sElect: Secure Election as a Service

Preprint · May 2019

DOI: 10.13140/RG.2.2.17094.93760

3 authors:

Mohamed Nassar
American University of Beirut
56 PUBLICATIONS **292** CITATIONS
SEE PROFILE

Bassel Rawda
American University of Beirut
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Mohamad Mardini
American University of Beirut
1 PUBLICATION **0** CITATIONS
SEE PROFILE

sElect: Secure Election as a Service

Mohamed Nassar, Bassel Rawda, Mohamed Mardini
Computer Science Department
Faculty of Arts and Sciences
American University of Beirut (AUB)
[mn115|mjm26|brr01] at aub.edu.lb

May 2, 2019

Abstract

Online voting is a challenging socio-technical problem that is still an open research question. Current approaches are based on involved cryptographic solutions that hardly can be explained to the average voter. Still, attackers may be able to circumvent the system without breaking the cryptographic protocols. In this paper, we demonstrate a toy voting protocol based on additive homomorphic encryption and two non-colluding parties. The protocol design aims at preserving the essential security properties such as election integrity and voter anonymity while being much simpler to explain. We propose a RESTful implementation of a web front-end of the proposed system. We detail the web services architecture, the database schemas, the employed technologies and the functional building blocks. We highlight the fact that cryptography is only one facet of security. By conducting several web attack scenarios to test the robustness of the web interface, we show that the system may still be compromised without breaking the cryptography.

Keywords: Voting, Cybersecurity, Homomorphic Encryption

Topics Web and Cyber Security

1 Introduction

Voting is a method for a group of people to collectively elect a person, take a decision or express an opinion. Historically voting had many forms such as live voices, paper ballots, mechanical

machines, touch screens (Direct Recording Electronic or DRE), optical scanners and online voting using Internet.

Online voting systems are gaining acceptance with the widespread use of secure web services and cloud computing such as electronic currency and online banking. However, many researchers agree that online voting is hard. It has challenging privacy, security and accountability issues. In this paper we opt for the simplicity, intelligibility and usability of the voting protocol. Nevertheless we preserve the anonymity of the voters and the integrity of the results. We propose a cryptographic solution that is simple to explain to the voters. Our protocol is based on partially homomorphic encryption and two non-colluding parties. We provide a RESTful implementation of the voting framework using micro and distributed web services. More importantly, we recognize that the correctness of the cryptography does not ensure security, and that security is a process rather than just a product. Therefore, we pen-test our web services for the most known web and remote vulnerabilities. We report on the performance and the security of our system.

The rest of the paper is organized as follows: Section II summarizes related work. Section III describes the voting protocol. Section IV presents the RESTful implementation and the technologies used. Section V reports on the penetration and performance testing. Finally, section VI concludes the paper.

2 Related Work

Many secure and verifiable voting schemes are proposed with solid cryptographic foundations. They ensure the verifiability of the results (also known as the integrity property) and in the same time deny any match between the votes and the corresponding voters (also known as the ballot secrecy property). Additional security criteria such as software independence and end-to-end verifiability have emerged. Most protocols use mix nets (e.g. [1], [6]) and homomorphic encryption. This research resulted in many online systems offering verifiable elections such as Helios (`https://vote.heliosvoting.org/`). With the arising popularity of Blockchain applications, voting has also been formalized in terms of a smart contract [7].

However, online voting has many practical constraints. Helios 2.0 is shown to be vulnerable to a man-in-the-middle attack by installing a browser rootkit that detects the ballot web page and modifies votes [5]. For a thorough evaluation of Helios based on the requirements issued by the Council of Europe in 2017 we refer the reader to [2]. Blockchain has its security weaknesses. Several attacks against smart contracts are surveyed in [3]. In [11] the experience of attacking the Washington, D.C. Internet voting system highlights its many weaknesses. Within 48 hours of the system going live, it was almost completely compromised. Every vote was revealed and changed. Election officials did not detect the intrusion for nearly two business days. This is why many countries dropped electronic voting for absentee overseas voters over cybersecurity

