Koinonia: **Verifiable E-Voting with Long-term Privacy**

Huangyi Ge
geh@purdue.edu
Purdue University

Sze Yiu Chau
schau@purdue.edu
Purdue University

Victor E Gonsalves
vgonsalv@purdue.edu
Purdue University

Huian Li
huili@iu.edu
Indiana University

Tianhao Wang
wang2842@purdue.edu
Purdue University

Xukai Zou
xkzou@cs.iupui.edu
Indiana University

Ninghui Li
ninghui@cs.purdue.edu
Purdue University

ABSTRACT

Despite years of research, many existing e-voting systems do not adequately protect voting privacy. In most cases, such systems only achieve "immediate privacy", that is, they only protect voting privacy against today's adversaries, but not against a future adversary, who may possess better attack technologies like new cryptanalysis algorithms and/or quantum computers. Previous attempts at providing long-term voting privacy (dubbed "everlasting privacy" in the literature) often require additional trusts in parties that do not need to be trusted for immediate privacy.

In this paper, we present a framework of adversary models regarding e-voting systems, and analyze possible threats to voting privacy under each model. Based on our analysis, we argue that secret-sharing based voting protocols offer a more natural and elegant privacy-preserving solution than their encryption-based counterparts. We thus design and implement Koinonia, a voting system that provides long-term privacy against powerful adversaries and enables anyone to verify that each ballot is well-formed and the tallying is done correctly. Our experiments show that Koinonia protects voting privacy with a reasonable performance.

ACSAC '19, December 9–13, 2019, San Juan, PR, USA

ACM ISBN 978-1-4503-7628-0/19/12…$15.00
https://doi.org/10.1145/3359789.3359804

ACM Reference Format:
Huangyi Ge, Sze Yiu Chau, Victor E Gonsalves, Huian Li, Tianhao Wang, Xukai Zou, and Ninghui Li. 2019. Koinonia: Verifiable E-Voting with Long-term Privacy. In *2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3359789.3359804

1 INTRODUCTION

Election is the cornerstone of modern democracies; however, the correct functioning and public trust of election depends on the equipment/technology used to cast and count ballots. The events surrounding the 2000 United States presidential election and the 2012 election interference by hurricane Sandy demonstrated the shortcomings of conventional voting technology, and amplified the needs to develop more advanced voting technologies. Some countries have experimented with online voting systems in their elections [67]. However, due to stringent security and privacy requirements [21, 39], online voting systems have yet to be widely deployed.

An e-voting system typically has many Voters, a central Server, and multiple Authorities who help to ensure voting privacy even when the Server is malicious. While e-voting protocols and systems have been extensively studied, we find that a comprehensive and careful analysis of the adversary model is still lacking. For example, many e-voting protocols provide voting privacy only against today's adversaries. That is, they use public-key cryptographic primitives with parameters that are believed to be secure today. We call this level of privacy "immediate privacy". However, in the not-so-distant future, faster computers, better cryptanalysis algorithms and/or the emergence of quantum computers may compromise the privacy of these votes. In this paper, we call privacy against such adversaries "long-term privacy".

In the literature, "long-term privacy" has been dubbed "everlasting privacy". Many attempts at achieving this require trusting parties that are not trusted for immediate privacy, resulting in significantly weaker privacy guarantee. For example, Helios [1], perhaps the most prominent e-voting system, uses threshold cryptography to ensure that the Server, even if malicious, cannot compromise voter privacy without collusion from other Authorities involved in the voting protocol. However, since Helios publishes encrypted ballots in a public bulletin, it provides only immediate privacy. Demirel et al. [33] proposed enhancements that aim at adding everlasting privacy to Helios. In the new protocol, the server, instead of publishing encrypted votes in the public bulletin, publishes cryptographic commitments (which are information-theoretic hiding) of encrypted votes. We note that while this provides long-term privacy against an external advesary, it does not provide long-term privacy against the Server, who still sees the ciphertexts and can recover the plaintext in the future when the encryption scheme can be broken. Therefore, while no single entity is trusted for immediate privacy, trust in the Server is necessary for long-term privacy. The idea for achieving everlasting privacy in Demirel et al. [33] comes from Moran and Naor [48–50], and the protocols therein suffer from the same weakness.

More recently, Pino [31] proposed a voting protocol to deal with potential threats posed by quantum computers. That protocol builds on [27], which uses cryptographic commitments. In such protocols, privacy is information-theoretic and is ensured so long as at least one single Authority is honest, and integrity is guaranteed using cryptographic commitment and zero-knowledge proofs, whose security is often based on hard problems such as discrete logarithm, and can be broken with quantum computers. Pino [31] developed new commitment and zero-knowledge proof protocols whose security are based on hard problems involving lattices, so that integrity can be ensured even when quantum computers appear, so long as these problems remain hard.

We point out that previous work [31] is inadequate in dealing with the threats that quantum computers pose to today's e-voting. First, quantum computers in the future cannot retrospectively post threats to the **integrity** of voting conducted **today**, since vote counting and verification are performed in the present (assuming currently the adversary does not already have access to quantum computers to break the integrity). However, quantum computers in the **future** post serious threats to the **privacy** of voting conducted today. This is because the voting transcripts are often made available to the public to enable verification of integrity and correctness, but if the voting transcript contains ciphertexts of votes under encryption schemes that can be broken by quantum computers, then voting privacy can be compromised in the future. This threat is not addressed by [31].

Second, when dealing with threats of future adversaries, one has to also consider how the underlying communications are protected. Most papers on voting protocols make the standard assumption that communications between honest parties are private. In practice, however, such communications are usually protected using standard *computationally secure* cryptographic techniques, some of which are based on the assumptions that certain problems are difficult to solve. Such assumptions, however, might no longer hold in the future, possibly in the face of new advancements in algorithm design and quantum computers. This is true even in the relatively new field of quantum-safe crypto [1]. Some of the algorithms which are believed to be quantum-safe today might turn out to be not as secure as expected in the future. Hence, if one is concerned about future adversaries, especially when the adversary can monitor all the communications, then simply encrypting all the networked traffic might not be sufficient to assure voting privacy.

In order to better understand the threat of future adversaries to voting privacy, we propose a framework of adversary models. There are two dimensions. The first is what computational capabilities the adversary has, along which we classify adversaries into **current**, **future**, and **unbounded**. The second dimension is what the adversary can observe, along which we classify adversaries into **local** and **global**. (See Section 2 for details.) Composing these two dimensions gives us a lattice of adversary capabilities. Our analysis suggests that achieving privacy against the strongest adversarial model, global and unbounded, is extremely difficult, if not impossible in practice. However, it is possible to achieve voting privacy against the two models that are slightly weaker in the lattice, namely, a local, unbounded adversary (who has unbounded computational power, but does not conduct global eavesdropping), and a global, future adversary (who conducts global eavesdropping, but is unable to break quantum-safe crypto primitives used by the e-voting system).

Motivated by the principle of *"current integrity; future privacy"*, we design and implement an e-voting system dubbed **Koinonia**, a Greek word that means "sharing". The core Koinonia protocol bears a resemblance to the protocol in [27], however, we show that even simple additive secret sharing scheme can be used to satisfy voting privacy needs in our settings, which makes the solution easier to comprehend and implement. Additionally, for integrity and correctness guarantees, each ballot in Koinonia consists of cryptographic commitments of a voter's vote shares, as well as proofs that the shares are well-formed. Koinonia achieves everlasting (*i.e.*, information theoretic) voting privacy, as long as at least

[1] It is believed that quantum computers can only do limited damages to many of today's standardized hash and symmetric key algorithms, hence most of the quantum-safe research focus on public-key crypto, including the likes of digital signatures and key exchange algorithms.

one Authority is honest, and the adversary does not see the cleartext communication between a voter and this honest Authority (either because the adversary cannot break the encryption used to protect the communication, or when the adversary becomes capable of breaking the encryption scheme, the communication transcript no longer exists). We have also enhanced the basic protocol to improve its robustness against misbehaving entities.

The protocol in Koinonia is computationally efficient; only a small number of modular exponentiations are needed for each ballot. Verification of voting integrity can be conducted by any parties with low computational cost. We have implemented Koinonia and open-sourced the code [2].

The contribution of the paper is as follows:

- We consider the threats of future adversaries to e-voting, and identify a classification of possible adversaries. Using this framework, we are able to identify weaknesses of existing attempts at providing long-term privacy in protocols such as [33, 48–50].
- Our analysis suggests that a secret-sharing based voting scheme is more natural and appropriate in handling the threats of advanced future adversaries, especially in the face of quantum computers and uncertainties regarding the security of today's computationally secure crypto algorithms.
- We design and implement Koinonia, which uses an e-voting protocol that achieves strong long-term privacy guarantees, at a reasonably low cost of communication and computation. Koinonia also allows anyone to verify the final results.

The rest of our paper is organized as follows. Section 2 presents our framework of adversaries. Section 3 presents the high-level ideas used in Koinonia. The Koinonia protocol specification is given in Section 4, followed by descriptions of our implementation and performance evaluation results in Section 5. Related work is discussed in Section 6 and Section 7 concludes the paper.

2 AN FRAMEWORK OF ADVERSARIES

Intuitively, an e-voting protocol should hide how each individual votes. However, requiring that a voting protocol leaks no information about how each individual votes is impossible to achieve, because the final vote outcome, which is affected by each individual's vote, must be public. This problem becomes more acute when we consider that some voters may be malicious or corrupted. In the extreme case, suppose that all voters other than Alice are colluding, then they can infer how Alice voted from the published vote outcome and their own votes. Therefore, the best one can hope to achieve is that a voting protocol reveals nothing beyond the aggregate of all votes from honest voters who follow the protocol. This is formalized by requiring that the adversary cannot distinguish between two worlds such that the only differences between them are that honest voters vote differently, subject to the constraint that the aggregates of the honest voters are the same in both worlds.

