TrustedVote
Trusted Computing for Internet Voting

Diogo Monteiro and Paulo Ferreira

INESC-ID / Instituto Superior Técnico
University of Lisbon, Portugal
diogo.p.monteiro@tecnico.ulisboa.pt, paulo.ferreira@inesc-id.pt

Abstract. The wide availability of mobile devices, such as smartphones, enabled mobility in our lifestyles. However, traditional voting systems require physical presence of the voter at a specific place and time, which is incompatible with the concept of mobility. The goal of this paper is to propose an Internet voting system, called TrustedVote, that allows voters to cast their vote anywhere. Moreover, Internet voting significantly raises the turnout rate, reduces administrative costs and tallying time. In order to tackle malware and other insecurities in the client mobile platform, the solution is based on smartphones with a Trusted Execution Environment (TEE). TrustedVote leverages the isolation properties of TEEs available in Android and iOS smartphones to perform the cryptographic steps of an Internet voting system, such as vote encryption and voter authentication.

Keywords: Electronic Voting; Internet Voting; Security; Cryptography; Trusted Computing;

1 Introduction

The goal of democracy is to allow all the citizens to cast their vote that corresponds to their will. Traditional, paper-based, voting systems require the voter to cast his vote during a certain period (the election period) and at a specific place, which is inconsistent with the concept of mobility widely seen today (e.g., with smartphones, tablets, smartwatches, etc.). This lack of mobility compromises the goal of democracy, as the citizens may not be available to vote during the election period. Consider the scenario where a voter is registered to vote in Lisbon. A business meeting, which the voter must attend, is scheduled, at the last minute, to be held in Paris during the election period. With the traditional voting method, the voter would not be able to cast his vote.

1.1 Goal

The goal of this paper is to propose an internet voting protocol named *TrustedVote* that allows voters to vote anywhere. From this goal, the following requirements are considered:

1. **Accuracy** - it is not possible for an invalid vote to be counted in the final tally.
2. **Integrity** - a malicious attacker cannot, arbitrarily or in a deterministic way, modify a vote without detection at the client, communication channels or server.
3. **Democracy** - only authorized voters may cast a vote and an eligible voter may only cast one vote.
4. **Privacy** - no entity besides the voter learns how he cast his vote. He is not able to prove to a third party how he voted.
5. **Verifiability** - any independent entity is able to verify that all votes were counted correctly. Additionally, a voter can verify if his vote was recorded correctly.
6. **Robustness** - the protocol should consider the following robustness requirements:
   (a) **Availability** - the system should be available during the election period.
   (b) **Collusion Resistance** - the protocol should be resistant to collusion of corrupt voting authorities.
   (c) **Malware Resistance** - the insecure platform problem is defined as the insecurity that is found at the vote casting platforms because they are uncontrolled environments vulnerable to attacks. This problem should also be mitigated by tolerating malware in the client voting machine.
7. **Mobility** - the e-voting system should not impose mobility restrictions to the voter. The voter has the freedom to vote anywhere.
8. **Usability** - to successfully cast a vote, it is not required for the user to acquire special or dedicated devices that are not widely available. Moreover, from the point of view of the voter, the voting process imposed by the protocol should be intuitive and easily recognizable.

We consider the requirements that a non-electronic voting scheme should have, such as accuracy, integrity, democracy, privacy, verifiability, collusion resistance, availability and usability. In addition to non-electronic voting requirements, we consider: 1) mobility, that non-electronic voting cannot achieve because a voter must move to a voting booth to cast his vote and, 2) malware resistance.

The design and implementation of an 100% secure internet voting system is difficult. Protocols that achieve the core security properties (accuracy, integrity, democracy, privacy and verifiability) and robustness [6, 8, 2, 11], have serious usability problems. On the other side, usable e-voting schemes that guarantee the core security properties protecting the vote at server-side [12, 13, 15, 13, 8, 2, 11], fail to provide malware resistance. They do not consider malware in the voting client machine. A malware in the operating system is able to perform arbitrary operations on the vote before being encrypted, compromising the privacy and integrity of the vote without detection.