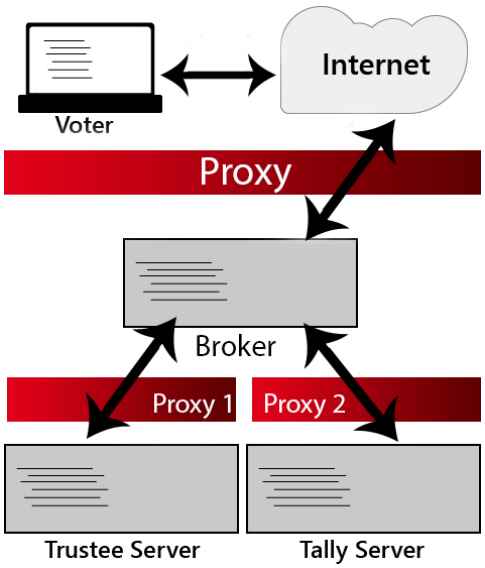


Figure 1: Voting Framework

fears[1]. Researchers consider voting as hard and suggest physical redundancy such as tally papers to accompany any online system [4]. The reason is that online voting is not able to to deal with compromises after they have occurred like in the case of online banking. In e-banking, transactions, statements, and logs allow customers to detect fraudulent transactions. The banking fraud is considered a marginal cost of doing business. Internet voting systems are not similar since they deny fine-grained logs that may compromise the identity of the voters. In this paper, we show the two facets of the game. We propose a simple and usable voting protocol using cryptographic foundations. Simplicity allows us to focus on the security of the implementation and the resilience against remote "black hat" attacks.

3 Voting Framework and Protocol

The voting framework is depicted in Fig. 1. The voter uses an end-device browser and connects through an Internet proxy or an anonymization service to a broker. The proxy has for goal to protect the anonymity of the voter. The broker has the front-end of the voting system and its goal is to provide end-to-end secure channels in between the voter and two back end servers: the trustee server which is responsible of distributing sealed envelopes, and the tally server which is responsible of receiving and counting the encrypted votes. The broker ensures that these two parties are not colluding and even does not know the address of each other. At the cryptography level, the voting protocol is based on a previous work of a subset of the authors [9]. Its threat model, cryptosystem, and communication model are summarized next.

[1] `http://www.reuters.com/article/us-france-election-cyber-idUSKBN16D233`

3.1 Threat model

Our voting protocol strives for the following requirements: (1) We assume that the trustee correctly follows the protocol and verifies the integrity of the election. Still, the trustee is not trusted to link votes and voters. Its threat model is honest-but-curious. (2) The tally server is not trusted to link votes and voters, to follow correctly the protocol or to give credible results. (3) The voter is not trusted to vote only once, or to follow the protocol. Any kind of cheating by the tally or the voters must be detected by the trustee. (4) The broker is the front-end of the voting system and is trusted as a secure and anonymous communication channel in between the voter and the two back ends. Therefore the security of the model is based on distributing trust and liability among multiple authorities allowing them to collectively detect if any one party is dishonest.

3.2 Homomoprhic encryption

We use Paillier's cryptosystem [10] for its ability to add votes under encryption. The trustee computes a public and a private key as follows: It first chooses two random large prime numbers p and q, computes $N = pq$ satisfying $\gcd(N, \phi(N)) = 1$, and considers the group $G = \mathbb{Z}^*_{N^2}$. It also considers $g \in G$ of order N. The public key is composed of N and g. The private key is composed of the Carmichael function $\lambda(N)$ and its inverse $\mu(N)$. The private key is kept at the trustee or can be divided using a (k, n) secret sharing scheme among n authorities. In this case, the decryption of the final results and the verification of the election is constrained by having at least k authorities provide their secret shares. The public key is distributed among all the parties. Using the public key, one encrypts a message $m \in \mathbb{Z}_N$ by picking a random number $r \in \mathbb{Z}_N$ and computing $c = g^m r^N \mod N^2$. Only the trustee can decrypt by computing the discrete logarithm of $c^{\lambda(N)} \mod N^2$ and obtaining $m\lambda(N)$. Then it multiplies by $\mu(N) \mod N$ and gets m. This cryptosystem is assymetric (public/private keys), probabilistic and IND-CPA (Indistinguishable under chosen plain-text attack). The additive homomorphic properties are:

$$\mathcal{E}_{\mathsf{pk}}((m_1 + m_2) \mod N) \equiv \mathcal{E}_{\mathsf{pk}}(m_1) * \mathcal{E}_{\mathsf{pk}}(m_2) \mod N^2$$

$$\mathcal{E}_{\mathsf{pk}}((a * m) \mod N) \equiv \mathcal{E}_{\mathsf{pk}}(m_1)^a \mod N^2$$

We presented a high performance implementation of this cryptosystem and enumerated several applications to secure cloud computation in [8].

