Split-Ballot Voting: Everlasting Privacy With Distributed Trust

TAL MORAN
Weizmann Institute of Science, Israel
and
MONI NAOR
Weizmann Institute of Science, Israel

In this paper we propose a new voting protocol with several desirable security properties. The voting stage of the protocol can be performed by humans without computers; it provides every voter with the means to verify that all the votes were counted correctly (universal verifiability) while preserving ballot secrecy. The protocol has "everlasting privacy": even a computationally unbounded adversary gains no information about specific votes from observing the protocol's output. Unlike previous protocols with these properties, this protocol distributes trust between two authorities: a single corrupt authority will not cause voter privacy to be breached. Finally, the protocol is *receipt-free*: a voter cannot prove how she voted even if she wants to do so. We formally prove the security of the protocol in the Universal Composability framework, based on number-theoretic assumptions.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*; E.3 [**Data**]: Data Encryption—*Public Key Cryptosystems*

General Terms: Security, Theory, Human Factors

Additional Key Words and Phrases: Voting Protocol,Everlasting Privacy,Universally-Composable,Receipt-Free

1. INTRODUCTION

Recent years have seen increased interest in voting systems, with a focus on improving their integrity and trustworthiness. This focus has given an impetus to cryptographic research into voting protocols. Embracing cryptography allows us to achieve high levels of verifiability, and hence trustworthiness (every voter can check that her vote was counted correctly), without sacrificing the basic requirements of ballot secrecy and resistance to coercion.

A "perfect" voting protocol must satisfy a long list of requirements. Among the most important are:

This work was partially supported by the Israel Science Foundation
Moni Naor is the Incumbent of the Judith Kleeman Professorial Chair

Accuracy The final tally must reflect the voters' wishes.

Privacy A voter's vote must not be revealed to other parties.

Receipt-Freeness A voter should not be able to prove how she voted (this is important in order to prevent vote-buying and coercion).

Universal Verifiability Voters should be able to verify that their own votes were "cast as intended", and any interested party should be able to verify that all the votes were "counted as cast".

Surprisingly, using cryptographic tools we can construct protocols that satisfy all four of these properties simultaneously. Unfortunately, applying cryptographic techniques introduces new problems. One of these is that cryptographic protocols are often based on computational assumptions (e.g., the infeasibility of solving a particular problem). Some computational assumptions, however, may have a built-in time limit (e.g., Adi Shamir estimated that all existing public-key systems, with key-lengths in use today, will remain secure for less than thirty years [Shamir 2006]).

A voting protocol is said to provide *information-theoretic privacy* if a computationally unbounded adversary does not gain any information about individual votes (apart from the final tally). If the privacy of the votes depends on computational assumptions, we say the protocol provides *computational privacy*. Note that to coerce a voter, it is enough that the voter *believe* there is a good chance of her privacy being violated, whether or not it is actually the case (so even if Shamir's estimate is unduly pessimistic, the fact that such an estimate was made by an expert may be enough to allow voter coercion). Therefore, protocols that provide computational privacy may not be proof against coercion: the voter may fear that her vote will become public some time in the future.

While integrity that depends on computational assumptions only requires the assumptions to hold during the election, privacy that depends on computational assumptions requires them to hold forever. To borrow a term from Aumann, Ding and Rabin [Aumann et al. 2002], we can say that information-theoretic privacy is *everlasting* privacy.

A second problem that cryptographic voting protocols must consider is that most cryptographic techniques require complex computations that unaided humans are unable to perform. However, voters may not trust voting computers to do these calculations for them. This mistrust is quite reasonable, because there is no way for them to tell if a computer is actually doing what it is supposed to be doing (as a trivial example consider a voting program that lets a voter choose a candidate, and then claims to cast a vote for that candidate; it could just as easily be casting a vote for a different candidate).

Finally, a problem that is applicable to all voting protocols is the problem of concentrating trust. We would like to construct protocols that don't have a "single point of failure" with respect to their security guarantees. Many protocols involve a "voting authority". In some protocols, this authority is a single-point of failure with respect to privacy (or, in extreme cases, integrity). Protocols that require the voter to input their votes to a computer automatically have a single point of failure: the computer is a single entity that "knows" the vote. This is not an idle concern: many ways exist for a corrupt computer to undetectably output information to an outside

party (in some cases, the protocol itself provides such "subliminal channels").

1.1 Our Contributions

In this paper we introduce the first universally-verifiable voting protocol with everlasting privacy that can be performed by unaided humans and distributes trust across more than one voting authority. This protocol has reasonable complexity ($O(m)$ exponentiations per voter, where m is the number of candidates) and is efficient enough to be used in practice.

We formally prove our protocol is secure in the Universal Composability (UC) framework, which provides very strong notions of security. Loosely speaking, we show that running our protocol is as secure as running the election using an absolutely trustworthy third party (the "ideal functionality"), to whom all the voters secretly communicate their choices, and who then announces the final tally (a formal definition of this functionality appears in Section 4).

Surprisingly, we can attain this level of security even though we base the voting protocol on commitment and encryption schemes that are not, themselves, universally composable (we propose using a modification of the Pedersen commitment scheme together with Paillier encryption; see Appendix A for details).

As part of the formal proof of security, we can specify precisely what assumptions we make when we claim the protocol is secure (this is not the case for most existing voting protocols, that lack formal proofs completely).

In addition, we formally prove that our protocol is receipt-free (voters cannot prove for whom they voted, even if they want to), using a simulation-based definition of receipt-freeness previously introduced by the authors [Moran and Naor 2006].

Our insistence on rigorous proofs of correctness is not just "formalism for the sake of formalism". We believe that formal proofs of security provide several very significant practical advantages. First, a precondition for proving security is providing a formal definition of what we are trying to prove. This definition is useful in itself: it gives us a better understanding of what our protocol achieves, where it can be used and what its failure modes are. This is especially evident for definitions in simulation-based models (such as universal composability), since the definition of an ideal functionality is usually very intuitive.

Secondly, even fairly simple protocols may have hard to find weaknesses. Without a formal proof, we can never be certain that we have considered all possible avenues of attack. A formal proof lists a small number of assumptions that imply the security of the protocol. This means that to verify that a particular implementation is secure, we can concentrate on checking only these assumptions: as long as they are all satisfied, we can be certain an attack will not come from an unexpected direction. To illustrate this point, we demonstrate a subtle attack against the receipt-freeness of the Punchscan voting system [Chaum 2006] (see Section 2.4).

Finally, even though formal proofs are not "foolproof" — our definitions may not capture the "correct" notion of security, or the proof itself may contain errors — they give us a basis and a common language for meaningful discussions about protocols' security.

1.2 Related Work

Voting Protocols. Chaum proposed the first published electronic voting scheme in 1981 [Chaum 1981]. Many additional protocols were suggested since Chaum's. Among the more notable are [Fujioka et al. 1992; Cohen(Benaloh) and Fischer 1985; Benaloh and Tuinstra 1994; Cramer et al. 1996; Cramer et al. 1997; Hirt and Sako 2000].

Most of the proposed voting schemes satisfy the accuracy, privacy and universal-verifiability properties. However, only a small fraction satisfy, in addition, the property of of receipt-freeness. Benaloh and Tuinstra [1994] were the first to define this concept, and to give a protocol that achieves it (it turned out that their full protocol was not, in fact, receipt free, although their single-authority version was [Hirt and Sako 2000]). To satisfy receipt-freeness, Benaloh and Tuinstra also required a "voting booth": physically untappable channels between the voting authority and the voter.

Human Considerations. Almost all the existing protocols require complex computation on the part of the voter (infeasible for an unaided human). Thus, they require the voter to trust that the computer casting the ballot on her behalf is accurately reflecting her intentions. Chaum [2004], and later Neff [2004], proposed universally-verifiable receipt-free voting schemes that overcome this problem. Reynolds [2005] proposed another protocol similar to Neff's.

