How not to Use a Privacy-Preserving Computation Platform: Case Study of a Voting Application

Jan Willemson[1,2]

[1] Cybernetica AS, Ülikooli 2, 51003 Tartu, Estonia
jan.willemson@cyber.ee
[2] STACC, Ülikooli 2, 51003 Tartu, Estonia

Abstract. We present an analysis of a recent proposal by Dang-awan *et al.* who develop a remote electronic voting protocol based on secure multi-party computation framework Sharemind. Even though Sharemind comes with provable security guarantees and an application development framework, the proposed protocol and its implementation contain a number of flaws making the result insecure. We hope this case study serves as a good educational material for future secure computation application and voting protocol developers.

Keywords: secure computation, electronic voting, protocol analysis

1 Introduction

Data processing is a field offering both threats and opportunities. On one hand, having access to larger amount of high-precision data allows to take better-informed policy decisions and as a result increase the quality of life of the whole society. On the other hand, having access to personal information may give rise to malicious profiling, manipulation, blackmailing or other types of misuse.

Thus a balance is required between the two extremes of making all data public (destroying individual privacy) and closing all data up (destroying data utility).

One possible equilibrium is provided by *secure computation* mechanisms that allow to generate aggregate results while still protecting the individual records. Being originally proposed in early 1980s, the respective methods have evolved over the decades, resulting in a number or practically applicable frameworks. Some of the currently actively developed examples include Sharemind [7], SPDZ [9], ObliVM [21], Chameleon [26], etc.

Unfortunately, building an application on top of such a framework does not yet guarantee that the application itself is secure. There are many places where things can go wrong if done carelessly.

A tempting area to deploy secure computation mechanisms is electronic voting. At the first sight, its problem setting closely resembles the one of secure computation. A potentially large number of voters each have their private input (political preference), and their joint interest is to compute an aggregate end result (the tally).

The major approach to join the two worlds has been using homomorphic encryption which has been studied and developed since 1980s [3, 4]. However, this approach is quite limited in the choice of voting protocols it is able to implement. Since homomorphic encryption allows performing only one kind of protected operation (say, addition), the resulting protocols can not go much further from simple vote counting. Implementing more involved paradigms like preferential voting or supporting more complex ballots becomes very challenging.

Deploying a fully-fledged secure computation framework as a basis for a flexible electronic voting solution is a natural idea. However, until recently there have been only a few incomplete attempts in this direction. In [13], Gang merely states the basic idea, and in [24], Nair *et al.* implement a simple Java application to add secret-shared votes. Gjøsteen and Strand take a different approach utilising recent advances in fully homomorphic encryption [16]. However, fully homomorphic encryption is still too inefficient to be applied on a large scale.

The first fully functional solution for secure computation based electronic voting was proposed and implemented by Dang-awan *et al.* in 2018 [10]. They have built their proof-of-concept application on top of Sharemind[3] [7], an established secure computation platform with good development tools, including high-level SecerC programming language for creating secure applications [6]. Regrettably, Dang-awan *et al.* made a number of mistakes in several stages of design and development, resulting in a completely insecure application.

In this paper, we will be going over their main flaws. Besides the direct protocol-analytic value, we find the result also very educational for the developers of both secure computation and electronic voting applications.

The paper is organised as follows. We begin with a short overview of the state of the art in both secure computation and electronic voting in Section 2, followed by a general overview of the system architecture of Dang-awan *et al.* [10] in Section 3. Next we analyse both the voting and

[3] https://sharemind.cyber.ee/

tallying processes in Sections 4 and 5, respectively. Finally, we draw some conclusions in Section 6.

2 State of the Art

Both secure computation and electronic voting domains have been actively studied for decades.

The idea and first protocols for secure computation come from Yao in early 1980s [30]. Yao built his protocols around the garbled circuits paradigm. In the later research, also other algebraic primitives like secret sharing [28] and fully homomorphic encryption (FHE) [14] have been proposed as the basis for secure computation. We refer to [8] and [23] for good recent surveys on the respective topics.

Even though FHE, in principle, supports outsourcing of arbitrary computations, the implied overhead of current implementations is still too large for even medium-size computations [2]. Garbled circuits are much faster, but assume a lot of bandwidth between the computing nodes. All in all, secret-sharing-based secure computation frameworks provide currently the best trade-off between performance and security properties [2].