3.3 Voting protocol

In its most basic version, our protocol allows binary votes (`Yes` or `No`, 0 or 1, `Bob` or `Alice`). The voter solicits envelopes from the trustee. The trustee decomposes each choice into two

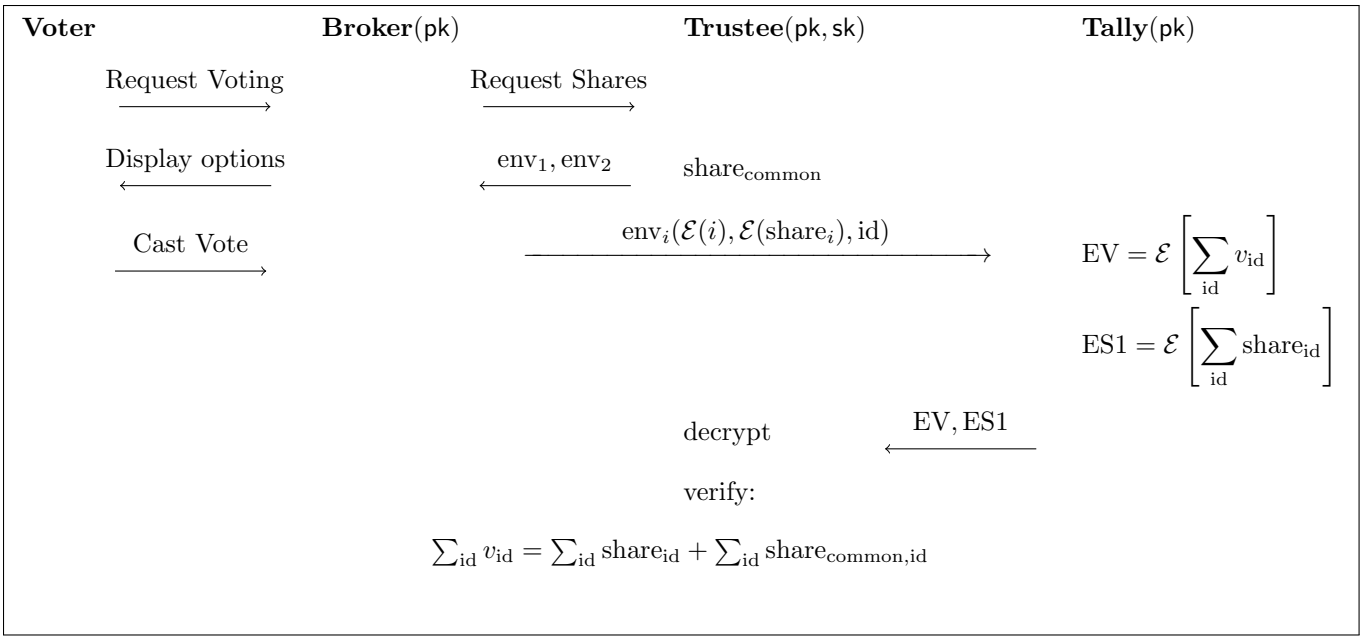


Figure 2: Voting protocol communication diagram

shares. One of the shares is common across the two choices:

$$0 = share_0 + share_{common}$$
$$1 = share_1 + share_{common}$$

In practice, the server starts by choosing $share_{common}$ as a random integer then computes the values of $share_0$ and $share_1$. For example, if by chance $share_{common} = -113$ then $share_0 = 113$ and $share_1 = 114$. This splitting scheme has an important property that can be used for verification:

The trustee encrypts the shares, gives the vote an id and generates two envelops:

$$env_1 = (\mathcal{E}_{\mathsf{pk}}(0) \mid\mid \mathcal{E}_{\mathsf{pk}}(share_0) \mid\mid id)$$
$$env_2 = (\mathcal{E}_{\mathsf{pk}}(1) \mid\mid \mathcal{E}_{\mathsf{pk}}(share_1) \mid\mid id)$$

The communication diagram of the protocol is shown in Fig. 2. We discuss more details, possible malleability attacks and countermeasures in [9]. In this paper, we put more focus on the web application and its security.