All three schemes are based in the "traditional" setting, in which voters cast their ballots in the privacy of a voting booth. Instead of a ballot box, the booth contains a "Direct Recording Electronic" (DRE) voting machine. The voter communicates her choice to the DRE (e.g., using a touch-screen or keyboard). The DRE encrypts her vote and posts the encrypted ballot on a public bulletin board. It then proves to the voter, in the privacy of the voting booth, that the encrypted ballot is truly an encryption of her intended vote.

Chaum's original protocol used Visual Cryptography [Naor and Shamir 1994] to enable the human voter to read a complete (two-part) ballot that was later separated into two encrypted parts, and so his scheme required special printers and transparencies. Bryans and Ryan showed how to simplify this part of the protocol to use a standard printer [Bryans and Ryan 2004; Ryan 2005]. A newer idea of Chaum's is the Punchscan voting system [Chaum 2006], which we describe in more detail in Section 2.4.

Previously, the authors proposed a voting protocol, based on statistically-hiding commitments, that combines everlasting security and a human-centric interface [Moran and Naor 2006]. This protocol requires a DRE, and inherently makes use of the fact that there is a single authority (the DRE plays the part of the voting authority).

Adida and Rivest [2006] suggest the "Scratch&Vote" system, which makes use of *scratch-off cards* to provide receipt-freeness and "instant" verifiability (at the polling place). Their scheme publishes encryptions of the votes, and is therefore only computationally private.

Our new scheme follows the trend of basing protocols on physical assumptions in the traditional voting-booth setting. Unlike most of the previous schemes we also provide a rigorous proof that our scheme actually meets its security goals.

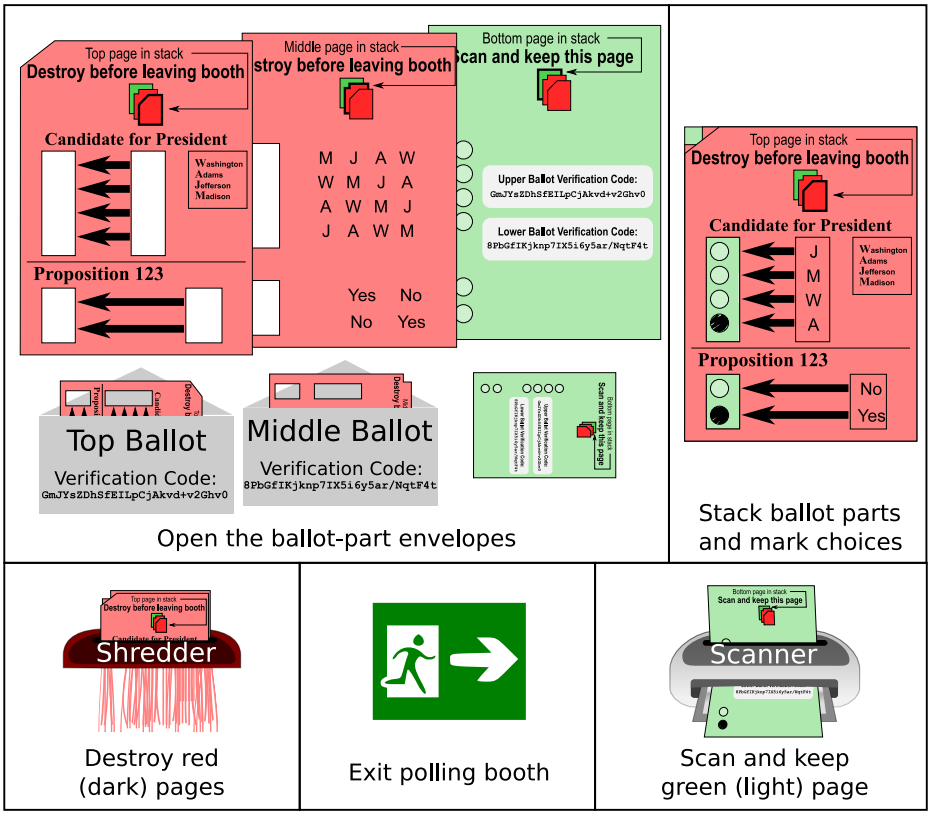

Fig. 1. Illustrated Sample Vote

2. INFORMAL OVERVIEW OF THE SPLIT-BALLOT PROTOCOL

Our voting scheme uses two independent voting authorities that are responsible for preparing the paper ballots, counting the votes and proving that the announced tally is correct.

If both authorities are honest, the election is guaranteed to be accurate, information-theoretically private and receipt-free. If at least one of the authorities is honest, the election is guaranteed to be accurate and private (but now has only computational privacy, and may no longer be receipt-free). If both authorities are corrupt, the tally is still guaranteed to be accurate, but privacy is no longer guaranteed.

An election consists of four phases:

(1) Setup: In this stage the keys for the commitment and encryption schemes are set up and ballots are prepared.
(2) Voting: Voters cast their ballots. This stage is designed to be performed using pencil and paper, although computers may be used to improve the user experience.

A vote consists of four ballots, two from each voting authority. The voter

selects one ballot from each authority for auditing (they will not be used for voting). The remaining two ballots are used to vote. The voter's choices on both ballots, taken together, uniquely define the vote. A partial copy of each ballot is retained by the voter as a "verification receipt" (a more detailed description appears in Section 2.2).

(3) Tally: The two authorities publish all of the ballots. Voters may verify that their receipts appear correctly in the published tally. The two authorities then cooperate to tally the votes. The final result is a public proof that the tally is correct.

(4) Universal Verification: In this phase any interested party can download the contents of the public bulletin board and verify that the authorities correctly tallied the votes.

2.1 Shuffling Commitments

One of the main contributions of this paper is achieving "everlasting privacy" with more than one voting authority. At first glance, this seems paradoxical: if a voting authority publishes any information at all about the votes (even encrypted), the scheme can no longer be information-theoretically private. On the other hand, without publishing information about the votes, how can two voting authorities combine their information?

We overcome this apparent contradiction by introducing the "oblivious commitment shuffle": a way for independent authorities to verifiably shuffle perfectly-hiding commitments (which will give us information-theoretic privacy).

The problem of verifiably shuffling a vector of *encrypted* values has been well studied. The most commonly used scheme involves multiple authorities who successively shuffle the encrypted vector, each using a secret permutation, and then prove that the resulting vector of encrypted values is valid. Finally, the authorities cooperate to decrypt the ultimate output of the chain. If even one of the authorities is honest (and keeps its permutation secret), the remaining authorities gain no information beyond the final tally.

This type of scheme breaks down when we try to apply it to perfectly-hiding commitments rather than encryptions. The problem is that in a perfectly-hiding commitment, the committed value cannot be determined from the commitment itself. Thus, the standard method of opening the commitments after shuffling cannot be used.

The way we bypass the problem is to allow the authorities to communicate privately using a homomorphic encryption scheme. This private communication is not perfectly hiding (in fact, the encryptions are perfectly binding commitments), but the voting scheme itself can remain information-theoretically private because the encryptions are never published. The trick is to encrypt separately both the message *and the randomness* used in the commitments. We use a homomorphic encryption scheme over the same group as the corresponding commitment. When the first authority shuffles the commitments, it simultaneously shuffles the encryptions (which were generated by the other authority). By opening the shuffled encryptions, the second authority learns the contents and randomness of the shuffled commitments (without learning anything about their original order). The second authority can now perform a traditional commitment shuffle.