Thus, there is the need to tolerate malware in the voter's computer while maintaining usability. A TEE is a special area of the main processor that executes in isolation from the rest of the hardware. It is possible to leverage secure

storage and the isolation feature to perform the cryptographic steps of an e-voting algorithm, guaranteeing privacy of the vote even to the operating system. The solution is to split the client of the voting protocol that executes in the voter's machines in two components: 1) a trusted component that executes sensitive operations in isolation from the operating system and, 2) an untrusted component that implements the steps of the voting protocol.

The remaining of this paper is organized as follows: Section 2 overviews the state of the art regarding electronic voting and trusted execution environments. In Section 3, we present a the architecture of TrustedVote. The implementation details is discussed in Section 4 and evaluation methodology is discussed in Section 5. Finally, we conclude in Section 6.

2 Related Work

This section outlines the related work and is divided in two main parts. The first part contains an overview of the state of the art regarding electronic voting protocols. We also informally discuss why each protocol does not meet all the requirements devised in Section 1.1. In the second part, we overview TEE technologies and how they can be used in the context of e-voting. Finally, we conclude with a summary of all related solutions.

Electronic voting protocols use secure channels to perform network communications (e.g. SSL/TLS). They assume the existence of Certification Authorities that certificate asymmetric public keys of the protocol entities.

2.1 REVS

REVS [9] is an e-voting protocol proposed by Joaquim *et. al* in 2003 that uses blind signatures [3] to separate the authentication of users from the authentication of ballots, removing the link between the voter and his ballot.

In the first phase of the REVS protocol, the voter casts a vote and produces a ballot. The voter computes a blinded version of the ballot B' and sends it to $t > \frac{n}{2}$ trusted administrators, where n is the number of administrators. Each administrator signs the blinded ballot B' with its private key and returns the computed signature to the voter. The ballot is only considered valid (and considered in the final tally) if the voter collects $t > \frac{n}{2}$ signatures from the administrators. At the final stage, the vote is sent to a Counter server (through an Anonymizer server that hides IP address and introduces random delays in the network) and counted after the election period.

REVS focuses on server-side fault tolerance, as the design allows for replication of all election servers. However, it is assumed that the machine used by the voter must be trusted and follow the protocol. If this assumption is not hold, then the integrity and privacy of the vote are compromised. This compromises the robustness of the protocol.

2.2 Java Card E-voting

The Java Card technology facilitates the development of Java applications to smart cards. A smart card is a device with very limited amount of memory and processing power. However, the smart card is tamper resistant. It provides an environment where the processor instructions and contents of the memory cannot be eavesdropped or tampered. The online voting system proposed by Mohammadpourfard *et. al* [10] takes advantage of the Java Card 3 technology to enhance its security.

The voter smart card starts by generating a secret alias aID and blinding it (aID') with a random blinding factor r. Along with the aID', the smart card sends to the election servers the voter identification id. The election servers check if the voter is eligible to vote and signs aID' with its private key. The smart card removes blinding factor r to retrieve the signature of aID. aID and its signature may now be used as tokens to authenticate the voter to the election servers.

The voter's smart card is now able to produce a ballot B and its blinded version B'. Next, it sends B' (with aID and its signature) to be signed. The voter removes the blinding factor from the returned signature, and sends B and signature to the election servers in order to be counted. The system assumes that all citizens have access to a Java smart card reader, which affects usability. However, the Java smart card reader can be attached to a mobile phone, achieving mobility.

2.3 EVIV

EVIV [7] is an end-to-end verifiable internet voting system that uses homomorphic encryption [8] takes into consideration that the client platform may be insecure and controlled by a malicious attacker. EVIV assumes that the voter has a component called Voter Security Token (VST), that is responsible for the encryption of the vote and authentication of the voter using digital signatures. In EVIV, the VST is implemented using a tamper-proof Java smart card containing the private key of the voter.

Prior to the election, the voter inserts the VST into his computer, in order to generate a code card to the upcoming election. The code card associates a random vote code (string) for each candidate. During the voting period, the voter uses the code card to insert the vote code associated with the candidate. The code card mechanism creates a secure channel between the voter and the VST.

