Apollo

A secure, anonymized voting system using the Paillier cryptosystem

Varun Mohan, Rahul Sridhar, Lawrence Sun, Kevin Zhu
{vmohan, rsridhar, sunl, kevinzhu}@mit.edu

6.857 Project Report

May 11, 2016

1 Introduction

With the growing number of problems with traditional voting methods, there has been mounting theoretical interest in leveraging cryptography to run end-to-end verifiable elections [1, 15]. At the same time, voters have expressed rising interest in Internet-based voting schemes due to the almost unparalleled ease of access provided by such a service [13]. However, numerous challenges must be overcome before web-based voting can become a reality: voter coercion, malware, DDoS attacks, are all very real problems that have the potential to cast complete doubt on the outcome of an election [14].

Although such problems make Internet voting unsuitable for presidential elections, it may be appropriate for use in elections with lower stakes, such as MIT's elections for positions in the Undergraduate Association. While the possibility of a targeted, malicious actor is low, basic guarantees about vote anonymity and election integrity are still useful in convincing candidates and voters to trust the outcome of the election.

In recent years, a small number of end-to-end verifiable voting systems have been created. Many of them, including Scantegrity and Wombat Voting, continue to use paper ballots, which have the advantage of being able to conduct a manual recount in the case of wide-spread, suspected fraud [6, 7]. The most prominent example of an end-to-end, Internet-based voting scheme is the open-source system Helios, which has been used and tested in a number of small-scale, low-stakes elections [2].

End-to-end auditable voting systems [12] have two main goals: anonymity and election integrity. Most systems accomplish anonymity through a combination of homomorphic tallying (summing ballots encrypted by the voters to produce an encryption of the result of the election) and mixnets (routing protocols that use a series of proxy servers to shuffle the incoming ballots). Helios, for example, uses the ElGamal cryptosystem with the Sako-Kilian mixnet. As with most voting schemes, Helios achieves end-to-end integrity using interactive zero-knowledge proofs (ZKPs) that convince each individual voter that (1) their vote was included in the election result, and that (2) no extra votes were added when computing the result [2].

In creating Apollo, our goal was to create an Internet voting scheme suitable for use in the MIT community that provides similar guarantees to Helios. Apollo takes advantage of a number of features of the MIT ecosystem (such as the widespread use of certificates for authentication) to side-step much of the complexity necessary in Helios.

2 Design Goals

Apollo was designed to be distributed, anonymous, and end-to-end auditable.

- *Distributed*: No single entity should control the entirety of the election process, making Apollo a distributed system and reducing the consequences of one module being compromised.
- *Anonymous*: Apollo should maintain voter anonymity, in that no party other than the voter should be able to determine who a voter voted for.
- *End-to-end auditable*: Voters should be able to confirm that the results of an election are valid, namely that their vote was counted correctly and the results of the election were tallied properly.

For our security/attack model, anyone can be an adversary, including the modules of our system. We assume that at most one module of our system is malicious. Under this model, Apollo still successfully maintains the anonymity of voters and end-to-end integrity.

3 Modules

Apollo makes use of five modules: a registrar, talliers, an aggregate tallier, an authority, and voters. All of these modules are capable of supporting several concurrent elections.

Voter: A voter is someone with an MIT certificate who is a specified participant of an election, for example students in 6.857. Voters interact with Apollo by registering with the registrar and sending their encrypted votes to talliers.

Registrar: The registrar's primary function is to authenticate voters. To register for an election, a voter presents his or her MIT certificate to the registrar, and the registrar confirms that the user is an eligible voter. The registrar maintains a list of encrypted votes and registered users.

Tallier: Talliers receive encrypted votes from users and aggregate them. Talliers are added to the system by registering with the registrar. When a new election is created, the registrar selects a subset of the registered talliers to be allocated for the election. Talliers make use of the fact that our encryption scheme is homomorphic, allowing them to sum together votes despite only knowing the ciphertexts.