2.2 Human Capability

Our protocol makes two questionable assumptions about human voters: that they can randomly select a bit (to determine which ballots to audit), and that they perform modular addition (to split their vote between the two authorities). The first is a fairly standard assumption (in fact, we do not require uniform randomness, only high min-entropy). The second seems highly suspect. However, it is possible to implement the voting protocol so that the modular addition occurs implicitly as a result of a more "natural" action for humans.

We propose an interface that borrows heavily from Punchscan's in order to make the voting task more intuitive. The basic idea is to form the ballot from three separate pages. The first page contains the list of candidates, along with a letter or symbol denoting each (this letter can be fixed before the election). The second page contains a table of letters: each column of the table is a permutation of the candidates. The third page is the one used to record the vote; it contains a scannable bubble for each row of the table in the middle page.

Holes are cut in the top page and middle pages, so that when all three are stacked a single random column of the table on the middle page is visible, as are the bubbles on the bottom page. The voter selects a candidate by marking the bubble corresponding to her choice. Since one authority randomly selects the table (on the middle page) and the other authority randomly selects which of its columns is used (determined by the position of the hole in the top page), the position of the bubble marked by the voter does not give information about her choice unless both the middle and top pages are also known.

2.3 Vote Casting Example

To help clarify the voting process, we give a concrete example, describing a typical voter's view of an election (this view is illustrated in Figure 1). The election is for the office of president, and also includes a poll on "Proposition 123". The presidential candidates are Washington, Adams, Jefferson and Madison.

Sarah, the voter, enters the polling place and receives two pairs of ballot pages in sealed envelopes, each pair consisting of a "Top" ballot page and a "Middle" ballot page (we can think of the two voting authorities as the "Top" authority and the "Middle" authority). Each envelope is marked either "Top" or "Middle", and has a printed "verification code" (this code is actually a commitment to the public section of the ballot, as described in Section 5.1). Sarah first chooses a pair of ballot pages to audit. This pair is immediately opened, and the "red" (dark) ballot pages inside the envelopes are scanned, as are the verification codes on the envelopes. Sarah is allowed to keep all parts of the audited ballots.

The election officials give Sarah a green (light) "bottom page" that is printed with the verification codes from both the remaining (unopened) envelopes (alternatively, the verification codes could be printed on a sticker that is affixed to the green page before handing it to Sarah). She enters the polling booth with the green page and both unopened envelopes.

Inside the polling booth, Sarah opens the envelopes and takes out the red pages. The middle page is printed with a table of letters representing the candidates (the letters were chosen in advance to be the first letter of the candidate's surname).

The order of the letters in the table is chosen randomly by the Middle authority (different ballot pages may have different orders). Similarly, the order of the "Yes" and "No" responses to Proposition 123 is random. The top page has a hole cut out that reveals a single column of the table — which column is randomly chosen by the Top authority. Sarah stacks all three pages (the top ballot page, the middle ballot part, and the green "bottom page"). Taken together, these pages form a complete ballot. Sarah wants to vote for Adams and to vote Yes on Proposition 123. She finds her candidate's letter on the ballot, and marks the corresponding bubble (the marks themselves are made on the green, bottom page that can be seen through the holes in the middle and top pages). She also finds the "Yes" choice for Proposition 123, and marks its corresponding bubble.

Sarah then separates the pages. She shreds the red pages that were inside the envelopes. To prevent vote-selling and coercion attacks, Sarah must be *forced* to destroy the red pages (e.g., perhaps the output of the shredder is visible to election officials outside the voting booth).

Sarah exits the voting booth holding only the marked, green page. This sheet of paper is then scanned (with the help of the election officials). The scanner can give immediate output so Sarah can verify that she filled the bubbles correctly, and that the scanner correctly identified her marks. Note that Sarah doesn't have to *trust* the scanner (or its software) in any way: The green page and the audited ballots will be kept by Sarah as receipts which she can use to prove that her vote was not correctly tabulated (if this does occur). At home Sarah will make sure that the verification code printed on the pages, together with the positions of the marked bubbles, are published on the bulletin board by the voting authorities. Alternatively, she can hand the receipts over to a helper organization that will perform the verification on her behalf.

2.4 The Importance of Rigorous Proofs of Security for Voting Protocols

To demonstrate why formal proofs of security are important, we describe a vote-buying attack against a previous version of the Punchscan voting protocol. The purpose of this section is not to disparage Punchscan; on the contrary, we use Punchscan as an example because it is one of the simplest protocols to understand and has been used in practice. A closer look at other voting protocols may reveal similar problems. Our aim is to encourage the use of formal security analysis to detect (and prevent) such vulnerabilities.

We very briefly describe the voter's view of the Punchscan protocol, using as an example an election race between Alice and Bob. The ballot consists of two pages, one on top of the other. The top page contains the candidates' names, and assigns each a random letter (either A or B). There are two holes in the top page through which the bottom page can be seen. On the bottom page, the letters A and B appear in a random order (so that one letter can be seen through each hole in the top page). Thus, the voter is presented with one of the four possible ballot configurations (shown in Figure 2).

To vote, the voter marks the letter corresponding to her candidate using a wide marker: this marks both the top and bottom pages simultaneously. The two pages are then separated. The voter chooses one of the pages to scan (and keep as a

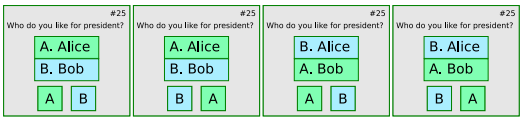

Fig. 2. Punchscan Ballot Configurations

receipt), while the other is shredded (these steps are shown in Figure 3).

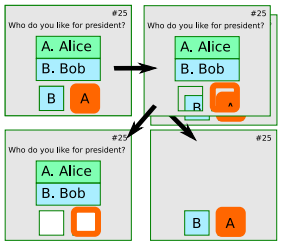

Fig. 3. Punchscan Ballot

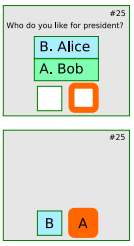

Fig. 4. "Bad" Receipts

Each pair of pages has a short id, which a voting authority can use to determine what was printed on each of the pages (this allows the authority to determine the voter's vote even though it only receives a single page). For someone who does not know the contents of the shredded page, the receipt does not give any information about the voter's choice.

Giving each voter a receipt for her vote is extremely problematic in traditional voting systems, since the receipt can be used to coerce voters or to buy votes. Punchscan attempts to prevent vote-buying by making sure that the receipt does not contain any information about the voter's choice. At first glance, this idea seems to work: if an adversary just asks a voter to vote for a particular candidate (by following the Punchscan protocol honestly), there is no way the adversary can tell, just by looking at the receipt, whether the voter followed his instructions or not.

Below, we show that for a slightly more sophisticated adversary, a vote-buying attack *is* possible against Punchscan.

2.4.1 A Vote Buying Attack. To demonstrate the attack, we continue to use the Alice/Bob election example. Suppose the coercer wants to bias the vote towards Alice. In this case, he publishes that he will pay for any receipt *except* those shown in Figure 4 (i.e., everything except a "B,A" bottom page on which "A" was marked, and a "B,A" top page on which the right hole was marked).

This attack will force one fourth of the voters to vote for Alice in order to get paid. To see why, consider the four possible ballot configurations (in Figure 2). Since the coercer will accept any marking on an "A,B" top page or an "A,B" bottom page, in three of the four configurations the voter can vote as she wishes. However, if both the top and the bottom pages are "B,A" pages (this occurs in one fourth of the cases), the voter is forced to vote for Alice if she wants to return an acceptable receipt.