EVIV ensures the security properties defined in Section 1.1, but fails to provide usability, as it is assumed that the voter has access to a smart card reader.

2.4 Trusted Execution Environments

There is a clear trade off between malware resistance and usability in e-voting protocols. Protocols that tolerate malware in the client platforms have low usability [6, 7, 10] and protocols with high usability do not tolerate malware [15,

14, 9, 2]. In the context of e-voting protocols, TEEs are able to provide malware resistance. They can protect against privacy and integrity attacks executed by malicious software in the operating system, without compromising usability.

A TEE is a dedicated area of the main processor that executes in isolation from the remaining of the hardware. It offers a secure environment for applications to execute. This section delineates the main concepts of ARM TrustZone, aTEE technology that is used in TrustedVote.

ARM TrustZone is a security extension architecture available in ARM processors that allows execution of code and services isolated from the operating system [1]. The hardware layer of ARM TrustZone provides two worlds of execution: 1) the normal world, where the rich operating system kernel runs, and 2) the secure world, where the secure operating system runs. The secure operating system supports multiple Security Servers where code that handles sensitive computations is executed without relying on a complex code base. At a given time, the processor is only executing instructions in one world. Therefore, the hardware is equipped with the Secure Monitor Call (SMC) system call to switch between the two worlds of execution. The rich operating system kernel ships with the TrustZone Driver that is responsible to handle world switches. When a TrustZone-enabled application (Security Client) running in the normal world wants to perform a sensitive operation in the secure world, it calls the TrustZone Driver that issues the SMC system call. The hardware jumps to the secure world Monitor, that performs a secure world switch and copies required data across worlds, enabling the sharing of memory between the two worlds.

Each execution mode has its independent memory space. Code executing in normal world can only access normal world memory space, while code running in the secure world can only access secure world memory space.

As ARM processors are widely available in Android and iOS phones today, the ARM TrustZone design allows the development of secure applications for mobile environments. This way, it is possible to leverage this technology to develop secure components of an e-voting application. In the context of an e-voting application, malware resistance can be achieved if the sensitive operations (such as cryptographic steps and the casting of a ballot) are delegated to the secure world, because malware in the rich operating system cannot read or write in secure world memory, nor intercept network communications issued by the secure world.

3 Architecture

This section describes the design of TrustedVote. TrustedVote protocol is an end-to-end verifiable e-voting system that tolerates malware in the client computers, while keeping high levels of usability, i.e without the need to use a device that is not widely available to the public.

To achieve that, TrustedVote combines the EVIV network and cryptography protocol with a new architecture of the client application that executes in

the voter's mobile device. The EVIV system is missing usability, because in its current implementation, it requires from the voter dedicated hardware (a smart card reader) to successfully cast a vote. TrustedVote removes the necessity of a separate tamper-proof smart card by using the isolation features provided by ARM TrustZone TEE (Section 2.4).

We assume that the adversary is able to control the operating system and execute arbitrary operations by installing malware. However, we assume that the ARM TrustZone hardware behaves correctly and we do not consider side channel or other physical attacks to the hardware.

The remaining of this Section is organized as follows. The MarkPledge 3 cryptography scheme used by TrustedVote is described in Section 3.1. Section 3.2 overviews the set of entities of TrustedVote. The TrustedVote network protocol is presented in Section 3.3. Finally, the client architecture is discussed in Section 3.4.