4 Implementation framework

We implement a RESTful sElect broker based on Node.js[2] and the V8 JavaScript engine. Node.js has an event driven, non-blocking I/O model and supports CRUD operations through

[2] `https://nodejs.org/en/`

Table 1: Election entry example

election_id	1544450091397
election_name	sElect Demo
email	basselrawda@hotmail.com
recipients	cheftainyoung@mail.com; bob@moon.com; etc.
password	$2b$10$Qi3fj3O/MbN6xjr.kPvo9OpWvbAN9p2
option0	Sugar
option1	Aspartame
question	what sweetener do you prefer?
startdate	2018-12-10
enddate	2018-12-22

Table 2: Common share entry example

election_id	1544450091397
voter_id	1853990
sharecommon	88176
voted	0

Table 3: Vote entry example

share	247050361716832635283720742362774836 59
election_id	1544450091397
vote	91065145015004310003321220265085311329

HTTP requests. We use the Node Package Manager (NPM)[3] which is a JavaScript package manager including built-in modules (e.g. Express) and community libraries (e.g. BigInt). Express[4] is a light-weight web application framework that helps organizing the web application following the MVC model, managing routes and handling requests and views. BigInt[5] supports cryptographic operations such as public and private key generation, encryption and hashing.

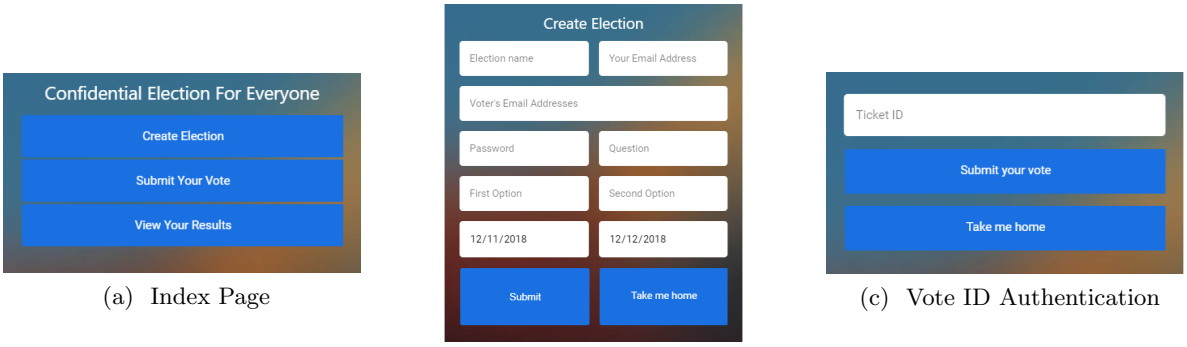
Our template engine is Handlebars[6] to generate dynamic and on-demand web pages. The database back-ends at both the trustee and the tally are MySQL[7] servers. BodyParser[8] is an Express middleware module to parse the incoming HTTP requests and handle data representations such as JSON, buffer, string and URL-encoded.

BCrypt[9] is used for password hashing. BCrypt is based on the Blowfish cipher[10]. Its robustness is parametrized by the number of salt rounds to perform. Adding one salt round

[3] `https://www.npmjs.com/`
[4] `https://expressjs.com/`
[5] `https://www.npmjs.com/package/big-integer`
[6] `https://handlebarsjs.com/`
[7] `https://www.mysql.com/`
[8] `https://www.npmjs.com/package/body-parser`
[9] `https://www.npmjs.com/package/bcrypt`
[10] `https://en.wikipedia.org/wiki/Blowfish_(cipher)`

doubles the time required to brute-force the hash. Cryptr[11] is used for hashing the vote Ids and the election tokens in URLs. Paillier-js[12] is used for encrypting shares and counting votes under encryption.

The trustee server database contains a table for the elections and a table for the common shares. For example, an election entry is shown in Table 1 and a common share entry is shown in Table 2. The tally server database has a table for the collected votes. An example and the table schema are shown in Table 3.