Although three-fourths of the voters can vote for any candidate, this attack is still entirely practical. When a race is close, only a small number of votes must be changed to tip the result in one direction. Compared to the "worst possible" system in which an adversary can buy votes directly, Punchscan requires the attacker to spend only four times as much to buy the same number of votes. Since the receipts are published, this attack can be performed remotely (e.g., over the internet), making it much worse than a "standard" vote-buying attack (such as chain-voting) that must be performed in person.

We note that the current version of Punchscan (as described in [Popoveniuc and Hosp 2006]) instructs the voter to commit to the layer she will take before entering the voting booth. This requirement does suffice to foil the attack described above. Similar attacks, however, may still be possible. The point we hope to make is that, lacking a formal proof of security, it is very hard to be certain.

3. UNDERLYING ASSUMPTIONS

One of the important advantages of formally analyzing voting protocols is that we can state the specific assumptions under which our security guarantees hold. Our protocol uses a combination of physical and cryptographic assumptions. Below, we define the assumptions and give a brief justification for each.

3.1 Physical Assumptions

Undeniable Ballots. To allow voters to complain convincingly about invalid ballots, they must be *undeniable*: a voter should be able to prove that the ballot was created by the voting authority. This type of requirement is standard for many physical objects: money, lottery-tickets, etc.

Forced Private Erasure. In order to preserve the receipt-freeness of the protocol, we require voters to physically erase information from the ballots they used. The erasure assumption is made by a number of existing voting schemes that require the voter to choose some part of the ballot to securely discard (e.g., Punchscan [Chaum 2006], Scratch&Vote [Adida and Rivest 2006]). In practice, this can be done by shredding, by chemical solvent, etc.

At first glance, it might appear that simply spoiling a ballot that was not correctly erased is sufficient. However, this is not the case; the voter must be *forced* to erase the designated content. Otherwise, a coercer can mount a vote-buying attack similar to the one described in section 2.4, where some voters are told to invalidate their ballots by refusing to erase them (and showing the complete ballot to the coercer).

Since only the voter should be able to see the contents of the erased part of the ballot, finding a good mechanism to enforce erasure may be difficult (e.g., handing it to an official to shred won't work). However, a large-scale attack that relies on circumventing this assumption may be detected by counting the number of spoiled ballots.

Voting Booth. In order to preserve privacy and receipt-freeness, the voter must be able to perform some actions privately. The actions the voter performs in the voting booth are opening sealed ballots, reading their contents and erasing part of the ballot.

Untappable Channels. We use untappable channels in two different ways. First,

in order to guarantee everlasting privacy and receipt-freeness, ballots must be delivered from the voting authorities to the voter without any information about their contents leaking to a third party. The amount of data each voter must receive is small, however, and the untappable channel may be implmented, for example, using sealed envelopes.

Second, for the same reason, communication between the two voting authorities is also assumed to take place using untappable private channels. The amount of information exchanged is larger in this case, but this is a fairly reasonable assumption: the voting authorities can be physically close and connected by direct physical channels.

The untappable channel can also be replaced by encryption using a one-time pad (since this is also information-theoretically private). However, to simplify the proof we consider only an ideal untappable channel in this paper.

Public Bulletin Board. The public bulletin board is a common assumption in universally-verifiable voting protocols. This is usually modeled as a broadcast channel, or as append-only storage with read-access for all parties. A possible implementation is a web-site that is constantly monitored by multiple verifiers to ensure that nothing is erased or modified.

Random Beacon. The random beacon, originally introduced by Rabin [Rabin 1983], is a source of independently distributed, uniformly random strings. The main assumption about the beacon is that it is unpredictable. In practice, the beacon can be implemented in many ways, such as by some physical source believed to be unpredictable (e.g., cosmic radiation, weather, etc.), or by a distributed computation with multiple verifiers.

We use the beacon for choosing the public-key of our commitment scheme and to replace the verifier in zero-knowledge proofs. For the zero-knowledge proofs, we can replace the beacon assumption by a random oracle (this is the Fiat-Shamir heuristic): the entire protocol transcript so far is taken as the index in the random oracle that is used as the next bit to be sent by the beacon.

3.2 Cryptographic Assumptions

Our protocol is based on two cryptographic primitives: perfectly-hiding homomorphic commitment and homomorphic encryption. The homomorphic commitment requires some special properties.

Homomorphic Commitment. A homomorphic commitment scheme consists of a tuple of algorithms: K, C, P_K, and V_K. $K\colon \{0,1\}^\ell \times \{0,1\}^\ell \mapsto \mathcal{K}$ accepts a public random bit-string and a private auxiliary and generates a commitment public key $cpk \in \mathcal{K}$. C is the commitment function, parametrized by the public key, mapping from a message group $(\mathcal{M}, +)$ and a randomizer group $(\mathcal{R}, +)$ to the group of commitments $(\mathcal{C}, \cdot)$.

P_K and V_K are a zero-knowledge "prover" and "verifier" for the key generation: these are both interactive machines. The prover receives the same input as the key generator, while the verifier receives only the public random string and the public key. To allow the verification to be performed publicly (using a random beacon), we require that all of the messages sent by V_K to P_K are uniformly distributed random strings.

For any probabilistic polynomial time turing machines (PPTs) K^*, P_K^* (corre-

sponding to an adversarial key-generating algorithm and prover), when $cpk \leftarrow K^*(r_K)$, $r_K \in_R \{0,1\}^\ell$ is chosen uniformly at random then, with all but negligible probability (the probability is over the choice of r_K and the random coins of K^*, P_K^* and V_K), either the output of $V_K(r_K, cpk)$ when interacting with P_K^* is 0 (i.e., the verification of the public-key fails) or the following properties must hold:

(1) Perfectly Hiding: For any $m_1, m_2 \in \mathcal{M}$, the random variables $C(m_1, r)$ and $C(m_2, r)$ must be identically distributed when r is taken uniformly at random from $\mathcal{R}$. (Note that we can replace this property with *statistically* hiding commitment, but for simplicity of the proof we require the stronger notion).
(2) Computationally Binding: For any PPT $\mathcal{A}$ (with access to the private coins of K^*), the probability that $\mathcal{A}(cpk)$ can output $(m_1, r_1) \neq (m_2, r_2) \in \mathcal{M} \times \mathcal{R}$ such that $C_{cpk}(m_1, r_1) = C_{cpk}(m_2, r_2)$ must be negligible. The probability is over the random coins of K^*, $\mathcal{A}$ and r_K.
(3) Homomorphic in both $\mathcal{M}$ and $\mathcal{R}$: for all $(m_1, r_1), (m_2, r_2) \in \mathcal{M} \times \mathcal{R}$, and all but a negligible fraction of keys, $C(m_1, r_1) \cdot C(m_2, r_2) = C(m_1 + m_2, r_1 + r_2)$.
(4) Symmetry: The tuple (K, C'), where $C'(m, r) \doteq C(r, m)$ should also be a commitment scheme satisfying the hiding and binding properties (i.e., it should be possible to use $C(m, r)$ as a commitment to r).

Finally we also require the interaction between P_K and V_K to be zero-knowledge: there should exist an efficient simulator that, for every r_K and $K(r_k, aux)$, produces a simulated transcript of the interaction that is computationally-indistinguishable from a real one — even though it is not given aux (the secret auxiliary input to K).

Simulated Equivocability. For achieving UC security, we require the commitment scheme to have two additional algorithms: $K' \colon \{0,1\}^{\ell'} \mapsto \{0,1\}^\ell$, $C' \colon \{0,1\}^{\ell'} \times \mathcal{C} \times \mathcal{M} \mapsto \mathcal{R}$, such that the output of K' is uniformly random. The scheme must satisfy an additional property when we replace r_K with $K'(l)$, where $l \in_R \{0,1\}^\ell$:

(5) Perfect Equivocability: For every $m \in \mathcal{M}$ and $c \in \mathcal{C}$, $C_{K^*(K'(l))}(m, C'(l, c, m)) = c$.