3.1 MarkPledge 3

The MarkPledge 3 technique was proposed by Joaquim and Ribeiro [8] and aims at providing tools that allow a voter to compute vote encryptions and verify if a vote encryption is correct. The MarkPledge 3 interface is divided in the following set of functions:

1. **Vote encryption** $\mathcal{VE}_{pk}(b, \theta, r) =< BitEnc(b), voteValidity >$
   The vote encryption function produces an encryption of a $NOvote$ ($b = -1$) or $YESvote$ ($b = 1$) that can be associated with a candidate. Moreover, this function also computes vote validity attributes, that allow to verify if the encryption is a valid encryption of a $YESvote$ or $NOvote$.
2. **Vote validity** $\mathcal{VV}_{pk}(BitEnc(b), voteValidity) = True$ or $False$
   The vote validity function returns $True$ if the vote encryption $BitEnc(b)$ corresponds to an encryption of a $NOvote$ ($b = -1$) or $YESvote$ ($b = 1$). Returns $False$ otherwise.
3. **Receipt creation** $\mathcal{RC}_{pk}(BitEnc(b), r, c) = (\vartheta, \omega)$
   The receipt creation function outputs a verification code ϑ that allows the voter to verify if the voter's computer encrypted the vote according to his intentions.
4. **Receipt validity** $\mathcal{RV}_{pk}(BitEnc(b), c, < \vartheta, \omega >) = True$ or $False$
5. **Tally** $\mathcal{TF}(allVotes) = (voteCount_1, ..., voteCount_n)$
   The tally function takes as input all submited votes and returns the final tally, i.e. the number of $YESvotes$ for each candidate.

3.2 Entities

TrustedVote takes into consideration the following entities and services:

– **Electoral Commission (EC)** - responsible for the entire election process, and authentication of all public data.

- **Enrollment Service (ES)** - responsible for the enrollment of every voter.
- **Election Registrar (ER)** - service that voters register to vote on a specific election.
- **Ballot Box (BB)** - service that voters use to send their vote.
- **Verification Service (VS)** - each organization runs an instance of the verification service that verifies if the votes and receipts are correct and valid.
- **Trustees ($\mathcal{T}$)** - set of organizations and parties that keep secret an ElGamal [5] election asymmetric key pair (K_{pub}, K_{priv}). Each trustee has a share of K_{priv} such that the decryption of a message requires the collaboration of $t < n$ trustees, where n is the total number of trustees [4].
- **Voter ($\mathcal{V}$)** - person with access to an ARM TrustZone-enabled mobile device that wishes to vote in the election.

It is assumed that each entity has an asymmetric key pair publicly known by the other entities. For instance, the key pair of the Election Registrar service is represented as (K_{ER}, K_{pER}), where K_{ER} is the public key, and K_{pER}. The encryption of *message* with K_{ER} is denoted by $(message)_{K_{ER}}$ while the *message* and concatenation of the signature of *message* with K_{pER} is denoted by $(message)_{K_{pER}}$

3.3 Protocol

TrustedVote is built on top of an ElGamal election key (K, K_p) [5]. However, it requires that the election private key K_p is split between a set of n trustees, such that a decryption operation requires the cooperation of $t < n$ trustees. The votes are encrypted using the Markpledge 3 technique, that ensures the voter (or any independent organization) can verify if the vote is counted and is encrypted correctly.

TrustedVote protocol is divided in three phases: 1) the election setup, 2) the voting phase, and 3) the tally and verification phase.

Election Setup (before the voting period)

1. $\mathcal{T}_t \rightarrow Bulletin\ Board \mid (pk)_{K_{p\mathcal{T}_t}}$
   Each trustee t sends his share of the election public key pk to the Bulletin Board. The election public key K is now available.
2. $EC \rightarrow Bulletin\ Board \mid (candidateList, electionParameters)_{K_{pEC}}$
   Electoral commission sends the candidate list and election parameters (election public key K) to the public Bulletin Board.
3. $ER \rightarrow \mathcal{V} \mid (candidateList, electionParameters)_{K_{pEC}}$
   The voter client retrieves the candidate list and the election public key from the Election Registrar.
4. $\mathcal{V} \rightarrow ER \mid (ballot)_{K_{p\mathcal{V}}}$
   Voter registers to vote by creating a ballot. A TrustedVote ballot is a structure composed by an array of k vote encryptions, where k is the number of

candidates and each vote encryption is computed using the vote encryption (Equation 1) from the MarkPledge 3 specification (Section 3.1). Each vote is encrypted with the election public key K.

$$\mathcal{VE}_{pk}(b, \theta, r) =< BitEnc(b), voteValidity > \quad (1)$$

(missing sum validity, i.e the proof that there only exists one $YESvote$ in a given ballot)
Upon receiving the ballot, the Election Registrar validates the vote and receipt using $\mathcal{VV}$ and $\mathcal{RC}$ from the MarkPledge 3 specification, respectively. It signs the ballot and sends it to the Bulletin Board.