Aggregate Tallier: The aggregate tallier computes the total encrypted sum of the votes using the encrypted sums provided by each tallier. Once an election has finished, each tallier sends its aggregate total of votes to the aggregate tallier. Then, the aggregate tallier sends its sum of votes to the authority.

Authority: When an election is created, the registrar requests that the authority generate a public/private key pair. The public key is used by voters to encrypt their votes, and the private key is used by the authority to decrypt the final results of the election.

4 Cryptographic Security

Apollo makes use of the Paillier cryptosystem to encrypt all votes, granting anonymity to voters, as only the tallier and registrar, which do not possess the private key, can determine which voter cast a given vote. Additionally, Apollo uses zero-knowledge proofs to prevent voters from sending invalid votes and serve as an end-to-end check that the election was tallied correctly.

4.1 Paillier Encryption

The Paillier cryptosystem is an asymmetric public-key encryption algorithm [3]. It is extremely useful because its encryption is additively homomorphic: given two plaintexts v_1, v_2 and public/private keys (pk, sk) we have

$$D_{sk}(E_{pk}(v_1) \cdot E_{pk}(v_2)) = v_1 + v_2$$

Because of this property, the Paillier cryptosystem is widely used in voting. It can be broken up into the following three components.

4.1.1 Key Generation

To create a public/private key pair, first generate two large primes p, q that satisfy $\gcd(pq, (p-1)(q-1)) = 1$. Let $n = pq$ and $\lambda = \varphi(n)$. Generate a random g modulo n^2 that is relatively prime to n^2. It is ensured that n divides the order of g by computing the existence of

$$\mu = (L(g^\lambda \pmod{n^2}))^{-1} \pmod{n}$$

where $L(u) = \frac{u-1}{n}$. If this property holds, the public key pk is then (n, g), and the private key sk is (λ, μ).

4.1.2 Encryption

Given a message m, generate a random number $r \in \mathbb{Z}_n^*$. The encrypted message is then computed as

$$c = g^m \cdot r^n \pmod{n^2}$$

4.1.3 Decryption

Given a ciphertext $c \in \mathbb{Z}_{n^2}^*$, the plaintext is simply

$$m = L(c^\lambda \pmod{n^2}) \cdot \mu \pmod{n}$$

4.2 Multi-Candidate Voting Scheme

We now specify the process of how voters convert their ballots to an integer. This scheme is taken from [5]. For an election with M candidates, we give each candidate a unique ID from 0 to $M - 1$. We assume that there are fewer than N voters. To vote for candidate i, the voter encrypts the message N^i.

After tallying all the encrypted votes, the decrypted result will be of the form

$$r = \sum_{i=0}^{M-1} v_i N^i$$

because the Paillier cryptosystem is additively homomorphic. After representing r in this form, v_i represents the number of votes received by candidate i. Furthermore, the message N^i represents a vote for candidate i since fewer than N people can vote for a candidate, so a vote meant for a particular candidate will never be counted in another candidate's vote count. It should be noted that during key generation, the modulus n must be larger than N^M for this scheme to work.

4.3 Zero-Knowledge Proofs

We make use of two zero-knowledge proofs (ZKPs) in our voting protocol. One is used to verify that an encrypted vote is valid, while the other is for the election authority to prove that the final decryption was performed correctly. Proofs for the validity of the following schemes can be found in [4].

4.3.1 Validity of Vote

When voters send their votes to the tallier, they must prove to the tallier their vote is an encryption of one of $\{1, N, N^2, \ldots, N^{M-1}\}$. To do this, a ZKP takes place between a voter and tallier. Suppose the voter wishes to send a vote of N^k whose ciphertext is:

$$C = g^{N^k} \cdot r^n \pmod{n^2}$$

Then the following protocol takes place:

1. The voter randomly generates $e_0, \ldots, e_{M-1}$ from 1 to 2^b and $z_0, \ldots, z_{M-1}$ from 1 to N^2 and computes, for each i between 0 and $M - 1$:

$$a_i = z_i^n \cdot \left(\frac{C}{g^{N^i}}\right)^{-e_i} \pmod{n^2}$$

2. The voter replaces a_k with v^n, where v is randomly chosen between 1 and n^2.

3. The voter then sends $a_0, a_1, \ldots, a_{M-1}$ to the tallier.
4. The tallier sends a random number e_{chall} to the voter, generated randomly between 1 and 2^b, where b is the numbers of bits in $\sqrt{n}$.
5. The voter sets e_k to satisfy
$$\sum_{i=0}^{M-1} e_i = e_{\text{chall}} \pmod{2^b}$$
and re-computes z_k as
$$z_k = v \cdot r^{e_k} \pmod{n^2}$$
6. The voter then sends the numbers $(a_0, \ldots, a_{M-1}, e_0, \ldots, e_{M-1}, z_0, \ldots, z_{M-1})$ to the tallier.
7. The tallier checks
$$\sum_{i=0}^{M-1} e_i = e_{\text{chall}} \pmod{2^b}$$
and for each i:
$$z_i^n = a_i \cdot \left(\frac{C}{g^{N^i}}\right)^{e_i} \pmod{n^2}$$

4.3.2 Validity of Decryption

Once the talliers aggregate their votes, the election authority decrypts the result. Proof must be specified that the election authority performed the decryption correctly. To do this, we make use of another ZKP. Given a ciphertext c and a plaintext m, the following protocol is used:

1. A voter requests that the authority prove the decryption.
2. The authority generates a random number a between 1 and N^2 and sends a^n to the voter.
3. The voter sends a random number e_{chall} to the authority, chosen between 1 and 2^b.
4. The authority computes (r, s) as
$$s = c \cdot g^{-m} \pmod{n^2}$$
$$r = s^{n^{-1} \pmod{\lambda}} \pmod{n^2}$$
5. The authority sends $z = a \cdot r^{e_{\text{chall}}} \pmod{n^2}$ to the voter.
6. The voter checks
$$z^n = a^n \cdot \left(\frac{c}{g^m}\right)^{e_{\text{chall}}} \pmod{n^2}$$

5 Protocol

Our protocol has three distinct phases: election creation, election vote processing, and election termination. These phases demarcate major changes in the state of our system modules.

5.1 Election Creation

To create an election, an election owner sends a list of voters V and candidates C to the registrar, corresponding to the election. The registrar sends a request to the authority to generate the public/private keys. The authority then publishes the public key pk for the election as well as an election ID E_{id}.

The registrar assigns registered talliers to the election greedily based on the amount of work already assigned to them. The number of talliers is proportional to the product of the number of voters and candidates since the amount of work performed to validate each vote grows linearly with the number of candidates due to the aforementioned "Validity of Vote" ZKP. After the talliers accept the election, the registrar publishes V, C, and the list of talliers T. Once this process is complete, the election can begin. These interactions are shown in Figure 1.

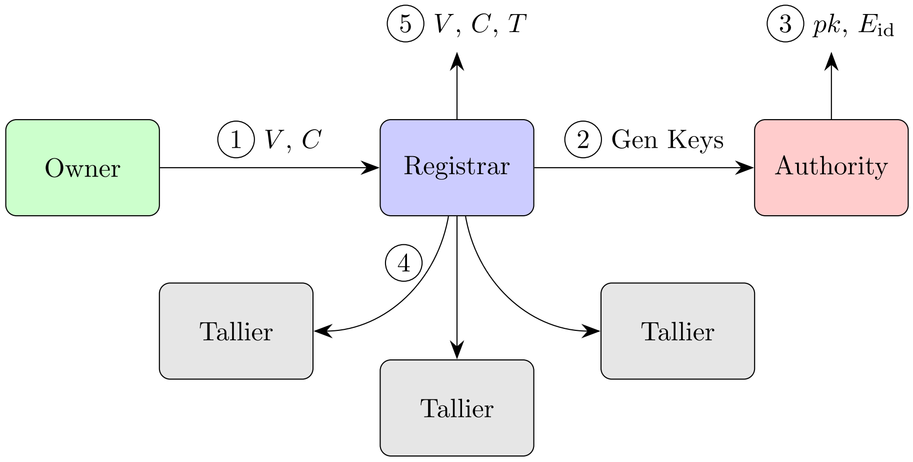


Figure 1: The election creation process.