That is, it is possible to generate a public-key that is identical to a normal public key, but with additional side information that can be used to efficiently open every commitment to any value

Homomorphic Public-Key Encryption. The second cryptographic building block we use is a homomorphic public-key encryption scheme. We actually need two encryption schemes, one whose message space is $\mathcal{M}$ and the other whose message space is $\mathcal{R}$ (where $\mathcal{M}$ and $\mathcal{R}$ are as defined for the commitment scheme). The schemes are specified by the algorithm triplets $(KG^{(\mathcal{M})}, E^{(\mathcal{M})}, D^{(\mathcal{M})})$ and $(KG^{(\mathcal{R})}, E^{(\mathcal{R})}, D^{(\mathcal{R})})$, where KG is the key-generation algorithm, $E^{(\mathcal{X})} \colon \mathcal{X} \times \mathcal{T} \mapsto \mathcal{E}^{(\mathcal{X})}$ the encryption algorithm and $D^{(\mathcal{X})} \colon \mathcal{E}^{(\mathcal{X})} \mapsto \mathcal{X}$ the decryption algorithm. We require the encryption schemes to be semantically secure and homomorphic in their message spaces: for every $x_1, x_1 \in \mathcal{X}$ and any $r_1, r_2 \in \mathcal{T}$, there must exist $r' \in \mathcal{T}$ such that $E^{(\mathcal{X})}(x_1, r_1) \cdot E^{(\mathcal{X})}(x_2, r_2) = E^{(\mathcal{X})}(x_1 + x_2, r')$. .

We do not require the encryption scheme to be homomorphic in its randomness, but we do require, for every x_1, r_1, x_2, that r' is uniformly distributed in $\mathcal{T}$ when r_2 is chosen uniformly.

To reduce clutter, when it is obvious from context we omit the key parameter for the commitment scheme (e.g., we write $C(m, r)$ instead of $C_{cpk}(m, r)$), and the randomness and superscript for the encryption schemes (e.g., we write $E(m)$ to describe an encryption of m).

Below, we use only the abstract properties of the encryption and commitment schemes. For an actual implementation, we propose using the Paillier encryption scheme (where messages are in $\mathbb{Z}_n$ for a composite n, together with a modified version of Pedersen Commitment (where both messages and randomness are also in $\mathbb{Z}_n$). More details can be found in Appendix A.

4. THREAT MODEL AND SECURITY

We define and prove the security properties of our protocol using a simulation paradigm. The protocol's functionality is defined by describing how it would work in an "ideal world", in which there exists a completely trusted third party. Informally, our security claim is that any attack an adversary can perform on the protocol in the real world can be transformed into an attack on the functionality in the ideal world. This approach has the advantage of allowing us to gain a better intuitive understanding of the protocol's security guarantees, when compared to the game-based or property-based approach for defining security.

The basic functionality is defined and proved in Canetti's Universal Composability framework [Canetti 2000]. This provides extremely strong guarantees of security, including security under arbitrary composition with other protocols. The ideal voting functionality, described below, explicitly specifies what abilities the adversary gains by corrupting the different parties involved.

We also guarantee receipt-freeness, a property that is not captured by the standard UC definitions, using a similar simulation-based definition (see Appendix C).

4.1 Ideal Voting Functionality

The voting functionality defines a number of different parties: n voters, two voting authorities A_1 and A_2, a verifier and an adversary. The voting authorities' only action is to specify the end of the voting phase. Also, there are some actions the adversary can perform only after corrupting one (or both) of the voting authorities. The verifier is the only party with output. If the protocol terminates successfully, the verifier outputs the tally, otherwise it outputs $\perp$ (this corresponds to cheating being detected).

When one (or both) of the voting authorities are corrupt, we allow the adversary to change the final tally, as long as the total number of votes changed is less than the security parameter k (we consider 2^{-k} negligible).[1] This is modeled by giving the tally privately to the adversary, and letting the adversary announce an arbitrary tally using the **Announce** command (described below). If one of the authorities is corrupt, we also allow the adversary to retroactively change the votes of corrupt voters, as a function of the tally (if we were to use a universally-composable encryption scheme, rather than one that is just semantically secure, we could do away with this requirement).

[1]This is a fairly common assumption in cryptographic voting protocols (appearing in [Chaum 2004; Bryans and Ryan 2004; Ryan 2005; Chaum 2006], among others).

If neither of the voting authorities is corrupt, the adversary cannot cause the functionality to halt. The formal specification for the voting functionality, $\mathcal{F}_{\text{vote}}$, follows:

Vote v, x_v On receiving this command from voter v, the functionality stores the tuple (v, x_v) in the vote database S and outputs "v has voted" to the adversary. The functionality then ignores further messages from voter v. The functionality will also accept this message from the adversary if v was previously corrupted (in this case an existing (v, x_v) tuple can be replaced). If one of the authorities was corrupted before the first **Vote** command was sent, the functionality will also accept this message from the adversary after the **Tally** command has been received (to change the vote of voters that were corrupted before the tally).

Vote $v, *$ This command signifies a forced random vote. It is accepted from the adversary only if voter v is coerced or corrupted. In that case, the functionality chooses a new random value $x_v \in_R \mathbb{Z}_m$, and stores the tuple (v, x_v) in the database.

Vote $v, \perp$ This command signifies a forced abstention. It is accepted from the adversary only if voter v is coerced or corrupted. In that case, the functionality deletes the tuple (v, x_v) from the database.

Tally On receiving this command from an authority, the functionality computes $\tau_i = |\{(v, x_v) \in S \mid x_v = i\}|$ for all $i \in \mathbb{Z}_m$. If none of the voting authorities are corrupt, the functionality sends the tally $\tau_0, \ldots, \tau_{m-1}$ to the verifier and halts (this is a successful termination). Otherwise (if at least one of the voting authorities is corrupt), it sends the tally, $\tau_0, \ldots, \tau_{m-1}$, to the adversary.

Announce $\tau'_0, \ldots, \tau'_{m-1}$ On receiving this command from the adversary, the functionality verifies that the **Tally** command was previously received. It then computes $d = \sum_{i=0}^{m-1} |\tau_i - \tau'_i|$ (if one of the authorities is corrupt and the adversary changed corrupt voters' choices after the **Tally** command was received, the functionality recomputes $\tau_0, \ldots, \tau_{m-1}$ before computing d). If $d < k$ (where k is the security parameter) it outputs the tally $\tau'_0, \ldots, \tau'_{m-1}$ to the verifier and halts (this is considered a successful termination).

Corrupt v On receiving this command from the adversary, the functionality sends x_v to the adversary (if there exists a tuple $(v, x_v) \in S$).

Corrupt A_a On receiving this command from the adversary, the functionality marks the voting authority A_a as corrupted.

RevealVotes On receiving this command from the adversary, the functionality verifies that both of the voting authorities A_1 and A_2 are corrupt. If this is the case, it sends the vote database S to the adversary.

Halt On receiving this command from the adversary, the functionality verifies that at least one of the voting authorities is corrupt. If so, it outputs $\perp$ to the verifier and halts.

Our main result is a protocol that realizes the ideal functionality $\mathcal{F}_{\text{vote}}$ in the universal composability model. A formal statement of this is given in Theorem 5.1, with a proof in Section 6.

4.2 Receipt-Freeness

As previously discussed, in a voting protocol assuring privacy is not enough. In order to prevent vote-buying and coercion, we must ensure *receipt-freeness*: a voter shouldn't be able to prove how she voted even if she wants to. We use the definition of receipt-freeness from [Moran and Naor 2006], an extension of Canetti and Gennaro's *incoercible computation* [Canetti and Gennaro 1996]. This definition of receipt-freeness is also simulation based, in the spirit of our other security definitions.