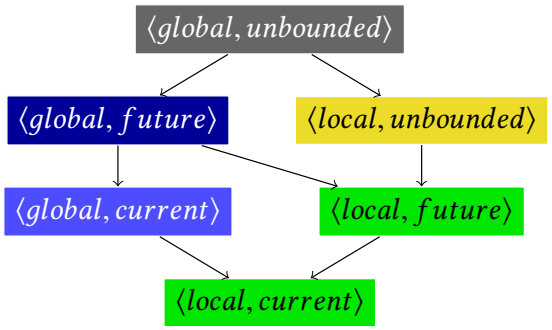

Figure 1: Possible adversary models in terms of message observation capabilities and computational power

While e-voting privacy has been studied quite extensively, one aspect that has not been systematically analyzed is the capabilities of the adversary. There are two dimensions. The first is what computational capabilities the adversary has, along which we identify three classes of adversaries. The **current** adversary is bounded by today's computational power. That is, we can assume that today's widely used public-key cryptography primitives, which are based on factoring and discrete log, are secure. The next level-up **future** adversaries may have access to quantum computers and/or better algorithms for breaking these primitives. Finally, we have the computationally **unbounded** adversary, which models the situation where even today's quantum-safe crypto algorithms turn out to be broken in the future.

The second dimension is what we assume the adversary can observe. We categorize this into two levels. A **local** adversary has access to all views of the parties involved in the election, except for the honest parties who will follow the protocol and not share additional information with the adversary. A **global** adversary in addition has access to all (possibly encrypted) communications between all honest parties as well. Such an adversary is not unrealistic. Some government agencies aim to monitor and store as much internet communications as possible. It is also easy to monitor a small number of targeted users, since most likely the communication channels are wireless. Figure 1 shows the lattice of adversary models combining computational and communication access capabilities, ordered by strength.

Ideally, we want to achieve voting privacy against the strongest possible (namely, ⟨*global*, *unbounded*⟩) adversary; however, doing so is very difficult, if not impossible in a

[2] https://github.com/gehuangyi20/Koinonia.

practicable setting. The reason is similar to why a commitment that is both perfect hiding and perfect binding is impossible in the normal setting [30]. A global adversary can observe all traffic from a voter. When the adversary is also unbounded, it can also decrypt all communications. (An exception is when an information-theoretically secure encryption scheme such as one-time pad is used, which is impractical in e-voting.) Thus the adversary sees all information sent by a voter (which can be intended for multiple receivers). In order for the election to be able to publish a result and produce a proof of the correctness of the result, it must be possible to interpret what the voter intends to vote from the combined information sent by the voter. In that case, an unbounded adversary should be able to discover what the voter votes as well.

We aim at achieving voting privacy against slightly weaker adversaries. First, consider the $\langle global, future \rangle$ adversary, which is assumed to be able to break number-theory based public-key crypto, but not quantum-safe crypto. By applying quantum-safe crypto to protect the communications between the parties, one renders this adversary equivalent to a local adversary.

The more interesting case is a $\langle local, unbounded \rangle$ adversary. We believe that privacy in this setting is important. Such an adversary controls some parties in the voting process and can see all information made available in the public channel (such as a public bulletin board) and stores this information. When in the future quantum computers (or other ways to break crypto) appear, the adversary can try to find out retrospectively how the voters voted. There are two common ways of achieving voter privacy. One approach is to use encryption, where one's vote is encrypted, and the ciphertext is sent to the Server. Either the encryption scheme is homomorphic and can be decrypted in a threshold setting, or mixing by multiple authorities is used to break the connection of an encrypted vote and its voter. The other approach is to use secret sharing, where one's vote is split into shares and sent to a number of authorities. Either way, one has to trust at least one authority is honest for privacy.

When we want to defend against a $\langle local, unbounded \rangle$ adversary, it makes sense to use secret sharing instead of encryption, for protecting voter privacy. In this setting, when one's vote is encrypted, then any party who sees the ciphertext could potentially compromise the privacy in the future (when crypto is broken). Therefore, instead of trusting *at least one Authority is non-malicious*, one effectively must trust *all parties who have access to ciphertexts are non-malicious*. On the other hand, secret sharing schemes can be information theoretic hiding, which is well suited for protecting privacy in the face of a $\langle local, unbounded \rangle$ adversary. While one can also secret-share encrypted votes to prevent any single party from obtaining the full ciphertext, the use of secret sharing renders encryption unnecessary.

3 OVERVIEW OF KEY IDEAS

In this section, we present the high-level ideas behind Koinonia, the e-voting software we designed and implemented, starting with the parties involved in secret-sharing-based voting schemes.

3.1 Sharing-based Voting Schemes

We consider the following parties:

- **Election Service Provider (ESP).** The ESP runs a website in support of e-voting and provides a **Bulletin Board**, which acts as a reliable broadcast channel.
- **Election Authority (EA).** Each election has one EA, which is in charge of maintaining a list of registered voters and publishing information specific to that election.
- **Tabulation Tellers.** There are $t > 1$ tallying tellers, or tellers, $T^1, T^2, \cdots, T^t$. They are involved in the voting and tallying process. Voting privacy relies on the assumption that at least one teller is honest and follows the protocol. (A teller here is equivalent to what was referred to as *Authority* in Section 1.)
- **Voters.** There are $n > 1$ voters, denoted as V_i where $1 \leq i \leq n$. We overload V_i to also denote the id of the i'th voter. We use subscript for voters and superscript for tellers.

ESP provides the software and platform for e-voting. One ESP may be used to support many elections, each of which may have its own EA.

In Koinonia, each vote is a vector of non-negative integers $\mathbf{v}_i = \langle v_{i,1}, v_{i,2}, \ldots, v_{i,c} \rangle$, where c is determined by the election, e.g., the number of candidates in the election. In most elections, each $v_{i,j}$ is one bit encoding a yes/no answer for one question; for example, whether to support a candidate for a position. Oftentimes, in an election, not all integer vectors are valid selections. For example, there may be three candidates, and each voter can vote for only one candidate. We use the predicate valid to denote the condition for valid votes; that is, $\mathsf{valid}(\mathbf{v}_i) = \top$ if and only if $\mathbf{v}_i$ is a valid selection vector for the election.

A voting scheme is specified by the following algorithms $(\mathsf{Setup}, \mathsf{Vote}, \mathsf{Verify}_\beta, \mathsf{Verify}_\rho, \mathsf{Verify}_\mathbf{r}, \mathsf{Aggregate}, \mathsf{TallyVerify})$. The three verification algorithms $\mathsf{Verify}_\beta, \mathsf{Verify}_\rho, \mathsf{Verify}_\mathbf{r}$ all output either $\top$ (indicating success) or $\perp$ (or failure). An election involves the following interactions:

- ESP runs an algorithm $\mathsf{Setup}(\lambda)$, which on input a security parameter λ outputs public system parameters Γ (including q which is a large prime number). Γ is then published.
- EA publishes election information E, which specifies the voting options such as the positions (e.g., president and

secretary) and candidates per position, the list of tabulation tellers (their online addresses and public keys), as well as a list of voter ids that are legitimate voters.
Γ and E are implicit inputs to all algorithms below.

- Voter V_i, who wants to cast a vote $\mathbf{v}_i$, runs an algorithm $\text{Vote}(V_i, \mathbf{v}_i)$, which outputs a vector $\langle \beta_i, \rho_i^1, \rho_i^2, \cdots, \rho_i^t \rangle$. The voter sends β_i, which we call the **ballot**, to ESP. Ballots contain auxiliary commitments of the vote used to achieve correctness and robustness guarantees. Voter V_i also sends each ρ_i^k (which we call a **vote share**) to the teller T^k.
- Each teller T^k runs $\text{Verify}_\rho(\rho_i^k)$ on each received ρ_i^k, and accepts it if and only if $\text{Verify}_\rho(\rho_i^k) = \top$.
- ESP runs Verify_β on each received ballot β_i, and accepts β_i if and only if $\text{Verify}_\beta(\beta_i) = \top$.
- ESP publishes and signs all accepted ballots, which we use $\mathbf{B}$ to denote.
- Let R^k denote all ballot shares received by the k'th teller T^k. Each T^k runs $\text{Aggregate}(\mathbf{B}, R^k)$, which outputs $\mathbf{r}^k$. T^k sends $\mathbf{r}^k$ to the ESP.
- On each $\mathbf{r}^k$, ESP runs $\text{Verify}_\mathbf{r}(\mathbf{B}, \mathbf{r}^k)$ and accepts it if it is valid. After ESP has accepted $\mathbf{r}^k$'s from all tellers, ESP publishes $\langle \mathbf{r}^1, \mathbf{r}^2, \cdots, \mathbf{r}^t \rangle$.
- Anyone can run $\text{TallyVerify}(\mathbf{B}, \langle \mathbf{r}^1, \mathbf{r}^2, \cdots, \mathbf{r}^t \rangle)$, which outputs either the vote outcome, which is a vector $\in \mathbb{N}^c$ or $\perp$, which denotes that the voting transcript is invalid.

As illustrated in Fig. 2, in voting schemes following the above convention, the interactions between the voters and the other entities, and between the tellers and the ESP are minimal. Each voter, after having the election information, computes the ballot and ballot shares without any interaction with the ESP or the tellers, and then sends these out. Each teller also computes the aggregates without interacting with the ESP and only sends the ESP results in the end. This enables proving voting privacy against the malicious adversary (instead of the honest-but-curious adversary), since the lack of interaction makes the two models equivalent in our setting. Anything a malicious adversary can hope to learn has to come from the messages computed by the honest parties without being interfered by the malicious parties.

3.2 Using Secret Sharing

In [27], Shamir's secret sharing is used for voting. Here we replace it with simple additive sharing, which serves our purpose. For simplicity, we consider a single election where each voter V_i's vote is a single bit $v_i \in \{0, 1\}$, and the final election outcome is computed by $\sum_{i=1}^n v_i$.