5. $\mathcal{V}$ generates a random code card for the upcoming election. A code card is an association of a random vote code (string) to a candidate. The code card also contains a confirmation code, that allows the voter to confirm in the receipt if the vote matches his intentions.

Voting phase (during the voting period)

1. $\mathcal{T}_t \rightarrow Bulletin\ Board \mid (randomNumber)_{K_{p\mathcal{T}_t}}$
Each trustee t sends a random number to the public Bulletin Board in order to generate an election challenge. The final election challenge is computed by applying bitwise XOR to all random numbers submitted by the trustees. The Electoral Commission signs the election challenge.
2. $Bulletin\ Board \rightarrow \mathcal{V} \mid (electionChallenge)_{K_{pEC}}$
3. Voter inserts the desired vote code in his mobile device and the mobile device presents to him the vote receipt and receipt validity.
4. The voter checks if the receipt is valid by checking if the confirmation code in his code card matches the row of the voted candidate and confirms the submission.
5. $\mathcal{V} \rightarrow BB \mid (vote, receipt, receiptValidity)_{K_{p\mathcal{V}}}$
If the previous step succeeds the vote, the voter sends the final vote, receipt and receipt validity to the Ballot Box.

Tally and verification (after the voting period)

1. $EC \rightarrow Bulletin\ Board \mid (homomorphicVoteAggregation)_{K_{pEC}}$
After the election period is over, the bulletin Board computes the homomorphic vote aggregation and asks the Electoral Commission to validate the computation by signing it.
2. $\mathcal{T}_i \rightarrow Bulletin\ Board \mid (partialDecryption, decryptionProof)_{K_{p\mathcal{T}_i}}$
Each trustee i collects the homomorphic vote encryption (verifying its signature) from the Bulletin Board and sends its partial decryption and decryption proof to the Bulletin Board. With t valid partial plain texts, the Bulletin Board is able to decrypt the final tally. The Electoral Commission signs the result.
3. $Bulletin\ Board \rightarrow VS \mid (candidateList, ballotList, voteList, receipts, receiptValidities)_{K_{pEC}}$
Any independent organization can retrieve the election data and verify every vote encryption validity for every vote and every receipt validity for every receipt.

3.4 Client architecture

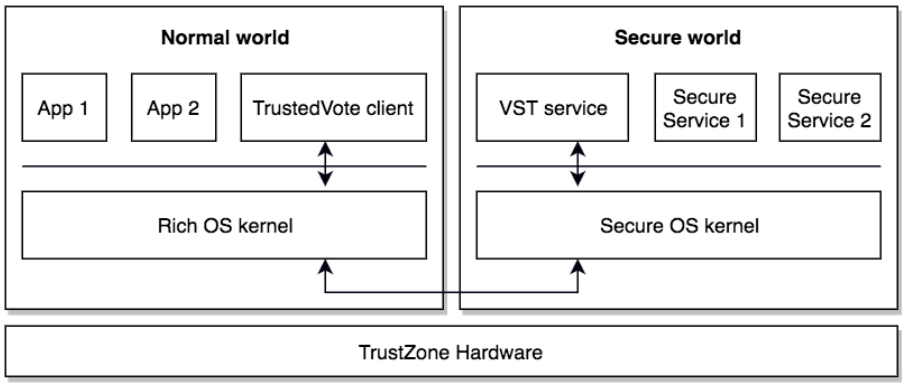

Fig. 1. TrustedVote client application architecture.

Figure 1 provides an overview of TrustedVote's client application architecture, mentioned as the $\mathcal{V}$ component in the protocol description (Section 3.3).