5.2 Election Process

Once an election has begun, voters send their votes using the following scheme. First, a voter encrypts his or her vote using the Paillier encryption algorithm with the published public keys. The encrypted vote $E(v)$ is then sent to the registrar, which stores $E(v)$ for the associated user. The voter then randomly selects a tallier and sends $E(v)$ to it. The voter also partakes in the "Validity of Vote" ZKP with the tallier.

Once the tallier verifies the vote, it queries the registrar for whether voter's $E(v)$ is legitimate. The registrar responds positively if and only if the voter has not already voted and $E(v)$ is equivalent to the one submitted by the voter. Once the tallier acknowledges a positive response, the registrar marks the voter has having voted, and the tallier adds $E(v)$ to its running tally. Finally, the voter receives a response indicating that his or her vote was successfully processed or was invalid. This protocol is outlined in Figure 2.

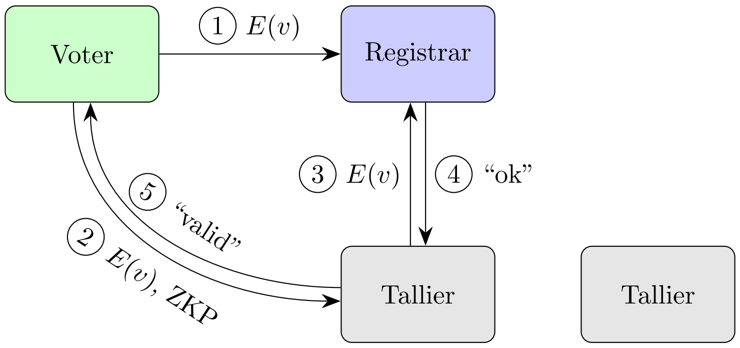


Figure 2: The election vote processing protocol.

5.3 Election Termination

During election termination, results are computed and published. Afterwards, voters can verify the end-to-end election integrity.

5.3.1 Computing Election Results

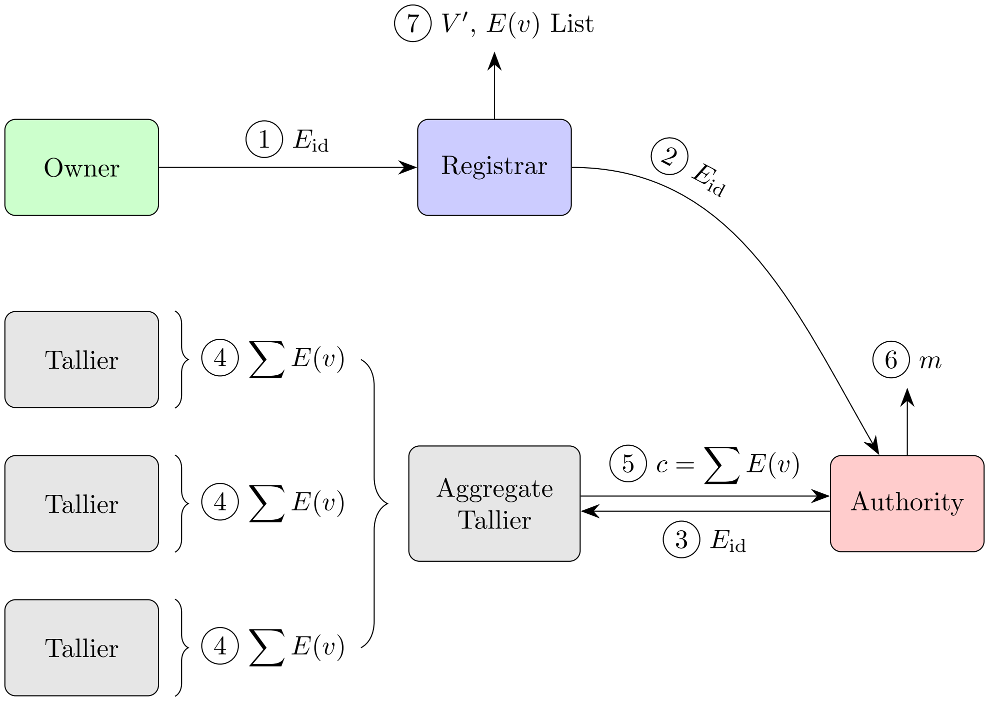


Figure 3: Computation of election results.