When voting, each voter V_i splits her vote v_i into t shares by first choosing $x_i^1, x_i^2, \cdots, x_i^{t-1}$ uniformly at random from $\mathbb{Z}_q$, and then computing

$$x_i^t = \left(v_i - x_i^1 - x_i^2 - \cdots - x_i^{t-1}\right) \mod q$$

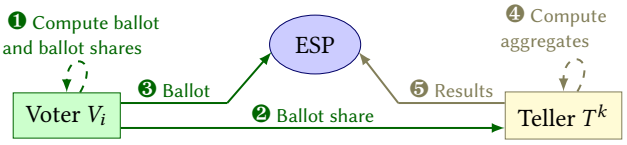

Figure 2: A high-level view of voting schemes based on secret sharing. Dashed lines show actions that entities can take without interacting with others; Solid lines show messages sent between entities. After the initial election setup, such schemes typically require only minimal interactions between entities for vote casting and tallying.

V_i then sends x_i^k, the k'th share of her vote, to T^k in a secure channel.

Each teller T^k receives a number of shares from voters. It then sends to the ESP the result of Aggregate, which is $X^k = \sum_{i=1}^n x_i^k \mod q$. The final outcome is

$$\sum_{k=1}^t X^k \mod q = \sum_{k=1}^t \sum_{i=1}^n x_i^k \mod q = \sum_{i=1}^n \sum_{k=1}^t x_i^k \mod q = \sum_{i=1}^n v_i$$

Any party can do a modulo-q sum of these numbers and recover the final outcome. After the voting finishes, each voter and teller should remove from private storage all their stored vote shares.

Note that given the voting outcome, the adversary can figure out how the set of all honest voters voted as a whole; however, assuming that at least one teller is honest, and the adversary cannot decrypt the communication between honest voters and the honest teller, even computationally unbounded adversary cannot learn any additional information.

3.3 Verifiable Teller Aggregates

Using only secret sharing does not provide any integrity/correctness guarantee. Any malicious voter, teller, or ESP can manipulate vote outcomes. If a teller is malicious, it can report an incorrect aggregate, e.g., adding or subtracting some number from the reported aggregate. To prevent this, each voter submits in the ballot cryptographic commitments of all the shares. This ballot is published on the bulletin board. When a teller publishes the aggregate, it also needs to publish evidences that the aggregate is indeed that of the shares in all published ballots.

Informally, a commitment scheme enables a party P to commit to a value x by computing and publishing $z = f(x, y)$, where y is a newly chosen random nonce. P can later "open" the commitment by revealing x and y. The hiding property of a commitment scheme requires that one cannot learn the value x from z, and the binding property requires that P cannot open the commitment using a pair of values x', y' such

that $x' \neq x$. Both the hiding and binding property can be either computational or unconditional (information theoretic). However, it is impossible to have both unconditional hiding and unconditional binding (in the normal setting [30]). For Koinonia, we use the Pedersen commitment scheme [52], which is given below.

Definition 1 (The Pedersen Commitment Scheme).

Setup A trusted third party chooses a group G of size q, a generator g of the group G, and a random element $h \in G$ such that $\log_g h$ is unknown.

Commit The domain of the committed values is $\mathbb{Z}_q$. To commit to a value $x \in \mathbb{Z}_q$, one chooses a random nonce $y \leftarrow \mathbb{Z}_q$ and computes the commitment $z = g^x h^y$.

Open To open a commitment z, one reveals x and y, and anyone can verify that $z = g^x h^y$.

The Pedersen commitment scheme is *unconditionally hiding*: Even with unlimited computational power it is impossible for an adversary to learn any information about the value x from z. This commitment scheme is *computationally binding*: Assuming that Discrete Logarithm is hard, it is computationally infeasible for an adversarial committer to open $z = g^x h^y$ using $x' \neq x$. These two properties together make the Pedersen commitment scheme well-suited for our needs of *future privacy* and *current integrity*.

The Pedersen commitment scheme has the homomorphic property. Given n commitments $z_i = g^{x_i} h^{y_i}$ for $1 \leq i \leq n$, the product of these n commitments

$$Z = \prod_{i=1}^{n} z_i = \prod_{i=1}^{n} g^{x_i} h^{y_i} = g^{\sum_i x_i} h^{\sum_i y_i}$$

is a commitment of $X = \sum_i x_i$, and can be opened using X and $Y = \sum_i y_i$.

For each vote share x_i, the voter V_i randomly chooses y_i, includes the Pedersen commitment $z_i = g^{x_i} h^{y_i}$ in her ballot β_i, and sends both x_i and y_i to the teller. When the teller reports an aggregated share $X = \sum_i x_i$, it also reports $Y = \sum_i y_i$, and the ESP can verify that the aggregate (X, Y) is correct by invoking the $\mathsf{Verify}_\mathbf{r}$ algorithm, which checks

$$\prod_i z_i = g^X h^Y$$

Since the Pedersen commitment scheme is information theoretically hiding, the commitments of the shares leak no information about the committed value; thus publishing the ballots does not affect voting privacy against the most powerful adversaries.

3.4 Proving Ballot Well-formedness

A malicious voter may try to cheat, by voting 2, 3, or more, instead of 1, for a candidate. Since each teller sees only a share of a vote, which reveals no information of the vote, it cannot detect such cheating. To prevent such malicious voter behavior, we require that each ballot have proofs that the shares are well-formed. There are several different possible definitions of well-formedness. Typically, an election requires that a vote for each candidate is either 1 or 0, that is, the proof shows that the product of the commitments of all shares for one candidate can be opened as either 1 or 0. Other constraints such as *"only one candidate among c candidates can be selected"* can also be established using proofs. This would require a proof that the product of commitments of vote shares for all candidates can be opened as a commitment of 1. The techniques for doing these two proofs are standard [26, 35, 52, 58]. We include descriptions for these two proofs in Appendix B.1 & B.2.

Exactly which kinds of proofs are needed depends on the actual requirements of a specific election, and is orthogonal to the Koinonia protocol; however, voting privacy requires that such proofs to be witness-indistinguishable [34]. A witness-indistinguishable proof (WIP) is a variant of a zero-knowledge proof. In a typical zero-knowledge proof of a statement, the prover will use a witness for the statement as input to the protocol, and the verifier will learn nothing other than the truth of the statement. In a WIP, this zero-knowledge condition is weakened, and the guarantee is that the verifier will not be able to distinguish between provers that use different witnesses. In particular, the protocol may leak information about the set of all witnesses. Unlike zero-knowledge proofs, WIP protocols remain secure when multiple proofs are being performed concurrently. In our context, a witness consists of the pair of a vote share x and and a nonce y such that $g^x h^y = z$, and our privacy goal requires that the proof does not enable one to distinguish the usage of (x, y) in the proof versus the usage of x', y' in the proof where $g^{x'} h^{y'} = z$.

In Koinonia, such well-formedness proofs are included in the ballots submitted to the ESP, and can be verified by everyone. This does not affect the everlasting privacy property of Koinonia, as the proofs used by an election are witness-indistinguishable, and reveal no information about the vote shares and the secret nonces.

3.5 Robustness

A teller may try to obstruct the tallying process by not submitting aggregates, or submitting aggregates that are incorrect. For example, a malicious party may register as a teller for the purpose of obstructing the voting process. Such behavior can be identified. If one desires to ensure that tallying can continue even in the presence of such tellers, each voter needs to store the vote shares until the tallying phase finishes. After a teller is identified as obstructing, the teller's role needs to

be replaced by a new teller or by the ESP, and having the voters send their shares to the new teller.

We note that existing e-voting protocols are also vulnerable to such a DoS attack. If homomorphic encryption is used, and collaboration of multiple parties is needed to decrypt, then one or more such parties can decide not to participate in the protocol for decrypting voting results. This problem is usually dealt with using threshold cryptography, so that not all m parties are required to decrypt. So long as the number of obstructing tellers is below a threshold, decryption can continue. This, however, also means that the collusion of fewer than m parties can compromise vote privacy. In the case of sharing a decryption key among m parties, each party possesses a piece of information that is not known to anyone else. In our setting, the information possessed by a teller is held by the voters, enabling the system to recover from an obstructing teller; thus we choose to maximize privacy assurance and uses a simple m-out-of-m secret sharing, instead of a k-out-of-m secret sharing.

A teller may try to obstruct in a more subtle manner. For example, it may report an aggregate that does not include some voters' shares, and claim to have never received the shares from the voters. This can be solved as follows. When a voter submits a vote share x, nonce y, and commitment $z = g^x h^y$ to a teller, the teller digitally signs $\langle z \rangle$ after verifying that $z = g^x h^y$. This signature is then included in the ballot. A ballot is valid only if it has signed acknowledgments for share commitments. Thus for each share, the ballot needs to include a commitment of the share as well as a signed acknowledgment from the teller who receives the share.

If a teller does not provide a signature after receiving the commitment, then the voter will immediately notice that the process of submitting her vote is obstructed by the teller. This needs to be resolved outside the voting protocol. Obtaining the digital signatures ensures that after the ballot is submitted, if a teller obstructs, then it can be identified as obstructing.

3.6 Ballot Legitimacy

Given a list of all published ballots, together with vote share aggregates from the tellers, everyone can verify that a vote outcome is consistent with the published ballots. If one defines voting correctness/integrity to mean just that, then this can be achieved without the need to trust any of the parties involved in the election. However, for the election result to be correct, we also need **ballot legitimacy**, which we define to mean that each of the published ballot is cast by an eligible voter, and every ballot cast by an eligible voter is included in the list.