(a) Index Page

(b) Create Election Page

(c) Vote ID Authentication

(d) Voting View

(e) Request Results

(f) View Results

Figure 3: Screen-shots of our broker front-end.

5 Functional Scenarios

The broker application hides the complexity from the end-user and provides a usable interface for election creation and vote casting. Screen-shots of the different views of our application are shown in Figure 3. Next we detail the supported functional scenarios.

5.1 Create Election

An administrator willing to create a new election has to fill and submit a form. A POST request is then sent to the web server of the broker. The web server stores the received information, including the election name, election ID, encrypted password, admin email, voter emails, the

[11] `https://www.npmjs.com/package/node-crypt`
[12] `https://www.npmjs.com/package/paillier-js`

poll question and options, start and end dates. The broker sends a request to the trustee server which creates a voter id and a common share for each voter. The voter ids are returned to the broker and distributed to voters through the mailing service. The mailing service also sends to the admin the election id and the public link to view the election results. The broker gets the public key of the trustee homomorphic encryption and shares it with the tally. The tally initializes the database table based on the received election id.

5.2 Submit Vote

The voter starts by entering the voter id. Once submitted, a request will be sent from the broker to the trustee server in order to check if the voter id is valid and whether the voter already voted or not. In case of eligibility the trustee responds with two encrypted envelops corresponding to the two options of the survey. The voter seamlessly chooses an envelope by picking an option and submit. The broker relays the envelop to the tally this time and registers the vote act locally. In this way the broker can detect votes drop from the tally in case of cheating. The tally extracts the vote and the verification share from the envelope and adds them to the sum of votes and the sum of shares, respectively.

5.3 Request Results

At the end of the election, the broker requests the results from the the tally, the web server sends a request to the trustee server for validation. The trustee decrypts the total share and total vote using its private key. Then it validates the votes by asserting the equality:

$$\sum \text{votes} = \text{Total shares} + \text{Total commons}$$

The total commons is computed by the trustee using the list of ids of casted votes. This list is sent by the tally and verified by the broker. if the verification holds, the results will be calculated using:

$$\text{Option}_0 = \text{all voted} - \sum \text{votes}$$
$$\text{Option}_1 = \sum \text{votes}$$

The results will be sent back to the broker for display.

6 Security and Performance assessment

It is clear from our implementation experience that it is not sufficient to have the cryptography correct in order to create a secure online voting system, not even close. Therefore, we conduct an extensive penetration testing assessment of sElect using a series of black hat tools. We also assess the performance of sElect.

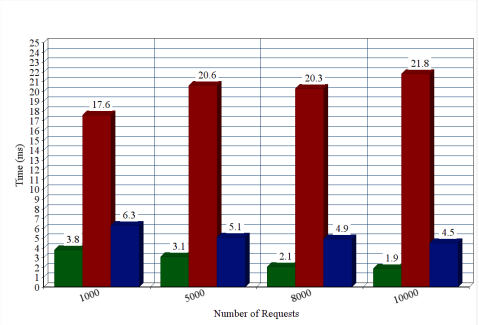


Figure 4: Vote submission performance simulation

6.1 Security assessment

We tested our implementation in black box mode against many different attacks and tools, we found that the system was mainly vulnerable to the following threats:

- SQL injection. Tests were accomplished using SQLMap (`http://sqlmap.org/`)
- Cross-site scripting. Tests were accomplished using Xsser (`https://tools.kali.org/web-applications/xsser`)
- Brute force hash and password cracking attacks for weak admin passwords.

The framework is further available for testing in form of a virtual machine that can be downloaded at `https://drive.google.com/file/d/1Et7-6Wt1dUFbOjnkAtM_KP83qntU_U25/view?usp=sharing`. The VM has a video with instructions to configure and run the voting web service.

6.2 Performance tests

We simulated stress conditions using Artillery (`https://artillery.io/`), a modern load testing toolkit. The vote submission performance is depicted in Fig. 4. The create election performance is shown in Fig. 5. The lowest 1%, the average and the lowest 5% response times are represented in the histogram bars (from left to right).

We also simulated flooding attacks which may result in vote rejection and an increase in request handling time. Results are shown in Fig. 6 and Fig. 7. Results show the importance of deploying load balancing and anti-DDoS countermeasures at the election time. In our scheme envelopes may be distributed prior to the election time.