The idea of using electr(on)ic means for vote recording is almost as old as human usage of electricity. On June 1, 1869, Thomas A. Edison was awarded U.S. Patent number 90,646 for an "electrographic vote-recorder" which he envisioned to be used in U.S. Congress elections. In 1870s, several proposals to use electric machinery to record votes were made in France [19]. Since then, mankind has experimented with various vote casting assistants including Direct Recording Electronic machines and Internet voting. We refer to Robert Krimmer's PhD thesis for a good historical overview of all the relevant developments and their societal context [19].

By early 2000s, Internet had become the primary means of data transfer. It was a natural question to ask whether votes could also be cast via Internet during elections. For example, there was such an experiment performed in 2000 Arizona presidential primaries [1]. The first legally binding country-wide elections with Internet voting as an option were organised in Estonia in 2005 [22]. Since then, several countries (e.g. Norway, Switzerland and Australia) have experimented with various approaches.

One of the main challenges with any kind of elections is ensuring integrity of the result. To achieve this, all of the processes must be transparent and independently auditable, preserving vote privacy at the same time. There is an inherent contradiction between these two requirements,

making finding a trade-off a delicate task (see e.g. [25] for a good overview of recent research in this direction).

The scheme of Dang-awan *et al.* [10] also tries to establish an equilibrium between privacy and verifiability. Unfortunately, it achieves neither of these properties.

3 System architecture

Remote electronic voting systems generally comprise of the following components.

- **Client software** working in the voter's environment and being responsible for displaying the options, getting the voter preference and securing it (by encrypting, signing and/or other means).
- **Voting server** being responsible for collecting and storing the votes (typically in a secured state).
- **Tallying server** is where the votes are opened (e.g. decrypted) and tabulated.

To prove that the required security properties of the system hold, frequently various **auditing components** are implemented in addition. In a more general sense, we can also consider operating systems, network connections, local legislation, etc. to be part of the picture, but out treatment will not go into these details.

Dang-awan *et al.* [10] start from the observation that securing the voting and tallying servers is a critical prerequisite for a trustworthy remote electronic voting system. Indeed, a breach in a server-side component has a potential to allow for a large-scale vote manipulation attack to be implemented unnoticed [29].

In order to decrease the need to rely on a single server, Dang-awan *et al.* propose to distribute the voting server between different parties and run secure multi-party computation (SMC) protocols between them to achieve the required functionality [10].

On the high level, a representative-based architecture is used [12] (see Figure 1).

In this architecture, voters act as (input) parties submitting their votes in a secret shared form to computation servers (also called *nodes*), of which there are three in the standard configuration of Sharemind. Each server also has a database back-end to store the shares, and a Node.js front-end to implement communication routines with the other parties.

Running applications in the distributed environment is managed by a special controller library. One example of such an application is the

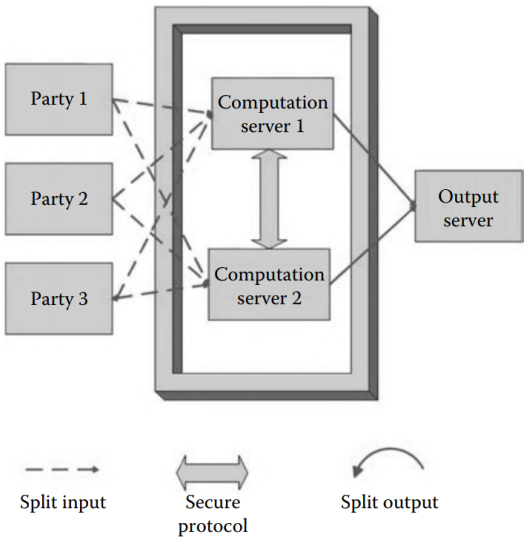

Fig. 1. Representative-based architecture for SMC [12]

tally triggered by the election organiser, with the aggregated end result designated for the output server. Note that the end result is computed and reaches the output server in secret shared form. In order to tabulate the result, it must be explicitly *declassified*, i.e. the shares of the output are combined according to the underlying secret sharing scheme to give the open aggregated tally.