Defending against the "missing ballot" attack. After the election closes, the Election Service Provider (ESP), who controls the bulletin board, publishes the list of all accepted ballots. Each teller can then find all vote shares that correspond to the commitments in the ballots, and compute their aggregates. We note that it is possible that a small number of vote shares are missing, as a voter may have started the voting process and sent vote shares to some tellers, but did not complete the voting process. However, if there are a significant number of vote shares missing from the ballot, then the teller can detect this. Also, each voter can check whether her ballot is included in the tallying. Even if not all voters actually perform the check, the fact that they can and some of them do, should be sufficient to deter a malicious ESP from intentionally not including a valid ballot. To be able to identify whether the ESP is indeed to blame (and not framed by malicious voters who submit all vote shares but not the ballot), the voting process should require the ESP to provide a signed receipt upon accepting a ballot. A voter can choose to either perform the check herself, or send the signed receipt to tellers to allow them to pursue the ESP if her ballot is not included. Disputes regarding ESP obstructing an election by not issuing a signed receipt for a valid ballot will be discovered immediately.

Defending against the "ballot stuffing" attack. Ensuring that every published ballot is cast by an eligible voter is more difficult. As entities such as ESP and EA can easily play the role of voters and submit vote shares. To ensure this, some mechanism is needed to authenticate ballots. This is obviously needed if an election is not open to everyone. Even if an election is open to everyone, some mechanism is needed to prevent a voter from voting more than once. Exactly how voter authentication is done is outside an e-voting protocol. For example, the default setting of Helios requires the EA to upload a list of allowed voters and their email addresses. Then each voter registers with the ESP, proving ownership of an email address on the list, before she can be allowed to vote. However, trusting ESP for ballot authenticity seems unsettling.

Another approach, which removes the trust on ESP, is for each voter to submit her ballot to the EA, who verifies voter identity, and then digitally signs the accepted ballot, which is then published in the bulletin board. In this approach, trust is shifted from the ESP to the EA. It might appear that trusting the EA for ballot legitimacy is unavoidable, since after all EA is the entity to decide the list of legitimate voters. However, we argue this is not the case. When publishing the list of eligible voters, cheating behavior by the EA will be detected, and thus the EA is deterred from doing that. However, if the EA signs each ballot that is not linked to any voter, this cannot be detected.

Another possibility is for the EA to publish a list of voters together with their public keys (which can be a newly generated key specific to this voting, to provide pseudonymity). Each voter then digitally signs her ballot, which is published. This requires the existence of a public key infrastructure (PKI) before an election starts.

Without a PKI, trust on ESP and EA can be shifted to tellers. When the list of eligible voter email addresses is made public, each teller can also verify ownership of email addresses and accepts vote shares only from legitimate voters. If it is required that each voter must send vote shares to all tellers, then this removes trust in a single entity (ESP), and replaces it with trust that at least one of the tellers is honest, which is the same as the trust assumption for privacy.

What if one entity does not want to trust any of ESP, EA, and tellers, for voting integrity/correctness? This is achievable under the following conditions. First, the list of eligible voters is public. Second, each ballot can be linked to one voter, which can be contacted. Third, each voter can identify one's own ballot. To verify voting correctness, one can randomly select a number of ballots in the published list, extract the voter identities of them, and contact each one to confirm that the published ballot on the list is indeed hers. Note that this does not reveal how a voter voted, because the ballot contains only commitments of shares.

4 THE PROTOCOL IN Koinonia

We now describe the protocol in Koinonia.

4.1 The Setup Phase

System Parameter Generation. The setup algorithm is executed by the ESP and takes a security parameter λ as input.

$$\mathsf{Setup}(\lambda) = \langle G, q, g, h \rangle \,, \tag{1}$$

where G is a group consisting of q elements where discrete logarithm is hard, g is a generator of G, and h is an element of the group such that no party knows $\log_g h$. In our current implementation, G, q, g are determined by choosing a standard Elliptic curve, and h is computed by hashing the string "Koinonia".

Voter Registration. The EA publishes election information and the list of eligible voters. Let n be the total number of all registered voters, the EA publishes:

$$\langle V_1, \cdots, V_n \rangle \,. \tag{2}$$

Tabulation Teller Registration. The EA publishes t Tabulation Tellers, and publishes their network addresses and public keys.

$$\left\langle T^1, \mathsf{PK}^1; \cdots; T^t, \mathsf{PK}^t \right\rangle . \tag{3}$$

4.2 The Voting Phase

As shown in Fig. 3, the voting phase involves voters, tellers, and the ESP. For simplicity we focus on the interactions as seen by a particular voter, V_i, involving only an arbitrary teller, T^k.

Commitment Generation. Let c be the number of candidates, and $v_{i,j} \in \{0, 1\}$ be V_i's vote for candidate j. Voter V_i splits each vote $v_{i,j}$ into t random shares: $x^1_{i,j}, x^2_{i,j}, \cdots, x^t_{i,j} \in \mathbb{Z}_q$ by choosing, at uniform random, $x^1_{i,j}, x^2_{i,j}, \cdots, x^{t-1}_{i,j}$ from $\mathbb{Z}_q$, and computing

$$\forall_{j \in [1..c]} \; x^t_{i,j} = \left(v_{i,j} - \sum_{k=1}^{t-1} x^k_{i,j} \right) \mod q \tag{4}$$

V_i chooses $c * t$ nonce values $y^k_{i,j}$'s at uniform random from $\mathbb{Z}_q$, and constructs her matrix of commitments as follows:

$$M_i = \begin{pmatrix} z^1_{i,1} = g^{x^1_{i,1}} h^{y^1_{i,1}} & z^2_{i,1} = g^{x^2_{i,1}} h^{y^2_{i,1}} & \cdots & z^t_{i,1} = g^{x^t_{i,1}} h^{y^t_{i,1}} \\ z^1_{i,2} = g^{x^1_{i,2}} h^{y^1_{i,2}} & z^2_{i,2} = g^{x^2_{i,2}} h^{y^2_{i,2}} & \cdots & z^t_{i,2} = g^{x^t_{i,2}} h^{y^t_{i,2}} \\ \cdots & \cdots & \cdots & \cdots \\ z^1_{i,c} = g^{x^1_{i,c}} h^{y^1_{i,c}} & z^2_{i,c} = g^{x^2_{i,c}} h^{y^2_{i,c}} & \cdots & z^t_{i,c} = g^{x^t_{i,c}} h^{y^t_{i,c}} \end{pmatrix} \tag{5}$$

In this matrix, the j'th row includes commitments for all shares of V_i's votes for candidate j, and the k'th column includes commitments of shares to be sent to teller T^k.

Sending Shares. V_i sends to T^k, via a secret channel, $\rho^k_i = \left\langle X^k_i, Y^k_i, Z^k_i \right\rangle$, where $X^k_i = \langle x^k_{i,1}, x^k_{i,2}, \cdots, x^k_{i,c} \rangle$ consists of secret shares of the votes for the c candidates, $Y^k_i = \langle y^k_{i,1}, y^k_{i,2}, \cdots, y^k_{i,c} \rangle$ are the nonces, and $Z^k_i = \langle z^k_{i,1}, z^k_{i,2}, \cdots, z^k_{i,c} \rangle$ are the commitments. Such a channel can be established because T^k's public key is published and known to all voters.

The teller T^k verifies that the commitments in Z^k_i are indeed computed using values in X^k_i and Y^k_i, i.e.,

$$\mathsf{Verify}_\rho(X^k_i, Y^k_i, Z^k_i) = \top \text{ iff } \forall_j \; z^k_{i,j} = g^{x^k_{i,j}} h^{y^k_{i,j}} \tag{6}$$

If the verification succeeds, T^k stores X^k_i, Y^k_i, Z^k_i, and sends a digital signature of Z^k_i to V_i. Let σ^k_i denote these signatures.

Constructing Well-formedness Proofs. V_i then constructs a proof Π that the commitments are well-formed. As discussed in Section 3.4, what to prove depends on the requirements of an election. For example, if the requirement is that the vote for each candidate should be either 0 or 1, and that V_i must vote for one and only one candidate, i.e., $\sum_{j=1}^{c} v_{i,j} = 1$, then Π consists of the following two kinds of proofs:

$$\begin{aligned} &\forall_{j \in [1..c]} \;\; \pi_{i,j} \\ &\text{where } \pi_{i,j} \text{ is a proof that the product of the } j\text{'th row} \\ &\text{in } M_i \text{ is a Pedersen commitment of either 0 or 1} \\ &\text{(see Appendix B.2 for details about } \pi_{i,j}) \end{aligned} \tag{7}$$

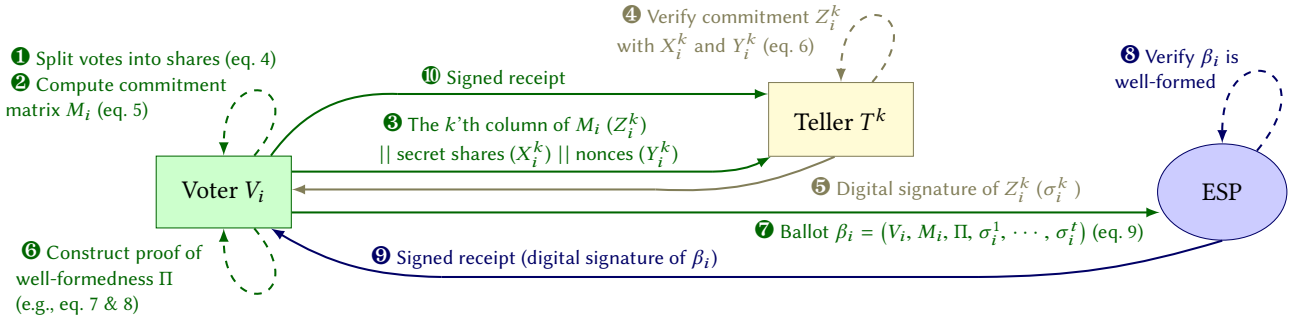

Figure 3: The Voting Phase flow and messages exchanged between Voter, Tellers and ESP in Koinonia

$$\pi_{i,0} = \sum_{j=1}^{c} \sum_{k=1}^{t} y_{i,j}^k$$

where $\pi_{i,0}$ is a proof that the product of all elements in M_i is a Pedersen commitment of 1 (see Appendix B.1 for details about $\pi_{i,0}$) (8)

Using $\pi_{i,0}$, one can verify that

$$g^1 h^{\pi_{i,0}} = \prod_{j=1}^{c} \prod_{k=1}^{t} z_{i,j}^k$$

This proves that a voter's vote for all candidates sums up to 1. Combining this with the condition that each vote for every candidate is either 0 or 1, we can prove that any vote is for one and only one candidate.