The application is designed on top of a processor with ARM TrustZone enabled and is split in two components. The trusted component *VST service* and the untrusted component *TrustedVote client*. These components map to the secure and normal worlds of ARM TrustZone, respectively.

VST service The *VST service* is responsible for executing the code that authenticates the voter, encrypts the vote using the MarkPledge 3 technique (Section 3.1), stores the private key of the voter and generates the code card. Next, we describe the secure calls provided by the *VST service*. The secure calls are presented as `NAME`: $(input) \rightarrow (output)$, where `NAME` is the name of the secure call, and *input* and *output* are the input and output parameters, respectively.

1. `RECEIVE_CANDIDATES_CALL`: $(candidateList) \rightarrow ()$
   Secure call that takes the candidate list returned by the Election Registrar as input and stores it on secure memory.
2. `RECEIVE_ELECTION_PARAMETERS_CALL`: $(electionParameters) \rightarrow ()$
   Secure call that receives the election public key K as parameter and stores it on secure memory.
3. `SET_ELECTION_CHALLENGE_CALL`: $(electionChallenge) \rightarrow ()$
   Secure call that takes the election challenge as input and stores it as secure memory.
4. `PREPARE_BALLOT_CALL`: $() \rightarrow (ballot, (ballot)_{K_{p\mathcal{V}}})$
   Secure call that creates an empty TrustedVote ballot during the election setup phase. The ballot is signed with the voter's private key $K_{p\mathcal{V}}$. The secure call returns the ballot and its signature.

5. `GENERATE_CODE_CARD_CALL`: $() \rightarrow (codeCard)$
   Secure call that generates a code card for the election. The code card is stored in secure memory address space and shown to the voter prior to the election.
6. `SELECT_CANDIDATE_CALL`: $(voteCode) \rightarrow (receipt, receiptValidity)$
   Secure call only executes during the election period, and accepts as input the vote code of the chosen candidate. The call then translates the vote code into the candidate and rotates the vote encryptions (in the ballot generated at `PREPARE_BALLOT_CALL`) until the $YESvote$ is aligned with the chosen candidate.
   Moreover, the vote receipt and its validity proof are computed using the RC and RV MarkPledge 3 functions. At the end, the receipt and its validity are returned.
7. `OK_SUBMIT_CALL`: $() \rightarrow (vote)$
   `OK_SUBMIT_CALL` returns the rotated ballot produced by `SELECT_CANDIDATE_CALL`.

The *VST service* must be executed in the secure world because otherwise: 1) the voter's private key could be stolen by the attacker, allowing him to impersonate the voter, and 2) the vote encryption operation could be intercepted by a malware, changing the $YESvote$ position without detection.

TrustedVote client The TrustedVote client mediates the network communications between the *VST service* and election servers, and offers the user interface to the voter.

4 Implementation

The prototype implementation of election servers, i.e. Electoral Commission, Election Registrar, Ballot Box, Bulletin Board and Trustee, was made in Python 2.7.6 (missing ref) and Django Framework 1.10.6 (missing ref). Python is an open sourced high level language that allows fast development of applications and easy maintenance. Django is a web framework that ships with a powerful Object Relational Mapper, allowing a fast development of persistent databases with low effort. The database system used was MySQL version 5.6.21 (missing ref). These set of tools were chosen in order to reduce the effort of development and maintenance of TrustedVote.

The client prototype was implemented in the *iMX53 Quick Start Board* from Freescale. Currently, the bootstrap code in the majority of the boards equipped with ARM processors switch to normal world. As this code is written in ROM, it is impossible to load an operating system in secure world before the world switch happens. The *iMX53 Quick Start Board* is one of the few boards that does not contain this world switch written in its bootstrap code.

In order to build the secure OS kernel, we chose Genode framework (missing ref). The Genode framework is a collection of building blocks, such as filesystems,

protocol stacks and libraries, that can be used to build a kernel. Genode ships with a *base-hw* configuration that runs a microkernel (with only 20KLOC) in secure world and Linux 2.6 in the normal world. This way, the *VST service* was implemented on top of the microkernel built by *base-hw* configuration of Genode. In the current implementation, the *VST service* depends on *openssl* library to manipulate integers with arbitrary number of bits, and *libc++* to reuse its implementation of vectors and maps. We modified the Linux kernel to include a special system call that executes the SMC instruction when a secure call to the *VST service* is issued.