Declassification operation is security-critical. In principle, it can also be applied on intermediate results or even input values, thus violating their privacy. On the other hand, this operation is required to obtain human-readable output of the computations, so we can not just prohibit it. Thus, declassification has to be invoked with great caution.

4 Voting

In the system proposed by Dang-awan *et al.*, vote casting is implemented via a Javascript library loaded into the browser. The library secret shares the vote (even though this operation is repeatedly confused with encryption in [10]) and sends the shares to Sharemind computing nodes.

The first problem we observe is that integrity of the votes is not protected by signatures or any other strong cryptographic mechanism. Instead, the authors propose a naïve cast-as-intended verification protocol.

The vote (consisting of the voter ID, position ID and candidate ID) is check-summed using CRC32, and this check sum is then hashed with SHA-1. The resulting hash is displayed to the voter as a receipt. After the vote shares are received by the Sharemind nodes, they perform the same operation and the resulting SHA-1 hash is returned to the voter for comparison. If the comparison succeeds, the voter should be assured of correct casting.

There are many flaws in this protocol. Perhaps the biggest conceptual problem is that all the communication between the voter and the central system (including displaying the check-sums) is performed through a single web browser. While this is definitely convenient, it creates a single point of attack. When the browser gets compromised (a scenario that is unfortunately very much possible), it can manipulate the displayed information arbitrarily. As a result, the voter can not distinguish whether the hash-check-sum displayed to her really matches the vote stored on the servers, or has it been maliciously changed before being shown to her.

It is exactly for this reason that remote cast-as-intended verification needs and independent channel. It may be implemented in various ways like the code sheets plus SMS as in Norway [15], or using an independent auditing device as in Estonia [17]. In any case it is clear that just relying on one medium for both vote casting and verification is not sufficient.

Second, applying SHA-1 after CRC32 does not add any security as SHA-1 is a deterministic function (but it does make the hashes longer, thus more difficult to compare by a human).

Third, no random salt is added before hashing. This means that the pre-images can be easily found by full inspection. To give some back-of-the-envelope estimates, let's consider the university student council election given as a use case by the authors of [10]. There would probably be about few thousand voters (say, up to 10,000), a few positions (say, about 10), and a few dozens of candidates (say, up to 100).

All in all, a vote has in the order of magnitude 10 million possible values. Since both CRC32 and SHA-1 are designed to be very fast to evaluate, pre-computing a table of 10 million hashes is an easy task. As all the hashes are put on a publicly accessible bulletin board (basically a webpage) for verification purposes, anyone with the pre-computed table can efficiently find out how everyone of the voters voted. This is definitely something that a well-designed election system should avoid in order to counter vote selling and other coercion attacks.

The double hashing proposed in [10] also leads to other problems. Note that the output of CRC32 is just 32 bits long and

$$\sqrt{2^{32}} = 2^{16} = 65536 \, .$$

This means that whenever the number of possible vote options

$$\#voters \times \#positions \times \#candidates$$

is roughly equal to or larger than 65536, there is a significant probability (50% or larger) of a collision due to the birthday paradox. Again, re-hashing the output with SHA-1 is useless against such collisions resulting already from CRC32.

What this means is that the server can present the same receipt to several voters whose choices happen to give a collision. This defeats the whole purpose of the cast-as-intended verification.

Ironically, the collisions may give some coercion-resistance to the scheme, but only in the case that the voter's true preference and the coercer's preference give the same CRC32 hash, which is in turn a very unlikely situation.

The way server-side verification hash is computed is also significant. Dang-awan *et al.* present the actual SecreC snippet that they have used (see Figure 2). Note that curly brace mismatch comes already from [10]; confusion of secret sharing and encryption is also evident here.

Note that all the vote components are explicitly declassified before computing the CRC32 hash. What this operation does is to make all these private values accessible to all the computing nodes. This defeats the whole purpose of using a secure computation platform in the first place, and thus constitutes the biggest flaw in the whole paper. Declassification should only happen in the very last stage of computation, i.e. only on the tally results.

The authors of [10] refer to unavailability of other hash functions on Sharemind platform other than CRC32 as the main reason for using it. All the above-mentioned problems away, it is unclear why they did not decide to at least use a privacy-preserving version of it in their implementation.