In our implementation of Koinonia, by default we assume that the election allows *null votes*, that is, a voter can choose not to support any candidates. In that case, Koinonia only needs to support the proofs described by eq. (7).

Hence V_i's ballot consists of the following:

$$\beta_i = \left(V_i, M_i, \Pi, \sigma_i^1, \cdots, \sigma_i^t\right) \tag{9}$$

Submitting Ballot. V_i submits to the ESP her ballot β_i. The ESP runs Verify_β, which verifies that β_i is **well-formed**, i.e., the voter is a valid voter, the signatures from the tellers are valid, and the proofs are valid. Then the ESP publishes via the bulletin board β_i, and also signs β_i and sends to V_i the signature as a signed receipt. V_i sends the signed receipt to all tellers.

4.3 Tallying and Verification

The tallying phase begins after the completion of the voting phase. As shown in Fig. 4, it mainly involves the ESP and tellers. Results can be fetched and verified through the bulletin board provided by the ESP.

Publishing Ballots. The ESP publishes the final list of all accepted ballots.

$$\mathbf{B} = \langle \beta_i, \quad \text{for } i \in \mathbf{I}^+ \rangle \,. \tag{10}$$

Where $\mathbf{I}^+$ denotes all i's such that Voter V_i's ballot β_i is included in the list of accepted ballots.

The Aggregate **function.** Each teller T^k first checks that any ballot for which it has received a signed receipt is included in $\mathbf{I}^+$. If not, it reports that the ESP is trying to exclude ballots and produces the signed receipt as an evidence.

Each teller T^k then computes the sum of the random shares included in β_i's for $i \in \mathbf{I}^+$

$$\left\langle \left(X_{*,j}^k = \sum_{i \in \mathbf{I}^+} x_{i,j}^k \bmod q, Y_{*,j}^k = \sum_{i \in \mathbf{I}^+} y_{i,j}^k \bmod q \right), \quad \text{for } j \in [1..c] \right\rangle. \tag{11}$$

T^k sends $(X_{*,j}^k, Y_{*,j}^k)$ and its digital signature, signed by T^k's private key, to the ESP, who accepts it after verifying that

$$\forall_j \; g^{X_{*,j}^k} h^{Y_{*,j}^k} = \prod_{i \in \mathbf{I}^+} Z_i^k \tag{12}$$

The ESP, after having accepted $(X_{*,j}^k, Y_{*,j}^k)$'s from all tellers, publishes $X_{*,j}^k, Y_{*,j}^k$ from all T^k's and digitally signs them.

The final outcome can then be easily computed

$$X_{*,j} = \sum_k X_{*,j}^k \qquad Y_{*,j} = \sum_k Y_{*,j}^k \tag{13}$$

Verification (the TallyVerify **function).** Each voter can verify that her ballot is included in the list of accepted ballots. Furthermore, even if a voter does not perform the check, so long as the voter follows the protocol and distributes the acknowledgment to all tellers, and at least one honest teller follows the protocol and checks that the ballot is included, omission of the ballot can be detected. Anyone can run the following TallyVerify procedure using the published ballots and aggregates as input.

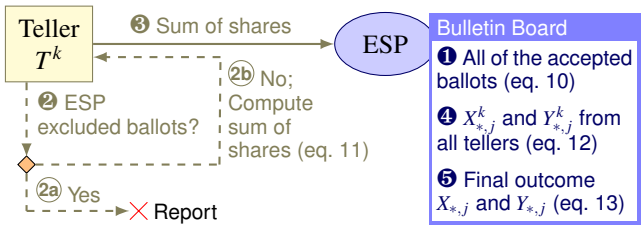

Figure 4: The Tallying Phase of Koinonia

- Check that the proof Π in each ballot β_i is correct.
- Computes the final outcome (see eq. (13) above)
- Check that the final outcome for each candidate j, $X_{*,j}$ and $Y_{*,j}$ satisfies the following condition:

$$\forall_j \quad g^{X_{*,j}} h^{Y_{*,j}} = \prod_{i \in \mathbf{I}^+} \prod_{k=1}^{t} z_{i,j}^k \tag{14}$$

- If any of the above checks fail, outputs $\perp$.

4.4 Complexity of Koinonia

We analyze computational complexity for a voter, and for the cost of verifying an election. We count the number of group exponentiations since its cost dominates other computations. Let c be the number candidates and t be the number of tellers. In the ballot generation, a voter computes ct commitments (eq. (5)), each of which requires 2 exponentiations. It also needs to generate c proofs (eq. (7)) that a committed value is either 0 or 1, each of which requires 3 exponentiations (eq. (17) in Appendix B), and one additional proof on the sum of all c votes. In our implementation, a voter can cast null votes, so we need another proof of 0 or 1, and hence $2ct + 3c + 3$ exponentiations in total. Alternatively, if the election requires exactly one candidate to be chosen, then the additional proof is a commitment of 1, which requires only 1 exponentiation (eq. (16) in Appendix B). Thus, in total, creating a ballot takes $2ct + 3c + 1$ exponentiations.

To verify a proof that a committed value is either 0 or 1 takes 4 exponentiations, and verifying a proof that a committed value is 1 takes 2 exponentiations (eq. (18) and eq. (16) in Appendix B, respectively). Hence verifying proofs in a ballot needs $4c + 4$ exponentiations if voters are allowed to cast null votes, or $4c + 2$ exponentiations if the election requires each voter to choose exactly one candidate. Note that the cost of verifying the aggregate shares published by tellers are correct, when amortized over each ballot, is small compared to that of verifying the well-formedness proofs of each ballot.

5 IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this section, we describe our implementation of Koinonia. We first show how it provides the capability of setting up elections on servers (ESP, EA, and Tellers) and handling votes from clients (Voters). We also discuss how we provide flexibly configurable secure communication channels as well as its implications on privacy.

5.1 System Components

The Koinonia system is written in `Node.js`. ESP, EA, and Tellers are independent server-side applications recommended to be running on different machines. Clients will get the Voter application from the Authority and execute it on a Web browser. We use `PostgreSQL` on ESP, EA, and Tellers as the database back-end for storing election data.

Koinonia **Libraries.** In Koinonia, ESP, EA, Tellers, and Voters share similar needs for protocol functions such as vote share generation and proof verification. We thus built two libraries, one for server-side and the other for client-side.

`WebCryptoAPI` [65] is a standard JavaScript API for performing cryptographic operations in web applications. Although `WebCryptoAPI` has already been supported by major browser vendors, at the time of writing, it has a limited number of cryptographic algorithms available, and does not natively support expensive big integer operations such as modular exponentiation. For ease of deployment, we implemented a Koinonia library for Voter that is based on SJCL [37, 62] and can run on typical web browsers, adding 500+ lines of JavaScript code.

Since cryptographic operations such as group exponentiation are computationally expensive, we implemented some of these functionally with native code as optimization for ESP, EA and Tellers, as server-side applications allow a more flexible deployment. Through `Node.js` C++ Addons, we added 1200+ lines of C++ code for wrapping around and routing many cryptographic operations to `OpenSSL` for significantly better performance. It initializes (eq.1) with an EC (Elliptic Curve) scheme, which fixes the group G and generator g, and a random EC point $h = (x, y) \in G$ generated based on the digest of a common string (x).

PKOV. Public Key Ownership Verification (PKOV) is a challenge-response protocol to show the server owning the public key under computational security. The client sends a pseudo-random message m to the server. The server computes the digest d (using SHA-256) of the concatenation of a prefix string "`PKOV`", the message m, and a 256-bit pseudo-random salt encoded in Base64. Then, the server signs the digest d with its private key. The server returns the signature, salt, and public key. The client accepts the public key if the signature verification is correct.

We run the PKOV protocol to retrieve the public key from a remote server during the election setup phase (Section 4.1). The PKOV protocol allows the server to choose asymmetric keys independent from the X.509 certificates used in HTTPS,

and allow one to experiment with new public-key crypto that is not yet widely deployed in common browsers.

ESP. The ESP is a web server providing the software and platform. One ESP may serve many EAs, who run different elections. The ESP provides a **Bulletin Board** to accept the well-formed ballot β_i and return the digital signature to the voter (see Fig. 3). The ESP is the only public channel to broadcast the accepted ballots and the election final result.

EA. The EA is a web server which allows the election administrator to control the progress of an election. The election administrator can use a browser to register an EA server on the ESP to obtain an unique election ID for the newly created election through the website hosted by the ESP. The ESP retrieves the public key of the EA using the PKOV protocol before returning the election ID. The administrator creates an election and registers teller servers that are to be used in the newly created election on the EA with the election ID. It is recommended that the administrator use only an internal network to connect to the EA for the creation/administration of elections.

In our current implementation, we use voters' email address and a 256-bit randomly generated voter ID to represent their identity. The administrator publishes the election information (election positions & candidates), the list of valid voter identities, and the list of teller servers with public keys. The EA signs all published information before the election opens. After an election is opened, the administrator sends from the EA a voting link to voters through emails. Each voting link contains the election ID, voter identity, and associated link digital signature. The digital signature in the voting link is a secret to authenticate the voter submitting the share and commitment to Tellers and the ballot to the ESP.