Parties all receive a fake input, in addition to their real one. A coerced player will use the fake input to answer the adversary's queries about the past view (before it was coerced). The adversary is not limited to passive queries, however. Once a player is coerced, the adversary can give it an *arbitrary strategy* (i.e. commands the player should follow instead of the real protocol interactions). We call coerced players that actually follow the adversary's commands "puppets".

A receipt-free protocol, in addition to specifying what players should do if they are honest, must also specify what players should do if they are coerced; we call this a "coercion-resistance strategy" The coercion-resistance strategy is a generalization of the "faking algorithm" in Canetti and Gennaro's definition — the faking algorithm only supplies an answer to a single query ("what was the randomness used for the protocol"), while the coercion-resistance strategy must tell the party how to react to any command given by the adversary.

Intuitively, a protocol is receipt-free if no adversary can distinguish between a party with real input x that is a puppet and one that has a fake input x (but a different real input) and is running the coercion-resistance strategy. At the same time, the computation's output should not change when we replace coerced parties running the coercion-resistance strategy with parties running the honest protocol (with their real inputs). Note that these conditions must hold even when the coercion-resistance strategy is known to the adversary.

In our original definition [Moran and Naor 2006], the adversary can force a party to abstain. We weaken this definition slightly, and allow the adversary to force a party to vote randomly. The intuition is that a uniformly random vote has the same effect, in expectation, as simply abstaining[2]. Our protocol is receipt-free under this definition (Theorem 5.2 gives a more precise statement of this fact).

Note that the intuition for why this is acceptable is not entirely correct: in some situations, the new definition can be significantly weaker. For example, when voting is compulsory, "buying" a random vote may be much cheaper than "buying" an abstention (the price would have to include the fine for not voting). Another situation where forcing randomization may be more powerful than forcing an abstention is if the margin of victory is important (such as in proportional elections). In many cases, however, the difference is not considered substantial enough to matter; we note that Punchscan and Prêt à Voter, two of the most widely-known universally-verifiable voting schemes, are also vulnerable to a forced randomization attack.

[2]Note that the attack we describe in Section 2.4 is not equivalent to forcing a random vote: the coercer forces voters to choose the desired candidate with higher probability than the competitor.

5. SPLIT-BALLOT VOTING PROTOCOL

In this section we give an abstract description of the split-ballot voting protocol (by abstract, we mean we that we describe the logical operations performed by the parties without describing a physical implementation). In the interest of clarity, we restrict ourselves to two voting authorities A_1,A_2, n voters and a single poll question with answers in the group $\mathbb{Z}_m$. We assume the existence of a homomorphic commitment scheme (K, C) (with the properties defined in Section 3.2) whose message space is a group $(\mathcal{M}, +)$, randomizer space a group $(\mathcal{R}, +)$, and commitment space a group $(\mathcal{C}, \cdot)$. Our protocol requires $\mathcal{M}$ to be cyclic and have a large order: $|\mathcal{M}| \geq 2^{2k+2}$, and we assume $m < 2^k$ (k is the security parameter defined in Section 4.1). Furthermore, we assume the existence of homomorphic encryption schemes with the corresponding message spaces.

5.1 Setup

The initial setup involves:

(1) Choosing the system parameters (these consist of the commitment scheme public key and the encryption scheme public/private key pair). Authority A_1 runs $KG^{(\mathcal{M})}$ and $KG^{(\mathcal{R})}$, producing $(pk^{(\mathcal{M})}, sk^{(\mathcal{M})})$ and $(pk^{(\mathcal{R})}, sk^{(\mathcal{R})})$. A_1 sends the public keys over the private channel to authority A_2. It also runs K using the output of the random beacon as the public random string, and the private coins used in running $KG^{(\mathcal{M})}$ and $KG^{(\mathcal{R})}$ as the auxiliary. This produces the commitment public key, cpk. Authority A_1 now runs P_K using the random beacon in place of the verifier (this produces a public proof that the commitment key was generated correctly).

(2) Ballot preparation. Each voting authority prepares at least $2n$ ballot parts (the complete ballots are a combination of one part from each authority). We identify a ballot part by the tuple $\vec{w} = (a, i, b) \in \{1, 2\} \times [n] \times \{0, 1\}$, where A_a is the voting authority that generated the ballot part, i is the index of the voter to whom it will be sent and b a ballot part serial number. Each ballot part has a "public" section that is published and a "private" section that is shown only to the voter. The private section for ballot part $B_{\vec{w}}$ is a random value $t_{\vec{w}} \in_R \mathbb{Z}_m$. For $\vec{w} = (2, i, b)$ (i.e., ballot parts generated by authority A_2), the public section of $B_{\vec{w}}$ consists of a commitment to that value: $c_{\vec{w}} \doteq C(t_{\vec{w}}, r_{\vec{w}})$, where $r_{\vec{w}} \in_R \mathcal{R}$. For $\vec{w} = (1, i, b)$ (ballot parts generated by A_1), the public section contains a vector of commitments: $c_{\vec{w},0}, \ldots, c_{\vec{w},m-1}$, where $c_{\vec{w},j} \doteq C(t_{\vec{w}} + j \pmod m, r_{\vec{w},j})$, and $r_{\vec{w},j} \in_R \mathcal{R}$ (i.e., the commitments are to the numbers 0 through $m-1$ shifted by the value $t_{\vec{w}}$). The authorities publish the public parts of all the ballots to the bulletin board.

5.2 Voting

The voter receives two ballot parts from each of the voting authorities, one set is used for voting, and the other to audit the authorities. The private parts of the ballot are hidden under a tamper-evident seal (e.g., an opaque envelope). Denote the voter's response to the poll question by $x_v \in \mathbb{Z}_m$. Informally, the voter uses a trivial secret sharing scheme to mask her vote: she splits it into two random shares whose sum is x_v. The second share is chosen ahead of time by A_2, while the first

is selected from the ballot part received from A_1 by choosing the corresponding commitment. A more formal description appears as Protocol 1.

Protocol 1 Ballot casting by voter v

1: Wait to receive ballots parts $B_{\vec{w}}$, for all $\vec{w} \in \{1,2\} \times \{v\} \times \{0,1\}$ from the authorities.
2: Choose a random bit: $b_v \in_R \{0,1\}$
3: Open and publish ballot parts $B_{(1,v,1-b_v)}$ and $B_{(2,v,1-b_v)}$. {these will be used for auditing the voting authorities}
4: Verify that the remaining ballot parts are still sealed, then enter the voting booth with them.
5: Open the ballot parts $B_{(1,v,b_v)}$ snd $B_{(2,v,b_v)}$.
6: Compute $s_v \doteq x_v - t_{(1,v,b_v)} - t_{(2,v,b_v)} \pmod m$. To reduce clutter, below we omit the subscripts b_v and s_v, denoting $c_{(1,v)} \doteq c_{(1,v,b_v),s_v}$, $r_{(1,v)} \doteq r_{(1,v,b_v),s_v}$, $c_{(2,v)} \doteq c_{(2,v,b_v)}$, $r_{(2,v)} \doteq r_{(2,v,b_v)}$ and $t_{(a,v)} \doteq t_{(a,v,b_v)}$. {The computation can be perfomed implictly by the voting mechanism, e.g., the method described in Section 2.2}.
7: Physically erase the private values $t_{\vec{w}}$ from all the received ballot parts. {This step is the "forced ballot erasure"}
8: Leave the voting booth.
9: Publish s_v {recall that $c_{(1,v)}$ and $c_{(2,v)}$ were already published by the authorities}.

5.2.1 Coercion-Resistance Strategy. We assume the adversary cannot observe the voter between steps 4 and 8 of the voting phase (i.e., while the voter is in the voting booth).