To end an election, the owner sends an "end election" request with E_{id} to the registrar. The registrar then requests the authority to compute the election result. The authority in turn asks the aggregate tallier to compute the total encrypted election tally. The aggregate tallier computes this value by first asking all talliers to stop servicing the election and send in their local encrypted tallies. The aggregate tallier simply computes the aggregate tally c by homomorphically adding the associated encrypted tallies and sends c back to the authority. The authority decrypts the encrypted tally and publishes the election results m. The registrar publishes a randomized, anonymous list of all encrypted votes as well as a list of participating voters V'. The protocol is shown in Figure 3.

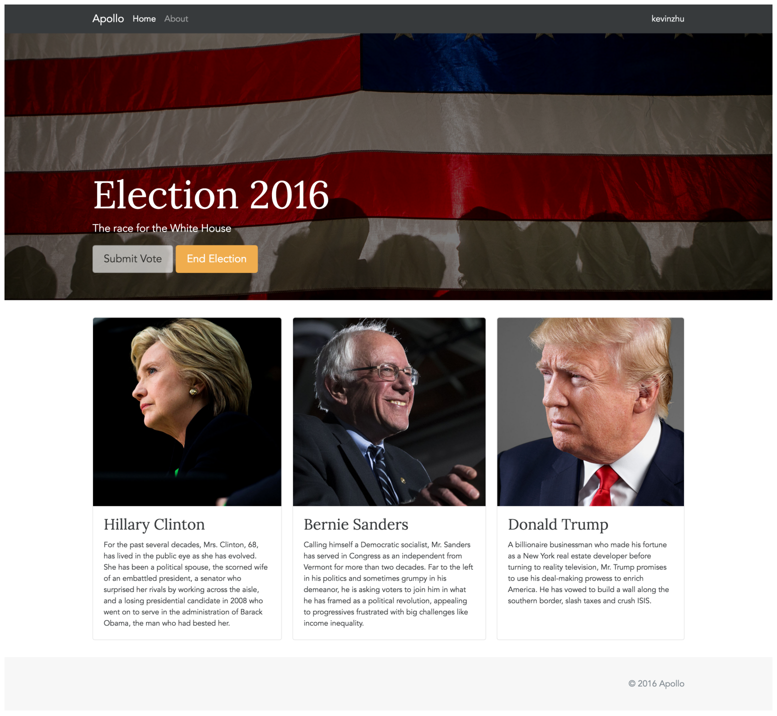
5.3.2 Voter–Election Verification

The voter verifies the election's integrity by first checking that their encrypted vote is on the list of published encrypted votes. The voter also checks that he or she is in V' and that the number of voters in V' is equal to the number of encrypted votes. This ensures that the voter's vote was not tampered with and that additional malicious votes were not added to the list of encrypted votes without corresponding participants.

The voter computes the aggregate encrypted tally c by multiplying all the encrypted votes. The voter then engages in the "Validity of Decryption" ZKP with the authority to ensure that the computed election result m is a valid decryption of c. This concludes the end-to-end election validation phase.

6 Implementation

Apollo was implemented in Python 3.5 as a Flask web application hosted on Heroku. The source code can be found at `https://github.mit.edu/vmohan/Apollo`, and an example election running on the Apollo platform can be found at `https://apollo-voting.herokuapp.com`; note that a valid MIT certificate is required to use the site. The client-side encryption of the vote and ZKPs were written in JavaScript using the `jsbn` library [9]. The Paillier cryptosystem was implemented using cryptographic primitives defined in Python's `pycrypto` library [10].