The microkernel of *base-hw* does not solve the problem of sharing memory across worlds. It is possible to share data across worlds in the CPU registers. But each CPU register can only store 4 bytes, which is not enough for transfering a TrustedVote ballot, for example. The solution implemented is to allocate a DMA shared memory buffer in normal world and write the address and size of the buffer in CPU registers.

5 Evaluation

TODO. How can I evaluate the protocol?

6 Conclusion

The goal of this paper is to propose a fully mobile internet voting system named TrustedVote. With this system, the voters can vote anywhere with an internet connection and be certain that neither the vote nor the election is compromised. In order to achieve this goal, ARM TrustZone is used to provide an implementation of EVIV's VST functionality. This is possible because ARM TrustZone allows the isolated execution of applications in mobile devices where TrustZone is enabled. Unlike previous systems, TrustedVote keeps a high level of usability as the voter does not have to acquire special hardware devices to successfully cast his vote.

(missing evaluation summary)

References

1. ARM. ARM Security Technology - Building a Secure System using TrustZone Technology. ARM Technical White Paper. 2009. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, accessed: 2016-11-28
2. Alrodhan, W., Alturbaq, A., Aldahlawi, S.: A mobile biometric-based e-voting scheme. 2014 World Symposium on Computer Applications and Research, WSCAR 2014 (2014)
3. Chaum, D.: Blind signatures for untraceable payments. In: Advances in cryptology. pp. 199–203. Springer (1983)

4. Cramer, R., Gennaro, R., Schoenmakers, B.: A Secure and Optimally Efficient Multi-authority Election Scheme. In: Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques. pp. 103–118. EUROCRYPT'97, Springer-Verlag (1997)
5. Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (Jul 1985)
6. Grewal, G.S., Ryan, M.D., Chen, L., Clarkson, M.R.: Du-Vote: Remote Electronic Voting with Untrusted Computers. Proceedings of the Computer Security Foundations Workshop 2015-September, 155–169 (2015)
7. Joaquim, R., Ferreira, P., Ribeiro, C.: Eviv: An end-to-end verifiable internet voting system. computers & security 32, 170–191 (2013)
8. Joaquim, R., Ribeiro, C.: An Efficient and Highly Sound Voter Verification Technique and Its Implementation, pp. 104–121. Springer Berlin Heidelberg (2012)
9. Joaquim, R., Zúquete, A., Ferreira, P.: REVS – A Robust Electronic Voting System. IADIS International Journal of WWW/Internet 1(i), 47–63 (2003)
10. Mohammadpourfard, M., Doostari, M.A., Ghaznavi Ghoushchi, M.B., Shakiba, N.: A new secure Internet voting protocol using Java Card 3 technology and Java information flow concept. Security and Communication Networks 8(2), 261–283 (2015)
11. Petcu, D., Stoichescu, D.A.: A hybrid mobile biometric-based e-voting system. In: 2015 9th International Symposium on Advanced Topics in Electrical Engineering (ATEE). pp. 37–42 (May 2015)
12. Ryan, P.Y.A., Bismark, D., Heather, J., Schneider, S., Xia, Z.: Prêt À Voter: a Voter-Verifiable Voting System. IEEE Transactions on Information Forensics and Security 4(4), 662–673 (Dec 2009)
13. Ryan, P.Y.: A variant of the Chaum voter-verifiable scheme. In: Proceedings of the 2005 Workshop on Issues in the Theory of Security. pp. 81–88. ACM (2005)
14. Ryan, P.Y.: Prêt À Voter with Paillier encryption. Mathematical and Computer Modelling 48(9), 1646–1662 (2008)
15. Ryan, P.Y., Schneider, S.A.: Prêt À Voter with re-encryption mixes. In: European Symposium on Research in Computer Security. pp. 313–326. Springer (2006)