhPK and compare it with the vote stored in the Votes table. To access information in the Votes table, a voter gets the encrypted index of her record in the Voters table, decrypts it, and sends a query for that record to the Votes table. We reiterate that establishing association via access pattern will not be possible due to the use of Oblivious RAM [6], [7], [8], [9]. A voter who finds out that the results do not match concludes that her was corrupted, and can call the election invalid.

We emphasize that, even in case of collusion between commission and server, the *Voters* table cannot be decrypted and information about identity cannot be revealed without the user's private key. Still, our architecture provides to any voter the ability to check the integrity of any other voter's vote, by comparing each line in the *Votes* table to its corresponding hash on the WORM *Hashes* table. If they do not match, vote integrity has been violated. Besides, the server cannot learn statistical information that could reveal data from subsets of the voting population from the *Votes* table, since the decryption of any computation results requires the commission's homomorphic secure key (fhSK).

Last, to verify the overall election result, a voter can arbitrarily select two numbers, say T and F, encrypt them to obtain fhPK(T) and fhPK(F), ask the server to return the homomorphically encrypted calculation outcomes RESULT + fhPK(T) and RESULT + fhPK(F), where RESULT is the election result on the votes, and then ask the commission to decrypt the server output; the commission returns the plain values A and B; then, the voter can verify the correctness of the reported RESULT by testing that $A - T = B - F$. In case that server cannot be trusted to compute the result correctly, we can verify the integrity of the *Votes* table using mechanisms described in previous sections, perform the calculation on another machine, and continue as described with the commission.

5. Conclusion

We outline how extant encryption schemes can be used so as to design an electronic voting system that provides for confidentiality and verifiability beyond extant solution. Our proposed architecture allows voters to ensure the confidentiality of their votes, preserve their anonymity, and also verify all aspects of the voting procedure. We have implemented the core features of our scheme and current work on the ORAM module. In the future, we plan to deploy a fully working prototype as open source, and also study the question of *participation privacy*, i.e., the privacy of the information on whether a voter has voted or not.

References

[1] B. Adida, "Helios: Web-based open-audit voting," in *USENIX Security Symposium*, 2008, pp. 335–348.

[2] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia, "Prêt à voter: A voter-verifiable voting system," *Trans. Info. For. Sec.*, vol. 4, no. 4, pp. 662–673, 2009.

[3] S. Heiberg, A. Parsovs, and J. Willemson, "Log analysis of Estonian internet voting 2013-2014," in *VoteID*, 2015, pp. 19–34.

[4] O. Kulyk, V. Teague, and M. Volkamer, "Extending helios towards private eligibility verifiability," in *VoteID*, 2015, pp. 57–73.

[5] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[6] S. Kamara, "Outsourced bits: A research blog on cloud computing, cryptography, security, privacy..." [Online]. Available: http://outsourcedbits.org

[7] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious RAM protocol," in *CCS*, 2013, pp. 299–310.

[8] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[9] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Privacy-preserving group data access via stateless oblivious RAM simulation," in *SODA*, 2012, pp. 157–167.