7 Conclusion and Future Work

Web developers may mistakenly estimate that implementing an election application is straightforward. However, security is a substantial issue that is critical to maintain fair and anonymous

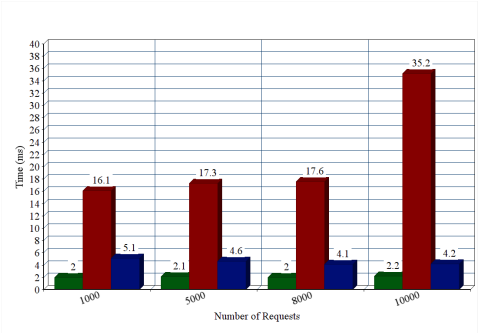


Figure 5: Election creation performance simulation

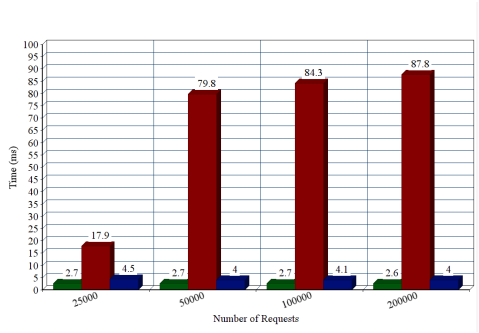


Figure 6: DDoS simulation of processing response time

voting. Security in voting systems is challenging given the multitude of attack vectors and vulnerabilities. In this paper, we have proposed sElect, an implementation of a binary voting system using homomorphic encryption, a broker and two non-colluding parties. We stress on the fact that getting the cryptography right is not enough. We showed that the system may be vulnerable to a series of attacks ranging from SQL injection and cross-site scripting to flooding and DDoS attacks. We discussed the technological choices of our implementation, its database schema, functional scenarios, performance and security assessment.

References

[1] B. Adida. *Advances in cryptographic voting systems*. PhD thesis, Massachusetts Institute of Technology, 2006.

[2] L. P. Alonso, M. Gasco, D. Y. M. del Blanco, J. A. H. Alonso, J. Barrat, and H. A. Moreton. E-voting system evaluation based on the council of europe recommendations: Helios voting. *IEEE Transactions on Emerging Topics in Computing*, 2018.

[3] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (sok). In *Principles of Security and Trust*, pages 164–186. Springer, 2017.

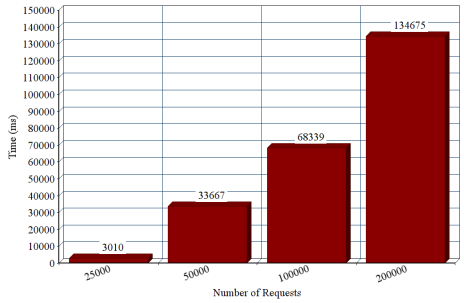


Figure 7: DDoS simulation of number of rejected votes

[4] Bruce Schneier. Voting Security. IEEE Security & Privacy. `https://www.schneier.com/essays/archives/2004/07/voting_security.html`, July/August 2004. [Online; accessed 2015-07-24].

[5] S. Estehghari and Y. Desmedt. Exploiting the client vulnerabilities in internet e-voting systems: Hacking helios 2.0 as an example. *EVT/WOTE*, 10:1–9, 2010.

[6] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology-EUROCRYPT 2000*, pages 539–556. Springer, 2000.

[7] P. McCorry, S. F. Shahandashti, and F. Hao. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*, pages 357–375. Springer, 2017.

[8] M. Nassar, A. Erradi, and Q. M. Malluhi. Paillier's encryption: Implementation and cloud applications. In *2015 International Conference on Applied Research in Computer Science and Engineering (ICAR)*, pages 1–5. IEEE, 2015.

[9] M. Nassar, Q. Malluhi, and T. Khan. A scheme for three-way secure and verifiable e-voting. In *15th ACS/IEEE international conference on computer systems and applications (AICSSA18)*, 2018.

[10] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[11] S. Wolchok, E. Wustrow, D. Isabel, and J. A. Halderman. Attacking the washington, dc internet voting system. In *Financial Cryptography and Data Security*, pages 114–128. Springer, 2012.