Hillary Clinton

For the past several decades, Mrs. Clinton, 68, has lived in the public eye as she has evolved. She has been a political spouse, the scorned wife of an embattled president, a senator who surprised her rivals by working across the aisle, and a losing presidential candidate in 2008 who went on to serve in the administration of Barack Obama, the man who had bested her.

Bernie Sanders

Calling himself a Democratic socialist, Mr. Sanders has served in Congress as an independent from Vermont for more than two decades. Far to the left in his politics and sometimes grumpy in his demeanor, he is asking voters to join him in what he has framed as a political revolution, appealing to progressives frustrated with big challenges like income inequality.

Donald Trump

A billionaire businessman who made his fortune as a New York real estate developer before turning to reality television, Mr. Trump promises to use his deal-making prowess to enrich America. He has vowed to build a wall along the southern border, slash taxes and crush ISIS.

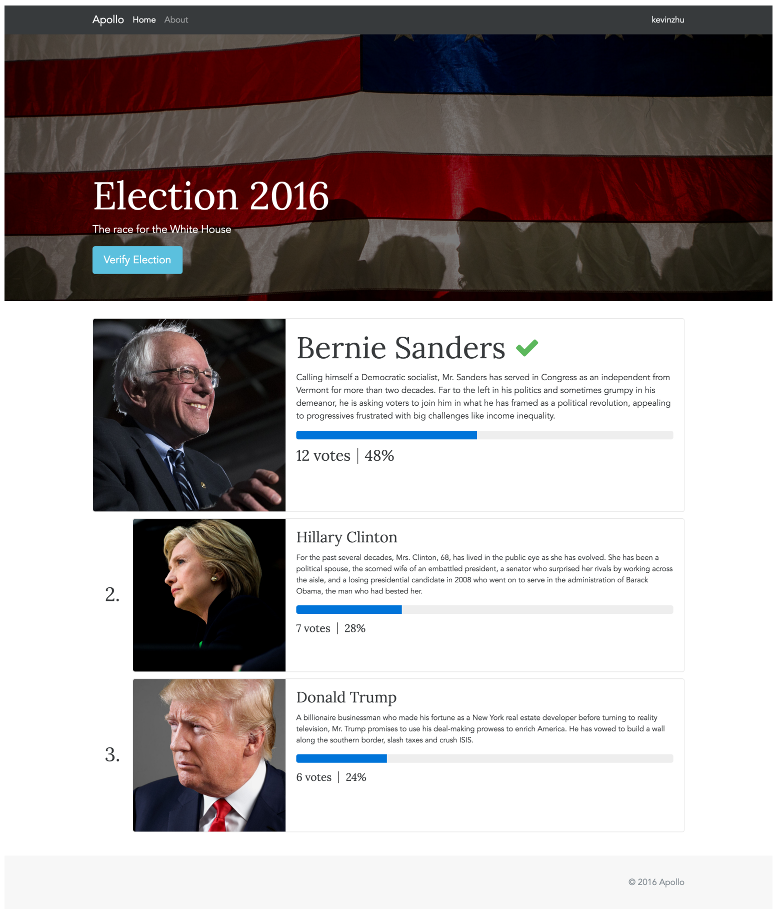
© 2016 Apollo

Figure 4: The voting page for a sample election held using Apollo from the perspective of the election owner.

The above figure shows a screenshot of an example election between the three remaining candidates of major political parties in the 2016 U.S. presidential election. To view this page, a voter must have an MIT certificate matching a username in a preset list of voters. The voter's Kerberos username, which is provided by the certificate, is displayed on the top right of the page. The procedure to vote is as follows:

1. The voter clicks on one of the candidates. A check mark then appears, indicating that the specified candidate was selected.
2. The voter clicks on the "Submit Vote" button. An encrypted version of the vote and parameters for the first ZKP are calculated client-side using JavaScript and sent to the registrar and a tallier. If both the registrar and the tallier accept the vote, the button turns green and displays "Vote Submitted". The encrypted vote is then stored as a cookie on the client for the second ZKP. However, if the vote was rejected, the button turns red and displays "Invalid Vote".
3. If the voter is the election owner, an orange "End Election" button is displayed. When the election owner presses this button, the talliers aggregate their votes and send the results to the authority to be decrypted. On the other hand, if the voter is not the election owner, the button is not displayed, and he or she need only wait for the election to end to see the results.