If the voter is coerced before step 4, the voter follows the adversary's strategy precisely, but uses random $t_{(a,v)}$ values instead of those revealed on the opened ballots. Because of the forced erasure, the adversary will not be able to tell whether the voter used the correct values or not. By using random values, the end result is that the voter votes randomly (coercing a voter to vote randomly is an attack we explicitly allow).

If the voter is coerced at step 4 or later (after entering the voting booth), she follows the regular voting protocol in steps 4 through 7. Even if she is coerced before step 7, she lies to the adversary and pretends the coercion occurred at step 7 (the adversary cannot tell which step in the protocol the voter is executing while the voter is in the booth). In this case, the adversary cannot give the voter a voting strategy, except one that will invalidate the ballot (since the voter has no more "legal" choices left). The voter must still convince the adversary that her vote was for the "fake input" provided by the adversary rather than her real input. To do this, she pretends the $t_{(2,v)}$ value she received was one that is consistent with the fake input and her real s_v. Using the example in Figure 1, if Sarah was trying to convince a coercer that she actually voted for Jefferson (instead of Adams), she would claim that the upper ballot part had the hole in the leftmost position (rather than the second position), so that her choice on the lower ballot part corresponds to Jefferson.

Note that the adversary can force the voter to cast a random ballot, for example by telling her to always fill the top bubble on the bottom page. However, forcing a random vote is something we explicitly do not prevent.

5.3 Tally

The tally stage is performed by the voting authorities and does not require voter participation (for the intuition behind it, see Section 2.1). Before the start of the tally stage, all authorities know, for every voter v: s_v, $c_{(1,v)}$ and $c_{(2,v)}$ (this was published on the public bulletin board). Each authority A_a also knows the private values $t_{(a,v)}$ and $r_{(a,v)}$ (the voter's choice is $x_v = s_v + t_{(1,v)} + t_{(2,v)} \pmod m$).

For all $1 \leq v \leq n$, denote $d_{3,v} \doteq c_{(1,v)}c_{(2,v)} = C(x_v \pmod m, r_{(1,v)} + r_{(2,v)})$. The value $d_{3,v}$ is a commitment to voter v's choice, up to multiples of m; both the value and the randomness for the commitment are shared among the two authorities (note that $c_{(1,v)} = c_{(1,v,b_v),s_v}$ is a commitment to $x_v - t_{(2,v)}$, while $c_{(2,v)}$ is a commitment to $t_{(2,v)}$, so their product, the homomorphic addition of the committed values, gives us a commitment to x_v as required).

The tally stage uses as subprotocols some zero-knowledge proofs. In particular, we use a proof that one vector of commitments "is a valid shuffle" of another, and a proof that "a committed number is in a specific range". Since we use perfectly-hiding commitments, the "value of a commitment" is not well defined. What we actually use is a *zero-knowledge proof of knowledge*: the prover proves that it "knows" how to open the commitment to a value in the range, or how to shuffle and randomize the first vector of commitments to construct the second. These zero-knowledge protocols are based on standard techniques; simple protocols that meet our requirements appear in Appendix B.

The tally is performed in two phases. The first phase (cf. Protocol 2) is a public "mix-net"-like shuffle of the commitments, while the second is a private protocol between the two authorities, at the end of which the first authority learns how to open the shuffled commitments.

In the first phase, the authorities, in sequence, privately shuffle the vector of committed votes and publish a new vector of commitments in a random order (we denote the new vector published by authority A_a: $d_{a,1}, \ldots, d_{a,n}$). The new commitments are rerandomized (so they cannot be connected to the original voters) by homomorphically adding to each a commitment to a random multiple of m. To prevent A_2 from using the rerandomization step to "tag" commitments and thus gain information about specific voters, A_1's randomization value is taken from a much larger range (A_2 chooses a multiple of m in $\mathbb{Z}_{2^k}$, while A_1 chooses one in $\mathbb{Z}_{2^{2k}}$; we require $|\mathcal{M}| > 2^{2k+2}$ so that the rerandomization doesn't cause the commitment message to "roll over"). Each authority also proves in zero-knowledge that the published commitments are a valid shuffle of the previous set of commitments (where by "valid shuffle", we mean that if it can open one vector of commitments, then it can open the other vector to a permutation of the values). A graphic representation of this phase of the tally appears in Figure 5.

The output of the final shuffle at the end of the first phase is a vector of commitments to the voters' choices that neither of the authorities can connect to the corresponding voters. However, neither of the authorities can open the commitments, since the secret randomness is shared among both of them. In the second

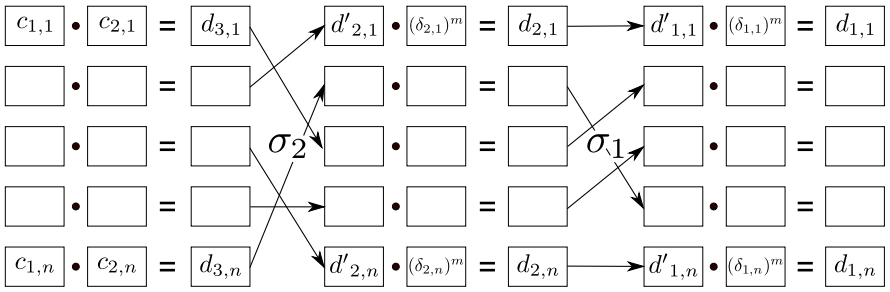

Fig. 5. Tally Phase 1

phase of the tally, the authorities perform the same shuffles on encrypted, rather than committed values (the encryptions all use the public key generated by A_1). At the end of the second phase, A_1 learns the information required to open the final commitment vector, and publishes this, revealing the tally. The communication between the authorities is over an untappable channel between them, so that the public information remains unconditionally private. This phase of the tally is specified by Protocols 3 (for authority A_1) and 4 (for authority A_2).

Protocol 2 Tally Phase 1: Authority A_a

1: Choose a random permutation $\sigma_a \colon [n] \mapsto [n]$,
2: Choose random values $u_{a,1}, \ldots, u_{a,n} \in_R \mathcal{R}$
3: Choose random values $z_{a,1}, \ldots, z_{a,n} \in_R \mathbb{Z}_{2^{(3-a)k}}$
4: If $a = 1$, wait for $d_{2,1}, \ldots, d_{2,n}$ to be published.
5: **for** $1 \leq i \leq n$ **do**
6: Choose a random value $u'_{a,i} \in_R \mathcal{R}$.
7: Publish $d'_{a,i} \doteq d_{a+1,\sigma_a(i)} \cdot C(0, u'_{a,i} - m u'_{a,i})$.
8: Publish $\delta_{a,i} \doteq C(z_{a,i}, u'_{a,i})$
9: Publicly prove in zero-knowledge (using the random beacon) that $\delta_{a,i}$ is a commitment to a value in $\mathbb{Z}_{2^{(3-a)k}}$ (i.e., that $z_{a,i} < 2^{(3-a)k}$)
10: Publicly prove in zero-knowledge (using the random beacon) that $c_{a,i}$ is a commitment to a value in $\mathbb{Z}_{2^{\lceil \log m \rceil}}$
11: Denote $d_{a,i} \doteq d'_{a,i} \delta_{a,i}^m = d_{a+1,\sigma_a(i)} C(z_{a,i} m, u_{a,i})$
12: **end for**
13: Publicly prove in zero-knowledge that $d'_{a,1}, \ldots, d'_{a,n}$ is a valid shuffle of $d_{a+1,1}, \ldots, d_{a+1,n}$ (using the random beacon).