Limitations of the out-of-the-box API are, in general, a poor excuse for using insecure primitives. CRC32 was never intended as a cryptographic hash functions, it can merely capture stochastic transmission errors, but it can not withstand active collision and pre-image attacks. In case stronger cryptographic primitives (hash functions, signatures, etc.) are needed, Sharemind provides the developers with various tools (like

```
/**************************************
app_save_vote.sc:
How sent encrypted values are saved
**************************************/
for (uint i = 0; i < size(candidateId); ++i) {
    table = arrayToString(
     declassify(electionId[i]));
     tdbInsertRow(datasource, table,
         {voterId, positionId[i],candidateId[i]});
     print("ROW INSERTED");
     message = bl_str(
      arrayToString(declassify(voterId))
      +arrayToString(declassify(positionId[i]))
      +arrayToString(declassify(candidateId[i])));
     hash[i] = CRC32(message); // hashes returned
    }
}
```

Fig. 2. Vote recording script

SecerC domain-specific language [6]) for extending the API. Of course, there would be some performance overhead, but this is inherent in the case of secure computations.

5 Tally

The tallying procedure of elections is essentially a histogram computation, and this Dang-awan *et al.* actually implement in a privacy-preserving manner. However, privacy is not the only requirement of the tally process. Perhaps even more important is integrity, i.e. making sure that vote counting was not manipulated by anyone.

The biggest problem in [10] is using Sharemind in its out-of-the-box, three-server passive security mode. What passive security means here is the ability to withstand an attacker who is only observing one of the computing nodes, but is not trying to actively interfere with it. However, this model is too weak for the voting use case.

Just to give a small illustrating example, recall that standard Sharemind uses additive secret sharing [7], i.e. a value $x \in \mathbb{Z}_{2^{32}}$ is divided into shares $x_1, x_2, x_3 \in \mathbb{Z}_{2^{32}}$ so that

$$x_1 + x_2 + x_3 = x \bmod 2^{32} ,$$

where the share x_i is held by the computing party P_i. What a malicious party can do is e.g. increasing his share of one of the votes and decreas-

ing another at the same time, resulting in the same change in the values of the vote sums. This would lead to a serious voting result trustworthiness violation as any computing node would be able to manipulate it undetected.

Of course, tally integrity concerns are inherent in any election system. This is why a large variety of approaches towards verification have been proposed in the research community (see, e.g. [18] for a good overview). Ideally, tally correctness should be checkable by everyone, or at least by a large number of designated independent auditors. The proposal by Dang-awan *et al.* does not foresee any of such mechanisms.

Ironically, the ability to find pre-images of hashes displayed on the bulletin board gives a way for anyone to compute the tally independently. However, this happens with the price of total vote privacy loss, which is something we do not want either.

If a multi-party computation engine like Sharemind is used to implement the voting server, measures ensuring security against active manipulation attacks should be deployed [20, 11].

On top of that, modern electronic voting systems target *software independence*, a state of affairs where verifying security properties of the system should not rely on assumptions about the underlying software platform [27]. A practical way of approaching this target is requiring strong independently verifiable cryptographic audit trail of all the critical operations.

In the case of vote tallying, there are two main approaches proposed and implemented that can achieve this property. First, homomorphic tallying allows combining votes under encryption so that the end result is the encryption of the final count. Second, votes can also be directly decrypted giving non-interactive zero-knowledge proofs of decryption. To facilitate independent auditing and protect vote privacy at the same time, mix-nets need to be applied in this case. We refer to [5] for a recent overview on these techniques.

Secure computations on top of secret sharing can actually be implemented in a homomorphic way. In fact, the tally routine of Dang-awan *et al.* makes use of homomorphic properties of Sharemind's additive secret sharing. But the missing piece of the puzzle is a software-independent cryptographic trail that can be verified for integrity by independent auditors. Developing such a component is a necessary prerequisite for a voting system to be considered secure in late 2010-s.

6 Conclusions

Implementing a secure computation application is tricky, even if you have access to a well-established platform with provable security guarantees like Sharemind.

First of all, it is crucial to understand all the security aspects of the application domain. Electronic voting is an especially complicated area, since there are a number of different and partly even contradicting requirements. Failure to take some of them (like coercion-resistance or tally integrity) into account will result in an insecure system.