Once the election has been terminated by the election owner, the application updates with the results of the election, as shown in Figure 5.



Bernie Sanders ✓

Calling himself a Democratic socialist, Mr. Sanders has served in Congress as an independent from Vermont for more than two decades. Far to the left in his politics and sometimes grumpy in his demeanor, he is asking voters to join him in what he has framed as a political revolution, appealing to progressives frustrated with big challenges like income inequality.

12 votes | 48%

2. Hillary Clinton

For the past several decades, Mrs. Clinton, 68, has lived in the public eye as she has evolved. She has been a political spouse, the scorned wife of an embattled president, a senator who surprised her rivals by working across the aisle, and a losing presidential candidate in 2008 who went on to serve in the administration of Barack Obama, the man who had bested her.

7 votes | 28%

3. Donald Trump

A billionaire businessman who made his fortune as a New York real estate developer before turning to reality television, Mr. Trump promises to use his deal-making prowess to enrich America. He has vowed to build a wall along the southern border, slash taxes and crush ISIS.

6 votes | 24%

© 2016 Apollo

Figure 5: The results page for the election shown in Figure 4, which can be accessed once the election has ended by refreshing the voting page.

This page ranks the candidates by the number of votes they received and also displays each candidate's vote totals and percentages. Voters can also verify the election using the "Verify Election" button, which performs the second ZKP. If the client determines that the election is valid, the button turns green and displays "Valid Election"; otherwise, it turns red and displays "Invalid Election".

7 Discussion

Apollo is much simpler in comparison to most other end-to-end verifiable voting schemes. As such, it does not provide exactly the same set of guarantees as other protocols and is vulnerable to a different set of threats.

7.1 Evaluation

In this section we revisit our three design goals and discuss how Apollo achieves each of them.

- *Distributed*: Apollo's tallying system is distributed across multiple machines which improves both security and performance.
- *Anonymous*: Apollo accomplishes voter anonymity through the use of the Paillier cryptosystem, which prevents any module from ever seeing a voter's ballot in plaintext.
- *End-to-end auditable*: This is accomplished through the use of ZKPs as discussed above. Any individual voter can verify both that his or her vote was counted in the final result, and that the result published by the authority is accurate. Note that *any* action that would result in a change in the election outcome by any of the modules will be detected by the end-to-end integrity check.

7.2 Security Threats

Browser Cryptography: Recall that a client's ballot is encrypted in JavaScript running in the client's web browser before it is sent to a tallier. This operation must be done client-side to preserve guarantees about voter anonymity even in the case of malicious or curious talliers. However, cryptography implemented in client-side JavaScript confers a number of significant risks [8]. Namely, the server-client connection is subject to man-in-the-middle attacks that replace the client's JavaScript with malicious code that performs the encryption incorrectly. Apollo addresses this by using HTTPS for all connections between the client and server. Furthermore, Apollo allows more security-minded clients to perform encryption themselves using whatever combination of languages and libraries they choose and send their votes through a POST request to our API.

Another significant critique of JavaScript cryptography is the lack of a cryptographically secure psuedo-random number generator (PRNG). As mentioned above, Apollo avoids JavaScript's default PRNG, using the generator defined in the `jsbn` libary instead.

Malicious Collusion: Recall that the authority is the only module with access to an election's decryption keys. As such, any collusion that compromises voter anonymity (without breaking the underlying cryptography) must involve the authority. If the authority compromises or colludes with an individual tallier while the election is running, then the authority can decrypt and read all the votes sent to that tallier, associating them with the voter who sent in the ballot. Assuming that the election utilizes a large number of talliers, the corresponding attack surface is relatively small.