Tellers. Each Teller T^k is a web server that accepts and verifies (eq. 6) a vote share (X_i^k, Y_i^k) and the associated commitment (Z_i^k, represented as an EC point) submitted by a voter V_i. The teller signs Z_i^k with its private key and sends the digital signature σ_i^k back to the voter V_i. X_i^k, Y_i^k are encrypted using *AES-CBC* before storing in the database. The encryption key is derived using *PBKDF2* (Password-Based Key Derivation Function 2) [42] from a startup password chosen by the server administrator. We recommend the Teller server administrator to be different from the election administrator, the EA server administrator, and the ESP server administrator.

In order for a Teller to serve an election, it has to be registered by the election administrator at the EA before the election opens. The registration sends the election information to the Teller server and retrieves the public key of the Teller server by running the PKOV protocol. The Teller server should retrieve the public key of the ESP and the EA by running the PKOV protocol before returning its public key to the EA.

Client. The voter uses the link received from the email to access the voting page provided by the ESP. The voter selects a candidate for each position and submits the vote.

On the client side (voter), the in-browser JavaScript generates the shares (eq. 4) and associated commitments (eq. 5) for each Teller and sends the vote shares and commitments to the Tellers via HTTP(S) requests. After receiving and verifying the signatures from all Tellers, the in-browser script constructs proof of well-formedness Π (eq. 7) and send the ballot β_i (eq. 9) to the ESP. Note that NULL vote is accepted so that $\pi_{i,0}$ becomes a proof that a value of a commitment of either 0 or 1. The in-browser script verifies signed receipt from the ESP and sends the receipt to one of the Teller servers.

The in-browser script can be hosted either by a trusted third party or by the ESP. The voter can run an offline script to verify the ballot after the election closes.

System Parameters and Tuning. Our Koinonia implementation provides configuration flexibility, by allowing the users to select appropriate security parameters to match their specific concerns and security requirements. The system is pre-configured with a default setting that is considered to be reasonable under the computational power possessed by future adversaries.

The ESP server administrator chooses the parameter λ to initialize the Koinonia Library (eq. 1). We choose curve secp256r1 [55], digest algorithm SHA-256, and "Koinonia" as the common string in the experiment. The server administrator (ESP, EA, and Tellers) can use `OpenSSL` to generate a private key which is recommended to be encrypted under a key derived from a chosen server startup password. In the experiment, we use EC key pairs with curve

Secure Communication and Long-term Privacy. Although the commitment scheme used in Koinonia is designed to be information theoretically hiding, however, if the communications between voters and the ESP, EA, and Tellers is being eavesdropped by a powerful $\langle global, future \rangle$ adversary, privacy can still be endangered. Thus, we use `stunnel` to provide secure communication channels between components. Stunnel provides a means of incorporating experimental algorithms that are not standardized into typical web browsers yet.

Specifically, with concerns about the possibility of a passive adversary monitoring enough of the current traffic and try to break them in the future with advanced algorithms or even quantum computers, we have incorporated Open

Table 1: running time of Koinonia library based on an election with one position (e.g. president), two candidates, and three Tellers

Koinonia Library based on SJCL (sample size 30)		
Voter	1. Generate Matrix M_i (eq. 5)	$146ms \pm 8.1\%$
	2. Generate Proof Π (eq. 7)	$109ms \pm 3.4\%$
	3. Verify the Signature σ_i^k from the Teller T^k	$11.9ms \pm 30\%$
	4. Verify the Signature from the ESP	$9.31ms \pm 10\%$
	5. Total Time (1+2+3×t (Number of Tellers) +4)	$301ms \pm 4.9\%$
Koinonia Library based on Node.js C++ Addons (sample size 10,000)		
Teller	1. Verify Commitments Z_i^k (eq. 6)	$1.01ms \pm 35\%$
	2. Total Time for Accepting Z_i^k (1 + Encrypt X_i^k, Y_i^k + Sign Z_i^k + Verify the Signature of the Voter Identity)	$2.37ms \pm 22\%$
ESP	1. Verify the Proof Π in the ballot β_i	$2.78ms \pm 26\%$
	2. Total Time for Accepting β_i (1 + $\forall k$, Verify σ_i^k + Sign β_i + Verify the Signature of the Voter Identity)	$5.77ms \pm 27\%$
Verifier	1. Verify Eq. 14 and the Proof Π in 10,000 Ballots (run in 8 Threads)	$6.7s$
	2. Total Time for the Verification (run in 8 Threads) (1 + $\forall i, \forall k$, Verify σ_i^k + $\forall i$, Verify the Signature of the Voter Identity)	$11s$

Quantum Safe (OQS) [3] with `stunnel` to offer the possibility of performing quantum-safe key exchange for the establishment of TLS connections. Other known key exchange algorithms, as supported by `OpenSSL`, can also be used. Like other system parameters, the exact ciphersuite to be used for the TLS connections can be configured by the users to address their specific concerns. Users are free to choose classical key exchange algorithms in favor of their maturity over the new quantum-safe ones. An evaluation of the merits between different algorithms is beyond the scope of this paper.

Since TLS is meant to provide end-to-end security, for usability sake, we have implemented a client-side provisioning proxy for setting up the various TLS tunnels automatically and multiplexing HTTP requests to the ESP and Tellers through the correct tunnel.

5.2 Election Tallying and Verification

Share Publishing. After Admin closes the election, a list of valid ballots are published on the ESP, the bulletin board. On each Teller T^k, the server administrator runs an offline script to discover the shares stored on the teller server by using the 256-bit voter ID and the teller signature σ_i^k. The script decrypts the X_i^k, Y_i^k from the filtered out shares and computes (eq. 11). The script also checks that any ballot for which a signed receipt has been received is included in the published ballots. The teller deletes the share after finishing the summation. After that, the script submits the summation result $X_{;j}^k, Y_{;j}^k$ and its digital signature, signed by T^k's private key, to the ESP.

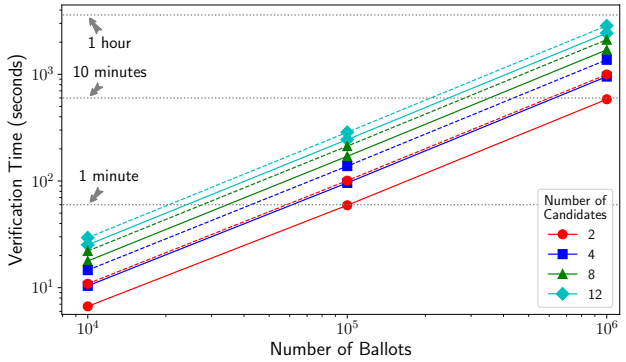

Figure 5: The Running Time of Koinonia Verification With One Position (e.g. President) and Three Tellers. Solid line shows the running time of verifying the proof Π (Esq. 7 and (eq. 14). Dashed line shows the running time taken by the solid line add with verification time of the signature of the voter Identity and σ_i^k signed by Teller T^k. X, Y axes are represented in log scale.

Final Publishing. The ESP server administrator runs an offline script which first verifies the signature of the summation results $X_{;j}^k, Y_{;j}^k$ and checks their correctness (eq. 12). The script reports Tellers which try to submit incorrect results and produces the signed result as evidence.

The script stores the final outcome $X_{;j} = \sum_k X_{;j}^k$ and $Y_{;j} = \sum_k Y_{;j}^k$ in the database, which will be immediately published through the ESP website.

Verification. The verification program is implemented according to Section 4.3. It is an offline script which can be run by any third parties (e.g. voter or teller). The verification program also provides an optional verification of the signature σ_i^k of the commitment Z_i^k (signed by Teller T^k) and the voter identity (signed by EA) for each ballot.

5.3 Performance Evaluation

We present the running time of Koinonia Library in Table 1 and show that the computation cost of our implementation is small, and the system can be deployed on commodity machine [4]. In terms of storage cost, for an election with one position (e.g. president), two candidates, and three Tellers, the size of each ballot is roughly 2.6KB (kilobyte). This increases linearly depending on the number of candidates and

[3] https://github.com/open-quantum-safe/liboqs

[4] The experiment runs on the machine with Intel Core i7-3770 3.40 GHz CPU and 16 GB RAM, running `Node.js` v8.4.0, `OpenSSL` 1.1.0f, and `PostgreSQL` v9.6.4.

the number of Teller servers. Regarding total execution time, the voter needs around 300 milliseconds to compute. Both Teller and ESP servers can verify the voter's submission within about 5 milliseconds. It takes around 11 seconds to verify the correctness of an election with 10,000 ballots (12 group exponentiations per ballot with 2 candidates).

We also measure the running time of the verification program using simulation, with elections having one position and three Tellers, 2 to 12 candidates, and 10,000 to 1 million ballots (see Fig. 5). The verification time increases linearly with the number of candidates and voters. Notice that the verification of ballots is easily parallelizable. The verification program running in 8 threads can verify the correctness of an election given 1 million ballots, each containing 12 candidates, within 1 hour. The optional verification of signature σ_i^k (dashed line) adds a small overhead, depending on the number of Tellers, on the original verification process (solid line).

6 RELATED WORK

In recent years, Internet-based voting has been utilized in governmental elections around the world such as Estonia [23, 24], New South Wales, Australia [13], Norway [38], Switzerland [15], and Utah, USA [8]. Unfortunately, due to the strict and conflicting requirements in voting [20], none of the existing voting systems adequately addressed all the challenges in e-voting [9, 36, 61]. Most of these systems focus primarily on integrity and have not considered long-term privacy as their main concerns.