It is also important to fully specify and understand the threat model. In case on remote electronic voting, the biggest vulnerabilities are inflicted by the weaknesses of the client platform. The protocol of Dang-awan *et al.* relies on a web browser as the sole voter device. If the attacker manages to gain the control of it, he can leave the voter with the impression that her vote was cast fine, whereas in reality it has been modified or blocked altogether. To counter this threat, a verification procedure utilising an independent channel is unavoidable.

Third, using a secure computation platform does not automatically imply security of the application. On one hand, there are several definitions of security. And on the other hand, there are simple programming flaws like declassifying values too early that render the whole framework useless. As declassification is a necessary part of secure computation protocols, it can not just be prohibited. It is inherently the responsibility of the application developer to use it only when absolutely necessary.

Secure computation platforms are in fast development and their functionality is expanding all the time. Nevertheless, it may sometimes happen that a desired API call is missing. In that case it is a bad idea to search through the API documentation for something remotely similar. CRC32 was never designed to be a cryptographic hash function (even though check-summing is sometimes also called hashing). Its collisions are easy to find and it is easy to invert. Confusing the design with an extra layer of cryptographic hashing does not improve the situation.

And last but not least – even the strongest cryptographic guarantees do not protect against poor overall protocol design. Hashing values from a relatively small domain without extra entropy leads to easy pre-image-finding even if the hash function would be implemented securely. In case of electronic voting applications, this leads to large-scale privacy violation and potential coercion attacks.

Learning is a process that involves making mistakes, and before a person is able to build something strong, he/she has to acquire knowledge of potentially weak places. We hope that this paper has served as a good study use case for the future architects of both secure computation and voting applications.

Acknowledgments The research leading to these results has received funding from the Estonian Research Council under Institutional Research Grant IUT27-1 and the European Regional Development Fund through the Estonian Centre of Excellence in ICT Research (EXCITE) and the grant number EU48684.

References

1. Report of the National Workshop on Internet Voting: Issues and Research Agenda (March 2001), internet Policy Institute, https://www.verifiedvoting.org/downloads/NSFInternetVotingReport.pdf
2. Archer, D.W., Bogdanov, D., Pinkas, B., Pullonen, P.: Maturity and Performance of Programmable Secure Computation. IEEE Security & Privacy 14(5), 48–56 (2016), https://doi.org/10.1109/MSP.2016.97
3. Benaloh, J.C., Fischer, M.J.: A Robust and Verifiable Cryptographically Secure Election Scheme (Extended Abstract). In: 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985. pp. 372–382. IEEE Computer Society (1985), https://doi.org/10.1109/SFCS.1985.2
4. Benaloh, J.C., Yung, M.: Distributing the Power of a Government to Enhance the Privacy of Voters (Extended Abstract). In: Halpern, J.Y. (ed.) Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing, Calgary, Alberta, Canada, August 11-13, 1986. pp. 52–62. ACM (1986), https://doi.org/10.1145/10590.10595
5. del Blanco, D.Y.M., Alonso, L.P., Alonso, J.A.H.: Review of Cryptographic Schemes applied to Remote Electronic Voting systems: remaining challenges and the upcoming post-quantum paradigm. Open Mathematics 16(1), 95–112 (2018)
6. Bogdanov, D., Laud, P., Randmets, J.: Domain-polymorphic Language for Privacy-preserving Applications. In: Proceedings of the First ACM Workshop on Language Support for Privacy-enhancing Technologies. pp. 23–26. PETShop '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2517872.2517875
7. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A Framework for Fast Privacy-Preserving Computations. In: Jajodia, S., López, J. (eds.) Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5283, pp. 192–206. Springer (2008), https://doi.org/10.1007/978-3-540-88313-5_13
8. Cramer, R., Damgård, I.B., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press (July 2015)
9. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. In: Safavi-Naini, R., Canetti, R.

(eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer (2012), https://doi.org/10.1007/978-3-642-32009-5_38
10. Dang-awan, R., Piscos, J.A., Chua, R.B.: Using Sharemind as a Tool to Develop an Internet Voting System with Secure Multiparty Computation. In: 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA). pp. 1–7. IEEE (July 2018)
11. Eerikson, H., Orlandi, C., Pullonen, P., Puura, J., Simkin, M.: Use your Brain! Arithmetic 3PC For Any Modulus with Active Security. Cryptology ePrint Archive, Report 2019/164 (2019), https://eprint.iacr.org/2019/164
12. Frikken, K.B.: Secure Multiparty Computation. In: Atallah, M.J., Blanton, M. (eds.) Algorithms and Theory of Computation Handbook, Volume 2: Special Topics and Techniques, pp. 14–1. . . 14–16. CRC Press (2009)
13. Gang, C.: An Electronic Voting Scheme Based on Secure Multi-party Computation. In: 2008 International Symposium on Computer Science and Computational Technology. vol. 1, pp. 292–294 (Dec 2008)
14. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009), https://doi.org/10.1145/1536414.1536440
15. Gjøsteen, K.: The Norwegian Internet Voting Protocol. In: Kiayias, A., Lipmaa, H. (eds.) E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7187, pp. 1–18. Springer (2011), https://doi.org/10.1007/978-3-642-32747-6_1
16. Gjøsteen, K., Strand, M.: A Roadmap to Fully Homomorphic Elections: Stronger Security, Better Verifiability. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y.A., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10323, pp. 404–418. Springer (2017), https://doi.org/10.1007/978-3-319-70278-0_25
17. Heiberg, S., Willemson, J.: Verifiable internet voting in Estonia. In: Krimmer, R., Volkamer, M. (eds.) 6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014. pp. 1–8. IEEE (2014), https://doi.org/10.1109/EVOTE.2014.7001135
18. Jonker, H., Mauw, S., Pang, J.: Privacy and verifiability in voting systems: Methods, developments and trends. Computer Science Review 10, 1–30 (2013), https://doi.org/10.1016/j.cosrev.2013.08.002
19. Krimmer, R.: The Evolution of E-voting: Why Voting Technology is Used and How it Affects Democracy. Ph.D. thesis, Tallinn University of Technology (2012), doctoral Theses Series I: Social Sciences
20. Laud, P., Pankova, A., Jagomägis, R.: Preprocessing Based Verification of Multiparty Protocols with Honest Majority. PoPETs 2017(4), 23–76 (2017), https://doi.org/10.1515/popets-2017-0038
21. Liu, C., Wang, X.S., Nayak, K., Huang, Y., Shi, E.: ObliVM: A Programming Framework for Secure Computation. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 359–376. IEEE Computer Society (2015), https://doi.org/10.1109/SP.2015.29

22. Madise, Ü., Martens, T.: E-voting in Estonia 2005. The first Practice of Country-wide binding Internet Voting in the World. In: Krimmer, R. (ed.) Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, August, 2nd - 4th, 2006 in Castle Hofen, Bregenz, Austria. LNI, vol. 86, pp. 15–26. GI (2006), http://subs.emis.de/LNI/Proceedings/Proceedings86/article4547.html
23. Martins, P., Sousa, L., Mariano, A.: A Survey on Fully Homomorphic Encryption: An Engineering Perspective. ACM Comput. Surv. 50(6), 83:1–83:33 (Dec 2017), http://doi.acm.org/10.1145/3124441
24. Nair, D.G., Binu, V.P., Kumar, G.S.: An Improved E-voting scheme using Secret Sharing based Secure Multi-party Computation (2015)
25. Puiggali, J., Cucurull, J., Guasch, S., Krimmer, R.: Verifiability Experiences in Government Online Voting Systems. In: Krimmer, R., Volkamer, M., Binder, N.B., Kersting, N., Pereira, O., Schürmann, C. (eds.) Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10615, pp. 248–263. Springer (2017), https://doi.org/10.1007/978-3-319-68687-5_15
26. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security. pp. 707–721. ASIACCS '18, ACM, New York, NY, USA (2018), http://doi.acm.org/10.1145/3196494.3196522
27. Rivest, R.L.: On the notion of 'software independence' in voting systems. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 366(1881), 3759–3767 (2008)
28. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)
29. Springall, D., Finkenauer, T., Durumeric, Z., Kitcat, J., Hursti, H., MacAlpine, M., Halderman, J.A.: Security analysis of the Estonian internet voting system. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 703–715. ACM (2014)
30. Yao, A.C.: Protocols for Secure Computations (Extended Abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982. pp. 160–164. IEEE Computer Society (1982), https://doi.org/10.1109/SFCS.1982.38