In contrast, if the authority manages to compromise or collude with the registrar while the election is running, the authority can decrypt and read every ballot in the election and correspond them with the voter who sent it. Note that we can alleviate this attack by ensuring that the registrar is a trusted third party. Nevertheless, the possibility of some vulnerability with the registrar server is significant, and as such this possibility is a considerable weakness in Apollo's protocol. However, to anyone other than the authority, or indeed the registrar itself, the encrypted votes contain no information that could compromise voter anonymity.

Threats to Integrity: Apollo provides strong integrity guarantees (as discussed above) as long as a significant fraction of voters bother verifying the election result. If none, or very few voters do this, attacks by various modules could potentially add or remove votes without detection. For example, a tallier could choose not to send or tally a particular voter's ballot, and if the voter does not verify the election after it is terminated, this change in outcome will go undetected. By making the difficulty of verifying the election as low as possible (one click of a button), we will hopefully increase the proportion of users verifying election integrity to acceptably high levels.

Threats Beyond Model: Like Helios, Apollo cannot and does not try to prevent attacks that involve malware, DDoS, or voter coercion. Assuming that Apollo will be used for elections with lowered-stakes, these threats are outside our model, and are not addressed.

7.3 Extensions

Apollo currently does not support write-in candidates, but they can be integrated into the system without too much difficulty. Namely, the election owner can set a maximum number of candidates for the election greater than the number of registered candidates at the beginning of the election. Voters can then submit write-in votes and add new candidates to the election, who other voters can vote for.

Apollo does not currently have a publicly-accessible interface for creating elections. This should be relatively simple to add to our application.

Another possible extension is to use non-interactive ZKPs via the FiatShamir heuristic [11]. This simplifies the communication required to establish proofs of knowledge while maintaining the same security guarantees.

8 Conclusion

We have presented the design of Apollo, a platform that can be used to create and run trustworthy Internet elections. Apollo provides voter anonymity and end-to-end integrity while avoiding much of the complexity of other systems such as Helios. We hope that Apollo will prove useful, both as an example of a relatively simple end-to-end auditable voting system, and as a secure voting application for the MIT community.

9 Acknowledgements

We would like to thank Ronald L. Rivest and Kevin C. King for their many insights that influenced the design and implementation of Apollo.

References

[1] Lawrence Norden and Christopher Famighetti. America's Voting Machines At Risk. *Brennan Center For Justice* (2015).

[2] Ben Adida. Helios: Web-based Open-Audit Voting. *17th USENIX Security Symposium* (2008).

[3] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *EUROCRYPT* (1999).

[4] Ivan Damgård and Mads Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. *Public-Key Cryptography* (2001).

[5] Andreas Steffen. E-Voting Simulator based on the Paillier Cryptosystem. *MSE Seminar on E-Voting* (2010).

[6] David Chaum, Richard T. Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: End-to-End Verifiability by Voters of Optical Scan Elections Through Confirmation Codes. *IEEE Transactions on Information Forensics and Security* (2009).

[7] Niko Farhi and Amnon Ta-Shma. An Implementation of Dual (Paper and Cryptograhic [sic]) Voting System. `http://www.cs.tau.ac.il/~amnon/Students/niko.farhi.pdf` (2013).

[8] Thomas Ptacek. Javascript Cryptography Considered Harmful. `https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/` (2011).

[9] Tom Wu. RSA and ECC in JavaScript. `http://www-cs-students.stanford.edu/~tjw/jsbn/` (2009).

[10] Dwayne Litzenberger. PyCrypto - The Python Cryptography Toolkit. `https://www.dlitz.net/software/pycrypto/` (2013).

[11] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *CRYPTO* (1986).

[12] Wikipedia. End-to-end auditable voting systems. `https://en.wikipedia.org/wiki/End-to-end_auditable_voting_systems` (2016).

[13] U.S. Vote Foundation. The Future of Voting. `https://www.usvotefoundation.org/sites/default/files/E2EVIV_full_report.pdf` (2015).

[14] Ronald Rivest. Auditability and Verifiability of Elections. `http://courses.csail.mit.edu/6.857/2016/files/L20-Auditability-and-Verifiablity-of-Elections-slides.pdf` (2016).

[15] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an Electronic Voting System. `http://avirubin.com/vote.pdf` (2004).