E-voting protocols typically utilize the following cryptographic techniques: mix-net [1, 3, 14, 16, 18, 19, 22, 29, 41, 45, 56, 57, 63, 66], homomorphic encryption [2, 4, 25, 27, 28, 40, 43, 47], blind signature [12, 17, 44, 51, 54] and secret sharing based on polynomials [6, 7, 27, 59] or simple (n, n) secret sharing [68]. Despite numerous previous work on everlasting voting privacy [10, 11, 27, 32, 33, 46–50, 64], we note that the adversary models under which these schemes can achieve privacy could be different. For example, some systems require complex and contrasting assumptions in order to deliver privacy protections [5, 29]. Other attempts at achieving everlasting privacy include [64] which combines ideas from Punchscan [53] with Prêt à Voter [56], [32] which combines Prêt à Voter [56] with unconditional hiding commitments, and [46] which relies on membership proofs. Moran *et al.* proposed an everlastingly private voting scheme [48] for direct-recording electronic (DRE) voting systems, as well as split-ballot voting [49, 50] for a conventional "voting booth" setup.

Demirel *et al.* aimed to improve Helios [1] with everlasting privacy [33], where each individual vote is encrypted with the Paillier cryptosystem, and the tally correctness is guaranteed by Pedersen commitments. However, under the $\langle global, future \rangle$ and $\langle local, unbounded \rangle$ adversary models, this design suffers from the same problem of today's encrypted information stored by the adversary might be used to reconstruct plain votes in the future (*e.g.*, when the decisional composite residuosity assumption that the Paillier cryptosystem relies on no longer holds).

Some protocols try to incorporate quantum-safe crypto in e-voting. One such effort is EVOLVE [31]. EVOLVE shares some similarities with Koinonia as it also uses a simple (n, n)-secret sharing. However, EVOLVE relies on a lattice-based homomorphic commitment scheme. Different from Koinonia which uses Pedersen commitment and is information-theoretically hiding, the lattice-based commitment in EVOLVE, which is based on the (assumption of the) hardness of M-SIS and M-LWE, is both computationally hiding and computationally binding [31], and hence the privacy guarantee is not as strong as that of Koinonia, especially in the face of $\langle global, future \rangle$ and $\langle local, unbounded \rangle$ adversaries.

The protocol closest in spirit to Koinonia is that by Cramer et al. [27]. It uses Pedersen's Verifiable Secret Sharing scheme [52], which combines Pedersen commitment with Shamir's secret sharing scheme [60]. Instead of using Shamir's secret sharing scheme, we show that a simple additive secret sharing scheme, which is easier to comprehend and implement, is sufficient for our privacy needs. We also discuss the security considerations regarding communications among the entities in different adversary models.

7 CONCLUSION

In this paper, we consider the problem achieving strong long-term privacy guarantees in e-voting protocols. We classify adversaries along two dimensions: local and global in terms of what the adversary sees, and current, future, and unbounded in terms of their computational ability. We argue that voting protocols based on secret sharing offer a more natural and elegant privacy-preserving solution than their encryption-based counterparts. Motivated by the "current integrity; future privacy" principle, we analyze the threats posed by future adversaries to voting privacy today, and design and implement Koinonia, which combines a simple but information theoretic hiding additive secret sharing scheme with Perdersen commitments to achieve long-term privacy and correctness guarantees. We also experimented ways of using quantum-safe key exchange algorithms in protecting the communication channels used by the entities in the protocol. Koinonia requires only modest computational costs to operate. We consider handling other issues of e-voting like voter coercion and vote selling as possible future work.

Acknowledgement

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under the ASED program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA.

REFERENCES

[1] B. Adida. 2008. Helios: Web-based Open-audit Voting. In *USENIX Security Symposium*, Vol. 17. USENIX Association, 335–348.

[2] B. Adida, O. De Marneffe, O. Pereira, and J. Quisquater. 2009. Electing a University President Using Open-audit Voting: Analysis of Real-world Use of Helios. In *Proceedings of the Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE)*.

[3] R. Araújo, A. Barki, S. Brunet, and J. Traoré. 2016. Remote Electronic Voting can be Efficient, Verifiable and Coercion-Resistant. (2016), 224–232.

[4] A. Azougaghe, M. Hedabou, and M. Belkasmi. 2015. An electronic voting system based on homomorphic encryption and prime numbers. In *11th International Conference on Information Assurance and Security (IAS)*. 140–145.

[5] Susan Bell, Josh Benaloh, Michael D. Byrne, Dana Debeauvoir, Bryce Eakin, Philip Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. 2013. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13), in USENIX Journal of Election Technology and Systems (JETS), Volume 1, Number 1*. USENIX Association, Washington, D.C., 18–39. https://www.usenix.org/conference/evtwote13/workshop-program/presentation/bell

[6] J. Benaloh. 1987. *Verifiable Secret Ballot Elections*. Ph.D. Dissertation. Yale University.

[7] J. Benaloh and D. Tuinstra. 1994. Receipt-free secret-ballot elections. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*. ACM, New York, 544–553.

[8] M. Bernhard. 2016. What happened in the Utah GOP caucus. *https://mbernhard.com/Utahvoting.pdf* (2016).

[9] Matthew Bernhard, Josh Benaloh, J. Alex Halderman, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, Poorvi L. Vora, and Dan S. Wallach. 2017. Public Evidence from Secret Ballots. *Second International Joint Conference on Electronic Voting, 24 âĂŞ 27 October 2017, Bregenz, Austria* (2017).

[10] J. Bos. 1992. Practical Privacy. *PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, Available online: http://alexandria.tue.nl/extra3/proefschrift/PRF8A/9201032.pdf* (1992).

[11] J. Bos and G Purdy. 1988. A voting scheme. *CRYPTO 88, Rump session, Does not appear in proceedings* (1988).

[12] C. Boyd. 1990. A new multiple key cipher and an improved voting scheme. In *EUROCRYPT*. 617–625.

[13] I. Brightwell, J. Cucurull, D. Galindo, and S. Guasch. 2015. An overview of the iVote 2015 voting system. *Technical report, Aug. 2015. https://www.elections.nsw.gov.au/__data/assets/pdf_file/0019/204058/An_overview_of_the_iVote_2015_voting_system_v4.pdf* (2015).

[14] C. Burton, C. Culnane, and S. Schneider. 2016. vVote: Verifiable electronic voting in practice. *IEEE Security & Privacy* 14, 4 (2016), 64–73.

[15] S. F. Chancellery. 2013. Ordinance of 13 december 2013 on electronic voting (veles). *https://www.admin.ch/opc/en/classified-compilation/20132343/index.html* (2013).

[16] D. Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Comm. of the ACM* 24, 2 (Feb. 1981), 84–90.

[17] D. Chaum. 1988. Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA. In *EUROCRYPT'88*. 177–182.

[18] D. Chaum. 2004. Secret-ballot receipts: True voter-verifiable elections. *Security Privacy, IEEE* 2, 1 (Jan.- Feb. 2004), 38 – 47.

[19] D. Chaum, P. Ryan, and S. Schneider. 2005. A practical voter-verifiable election scheme. In *Proc. of the 10th European Conf. on Research in Comp. Security (ESORICS'05)*. Milan, Italy, 118–139.

[20] B. Chevallier-mames, P. A. Fouque, D. Pointcheval, J. Stern, and J. Traoré. 2010. On some incompatible properties of voting schemes. (book chapter). In *Towards trustworthy elections, http://dl.acm.org/citation.cfm?id=2167913.2167924*. 191–199.

[21] B. Chevallier-Mames, P. A. Fouque, D. Pointcheval, J. Stern, and J. Traoré. 2010. Towards Trustworthy Elections. Chapter on some incompatible properties of voting schemes, 191–199.

[22] M. Clarkson, S. Chong, and A. Myers. 2008. Civitas: Toward a secure voting system. In *Proc. of IEEE S & P*. 354–368.

[23] Estonian National Election Commission. 2005. E-Voting System Overview. *Technical report, National Election Commission, Tallinn, Estonia* (2005).

[24] Estonian Internet Voting Committee. 2014. Statistics about Internet voting in Estonia. *http://www.vvk.ee/voting-methods-in-estonia/engindex/statistics* (2014).

[25] V. Cortier and B. Smyth. 2013. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security* 21, 1 (2013), 89–148.

[26] R. Cramer, I. Damgård, and B. Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO*. 174–187.

[27] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. 1996. Multi-Authority Secret-Ballot Elections with Linear Work. In *Advances in Cryptology – EUROCRYPT'96*. Vol. 1070. 72–83.

[28] R. Cramer, R. Gennaro, and B. Schoenmakers. 1997. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*. 103–118.

[29] C. Culnane, P. YA Ryan, S. Schneider, and V. Teague. 2015. vVote: a verifiable voting system. *ACM Transactions on Information and System Security (TISSEC)* 18, 1 (2015), 3.

[30] Ivan Damgård. 1999. Commitment Schemes and Zero-Knowledge Protocols. In *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*. Springer-Verlag, London, UK, UK, 63–86. http://dl.acm.org/citation.cfm?id=647423.760339

[31] Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. 2017. Practical Quantum-Safe Voting from Lattices. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1565–1581. https://doi.org/10.1145/3133956.3134101

[32] D. Demirel, M. Henning, J. van de Graaf, P. YA Ryan, and J. Buchmann. 2013. Prêt à Voter Providing Everlasting Privacy. In *International Conference on E-Voting and Identity*. Springer, 156–175.

[33] D. Demirel, J. Van De Graaf, and R. Araújo. 2012. Improving Helios with Everlasting Privacy Towards the Public. In *Proceedings of the International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE'12)*.

[34] U. Feige and A. Shamir. 1990. Witness Indistinguishable and Witness Hiding Protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing (STOC '90)*. ACM, New York, NY, USA, 416–426. https://doi.org/10.1145/100216.100272

[35] A. Fiat and A. Shamir. 1987. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Crypt.-Crypto '86*. Springer-Verlag, New York, 186–194.

[36] L. Fouard, M. Duclos, and P. Lafourcade. 2007. Survey on Electronic Voting Schemes. http://www-verimag.imag.fr/ duclos/paper/e-vote.pdf.

(2007).
[37] Github. 2009. Stanford Javascript Crypto Library. (2009). https://github.com/bitwiseshiftleft/sjcl [Online; accessed 8-May.-2017].
[38] K. GjÃÿsteen. 2011. The Norwegian Internet voting protocol. *In 3rd International Conference on E-Voting and Identity, VoteID âĂŹ11* (2011).
[39] G. S. Grewal, M. D. Ryan, S. Bursuc, and P. Y. Ryan. 2013. Caveat Coercitor: coercion-evidence in electronic voting. In *IEEE S & P*.
[40] G. S. Grewal, M. D. Ryan, L. Chen, and M. R. Clarkson. 2015. Du-Vote: Remote Electronic Voting with Untrusted Computers. In *IEEE 28th Computer Security Foundations Symposium*. 155–169.
[41] A. Juels, D. Catalano, and M. Jakobsson. 2005. Coercion-resistant electronic elections. In *Proc. of ACM WPES*. ACM, 61–70.
[42] B. Kaliski. 2000. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898. RFC Editor. http://www.rfc-editor.org/rfc/rfc2898.txt http://www.rfc-editor.org/rfc/rfc2898.txt.
[43] A. Kiayias, T. Zacharias, and B. Zhang. 2015. DEMOS-2: Scalable E2E Verifiable Elections Without Random Oracles. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, 352–363.
[44] K. Kim, J. Kim, B. Lee, and G. Ahn. 2001. Experimental Design of Worldwide Internet Voting System Using PKI. In *Proc. of SSGRR*.
[45] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. 2003. Providing Receipt-freeness in Mixnet-based Voting Protocols. In *Proc. of Info. Security and Cryptology*, Vol. 2971. 245–258.
[46] P. Locher and R. Haenni. 2015. *Verifiable Internet Elections with Everlasting Privacy and Minimal Trust*. Springer International Publishing, 74–91.
[47] P. Locher, R. Haenni, and R. E Koenig. 2016. Coercion-Resistant Internet Voting with Everlasting Privacy. In *International Conference on Financial Cryptography and Data Security*. Springer, 161–175.
[48] T. Moran and M. Naor. 2006. Receipt-free Universally-verifiable Voting with Everlasting Privacy. In *CRYPTO*. 373–392.
[49] T. Moran and M. Naor. 2007. Split-ballot Voting: Everlasting Privacy with Distributed Trust. In *ACM CCS*. 246–255.
[50] T. Moran and M. Naor. 2010. Split-ballot voting: Everlasting privacy with distributed trust. In *ACM Trans. Inf. Syst. Security*, Vol. 13. 16:1–16:43.
[51] M. Ohkubo, F. Miura, M. Abe, A. Fujioka, and T. Okamoto. 1999. An Improvement on a Practical Secret Voting Scheme. In *Proc. of 2nd Int. Workshop on Info. Security (ISW '99)*. 225–234.
[52] T. Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO'91 (Lecture Notes in Computer Science)*, Vol. 576. Springer Berlin / Heidelberg, 129–140.
[53] S. Popoveniuc and B. Hosp. 2006. An introduction to Punchscan. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*. 28–30.
[54] M. Radwin. 1995. An Untraceable, Universally Verifiable Voting scheme. *Seminar in Cryptology* (1995).
[55] Certicom Research. 2010. SEC 2. Standards for Efficient Cryptography Group: Recommended Elliptic Curve Domain Parameters. (2010). Version 2.0.
[56] P. Y.A. Ryan and T. Peacock. 2005. Prêt à Voter: A Systems Perspective. *University of Newcastle, Technical Report CS-TR-929* (2005).
[57] K. Sako and J. Kilian. 1995. Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT'95)*. Springer-Verlag, Berlin, Heidelberg, 393–403. http://dl.acm.org/citation.cfm?id=1755009.1755052
[58] C.-P. Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology – CRYPTO'89 Proceedings*, G. Brassard (Ed.). Lecture Notes in Computer Science, Vol. 435. Springer Berlin / Heidelberg, 239–252.
[59] B. Schoenmakers. 1999. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *CRYPTO*. 148–164.
[60] A. Shamir. 1979. How to share a secret. *Comm. of the ACM* 22, 11 (Nov 1979), 612–613.
[61] Eugene H. Spafford. 2012. Election and Voting. *Computer & Security* 31 (2012), 857 –858.
[62] E. Stark, M. Hamburg, and D. Boneh. 2009. Symmetric cryptography in javascript. In *Annual Computer Security Applications Conference (ACSAC'09)*. IEEE, 373–381.
[63] G. Tsoukalas, K. Papadimitriou, P. Louridas, and P. Tsanakas. 2013. From Helios to Zeus. In *EVT/WOTE*, Vol. 1. USENIX.
[64] J. van de Graaf. 2009. Voting With Unconditional Privacy by Merging Prêt à Voter and PunchScan. *IEEE Transactions on Information Forensics and Security* 4, 4 (Dec 2009), 674–684.
[65] W3C. 2017. Web Cryptography API. (2017). https://www.w3.org/TR/WebCryptoAPI/ [REC-WebCryptoAPI-20170126].
[66] S. Weber. 2006. A Coercion-resistant Cryptographic Voting Protocol – Evaluation and Prototype Implementation. *Master's thesis, Darmstadt University of Technology* (2006).
[67] Wikipedia. 2016. iElectronic voting by country. (2016). https://en.wikipedia.org/wiki/Electronic_voting_by_country [Online; accessed 21-Jun.-2016].
[68] X. Zou, H. Li, Y. Sui, W. Peng, and F. Li. 2014. Assurable, Transparent, and Mutual Restraining E-voting Involving Multiple Conflicting Parties. In *Proc. of IEEE INFOCOM*. 136–144.

A APPENDIX

B WITNESS INDISTINGUISHABLE PROOFS OF BALLOT WELL-FORMEDNESS

B.1 A Commitment of 1

To prove that a value z is a commitment of 1, i.e., one knows y such that $z = g^1 h^y$, one needs to prove the knowledge of the discrete log of $g^{-1}z$. This is a standard protocol, which is obtained by applying the Fiat-Shamir heuristic to the Schnorr protocol of knowledge of discrete logarithm.

To prove that one knows y such that $z = g^1 h^y$, let H be a one-way hash function, one randomly chooses $r \leftarrow_r \mathbb{Z}_q$, computes $d = h^r$, $e = H(z, d)$, $f = ((r - ey) \bmod q)$ and publishes (d, f). We write this as publishing

$$(d, f) : r \leftarrow_r \mathbb{Z}_q,\ d \leftarrow h^r,\ e \leftarrow H(z, d),\ f \leftarrow ((r - ey) \mod q) \tag{15}$$

To verify, one computes $e = H(z, d)$, and checks

$$d = h^f \left(zg^{-1}\right)^e \tag{16}$$

In the proof, the value e is the challenge. When running in the interactive setting, the value e is generated by the verifier. The protocol does not reveal the value y because if the challenge e is generated first, then one can choose a random f and then compute d that will satisfy the check (16), without knowing the value y. When the prover follows the

protocol, we have

$$h^f \left(zg^{-1}\right)^e = h^{f+ey} = h^{((r-ey) \bmod q)+ye} = h^r = d$$

Why this proves knowing y. The value e is an unpredictable challenge, since it is the result of a one-way hash function. Suppose that after one decides on d, one is able to come up with f_1 and f_2 such that

$$h^{f_1} \left(zg^{-1}\right)^{e_1} = d = h^{f_2} \left(zg^{-1}\right)^{e_2}$$

Then we know that

$$h^{(f_1-f_2)(e_2-e_1)^{-1} \bmod q} = zg^{-1}$$

That is, one recovers $y = (f_1 - f_2)(e_1 - e_2)^{-1} \bmod q$ such that $z = g^1 h^y$.

B.2 A Commitment of either 0 or 1

To prove that a value z is a commitment of either 0 or 1, we need to prove that one knows y such that $g^{-b}z = h^y$, without revealing y. We now give a non-interactive version of the proof, which was given in [27]. Let $\bar{b}$ denotes $1 - b$. The basic idea is that one conducts in parallel two proofs, for knowing the discrete log of $g^{-b}z$ and $g^{-\bar{b}}z$. Since one knows only that of $g^{-b}z$, one has to cheat for the proof for $g^{-\bar{b}}z$. One does this by choosing the challenge $e_{\bar{b}}$ and then responding to challenge e_b, which is derived based on e and $e_{\bar{b}}$. More specifically, one publishes

$$\begin{aligned}(d_0, d_1, e_0, e_1, f_0, f_1) : r \leftarrow_r \mathbb{Z}_q,\ e_{\bar{b}} \leftarrow_r \mathbb{Z}_q,\ f_{\bar{b}} \leftarrow_r \mathbb{Z}_q,\\ d_b \leftarrow h^r,\ d_{\bar{b}} \leftarrow h^{f_{\bar{b}}}(g^{-\bar{b}}z)^{e_{\bar{b}}},\\ e \leftarrow H(z, d_0, d_1),\\ e_b \leftarrow ((e - e_{\bar{b}}) \bmod q),\\ f_b \leftarrow ((r - e_b y) \bmod q)\end{aligned} \tag{17}$$

To verify the proof, one checks the following

$$\begin{aligned} & e = e_b + e_{\bar{b}} \bmod q \\ \bigwedge\ & d_b = h^{f_b} \left(g^{-b}z\right)^{e_b} \\ \bigwedge\ & d_{\bar{b}} = h^{f_{\bar{b}}} \left(g^{-\bar{b}}z\right)^{e_{\bar{b}}} \end{aligned} \tag{18}$$

When the prover follows the protocol, we note that $e = e_b + e_{\bar{b}} \bmod q$ and $d_{\bar{b}} = h^{f_{\bar{b}}} \left(g^{-\bar{b}}z\right)^{e_{\bar{b}}}$ hold by setup, and we have:

$$h^{f_b} \left(g^{-b}z\right)^{e_b} = h^{r-e_b y} (h^y)^{e_b} = h^r = d_b$$