5.4 Universal Verification and Output

The verification can be performed by anyone with access to the public bulletin board. This phase consists of verifiying all the published zero-knowledge proofs by running the zero-knowledge verifier for each proof, using the corresponding output of the random beacon for the random coins. The verifier then computes and outputs the final tally. The verification protocol appears as Protocol 5.

Protocol 3 Tally Phase 2: Authority A_1

1: Wait for Tally Phase 1 to terminate.
2: Verify all the published zero-knowledge proofs of shuffling.
3: **for** $1 \leq i \leq n$ **do**
4: Send to Authority A_2: $e_{1,i}^{(\mathcal{M})} \doteq E^{(\mathcal{M})}(s_i + t_{(1,i)}) = E^{(\mathcal{M})}(x_i - t_{(2,i)})$
5: Send to Authority A_2: $e_{1,i}^{(\mathcal{R})} \doteq E^{(\mathcal{R})}(r_{(1,i)})$
6: **end for**
7: Prove in (interactive) zero-knowledge to A_2 that $e_{1,1}^{(\mathcal{M})}, \ldots, e_{1,n}^{(\mathcal{M})}$ and $e_{1,1}^{(\mathcal{R})}, \ldots, e_{1,n}^{(\mathcal{R})}$ are encryptions of the message (resp. randomness) corresponding to $c_{(1,1)}, \ldots, c_{(1,n)}$.
8: Wait to receive $e_{2,1}^{(\mathcal{M})}, \ldots, e_{2,n}^{(\mathcal{M})}$ and $e_{2,1}^{(\mathcal{R})}, \ldots, e_{2,n}^{(\mathcal{R})}$ over the untappable channel from A_2.
9: **for** $1 \leq i \leq n$ **do**
10: Compute $\xi_i \doteq D^{(\mathcal{M})}(e_{2,\sigma_1(i)}^{(\mathcal{M})}) + z_{1,i}m$
11: Compute $\rho_i \doteq D^{(\mathcal{R})}(e_{2,\sigma_1(i)}^{(\mathcal{R})}) + u_{1,i}$
12: Verify that $d_{1,i} = C(\xi_i, \rho_i)$
13: **end for**
14: Publish $\xi_1, \ldots, \xi_n$ and $\rho_1, \ldots, \rho_n$.

Protocol 4 Tally Phase 2: Authority A_2

1: Wait for Tally Phase 1 to complete.
2: Verify all zero-knowledge proofs published in phase 1.
3: Wait to receive $e_{1,1}^{(\mathcal{M})}, \ldots, e_{1,n}^{(\mathcal{M})}$ and $e_{1,1}^{(\mathcal{R})}, \ldots, e_{1,n}^{(\mathcal{R})}$ over the untappable channel from A_1.
4: Verify the zero-knowledge proof that the encryptions correspond to the committed values and randomness
5: **for** $1 \leq i \leq n$ **do**
6: Send to Authority A_1: $e_{2,i}^{(\mathcal{M})} \doteq e_{1,\sigma_2(i)}^{(\mathcal{M})} E^{(\mathcal{M})}(t_{(2,\sigma_2(i))} + z_{2,i}m) = E^{(\mathcal{M})}(x_{\sigma_2(i)} + z_{2,i}m)$
7: Send to Authority A_1: $e_{2,i}^{(\mathcal{R})} \doteq e_{1,\sigma_2(i)}^{(\mathcal{R})} E^{(\mathcal{R})}(r_{(2,\sigma_2(i))} + u_{2,i})$
8: **end for**

5.5 Security Guarantees

We give two different formal security guarantees for this protocol, formally specified by the theorems below. The first is a guarantee for privacy and accuracy of the tally (Theorem 5.1), and the second a guarantee against vote-buying and coercion (Theorem 5.2).

THEOREM 5.1. *The Split-Ballot Voting Protocol UC-realizes functionality $\mathcal{F}_{\text{vote}}$, for an adversary that is fully adaptive up to the end of the voting phase, but then statically decides which of the voting authorities to corrupt (it can still adaptively corrupt voters).*

The reason for the restriction on the adversary's adaptiveness is that the homomorphic encryption scheme we use is *committing*.

Protocol 5 Verification

1: Verify the proof that the commitment key was generated correctly {the proof generated in step 1, Section 5.1}
2: **for** $1 \leq i \leq n$ **do** {Verify opened ballots chosen for audit}
3: Verify for all $j \in \mathbb{Z}_m$ that $c_{(1,i,1-b_i),j} = C(t_{(1,i,1-b_i)} + j \pmod m, r_{(1,i,1-b_i),j})$
4: Verify that $c_{(2,i,1-b_i)} = C(t_{(2,i,1-b_i)}, r_{(2,i,1-b_i)})$
5: **end for**
6: Verify the shuffle proofs (produced in step 13 of Protocol 2)
7: **for** $1 \leq i \leq n$ **do** {Verify opening of final, shuffled commitments}
8: Verify that $d_{1,i} = C(\xi_i, \rho_i)$
9: **end for**
10: **for** $i \in \mathbb{Z}_m$ **do** {Compute and output the tally}
11: Compute $\tau_i \doteq |\{j \in [n] \mid \xi_j \equiv i \pmod m\}|$
12: Output τ_i {The tally for candidate i}
13: **end for**

Note that this limitation on adaptiveness only holds with respect to the *privacy* of the votes *under composition*, since an adversary whose only goal is to change the final tally can only gain by corrupting both voting authorities at the beginning of the protocol.

The proof of Theorem 5.1 appears in Section 6

THEOREM 5.2. *The Split-Ballot voting protocol is receipt-free, for any adversary that does not corrupt any of the voting authorities.*

The formal proof of this theorem appears in Section 7. The intuition behind it is apparent from the coercion-resistance strategy (described in Section 5.2).

6. PROOF OF ACCURACY AND PRIVACY GUARANTEE (THEOREM 5.1)

In order to prove the security of our protocol in the UC model, we must show that there exists a simulator $\mathcal{I}$ (the "ideal adversary") that internally runs a black-box straight-line (without rewinding) simulation of the real-world adversary, $\mathcal{A}$, simulating its view of the honest parties and the functionalities used by the real protocol. At the same time, the simulator interacts with the ideal voting functionality to allow it to keep the real-world adversary's simulated view consistent with the input and output of the parties in the ideal world. The protocol UC-realizes the ideal functionality if no environment machine (which sets the inputs for the parties and controls the real-world adversary) can distinguish between a real execution of the protocol and a simulated execution in the ideal world.

The general idea of the simulation is that $\mathcal{I}$ creates, internally, a "provisional view" of the world for all honest parties. Throughout the simulation, it updates this view so that it is consistent with the information $\mathcal{I}$ learns. At any point, if the adversary corrupts a party then $\mathcal{I}$ also corrupts that party (learning its input, and updating the provisional view to be consistent with that information). If the adversary corrupts *both* of the election authorities, A_1 and A_2, $\mathcal{I}$ sends the **RevealVotes** command to $\mathcal{F}_{\text{vote}}$ (and learns the votes of all honest parties). Because the simulator can choose the commitment key in such a way that commitments

are equivocable, it can make sure the "visible" parts of the provisional view (those that can be seen by the adversary) are also consistent with what the environment knows at all times (by changing retroactively the contents of the commitments). Thus, $\mathcal{I}$ is able to simulate the honest parties exactly according to the real protocol, creating a view that is statistically close to the view in the real world.

There are four main sticking points in this approach:

(1) The adversary can prepare "bad ballots" in the setup phase (which it does not know how to open correctly). Since the commitment is perfectly hiding, the simulator cannot tell at that point which of the ballots is bad. We deal with this by allowing $\mathcal{I}$ to change the tally by a small number of votes (the number depends on the security parameter). The idea is that there are only two ways for the adversary to change the tally: either it can equivocate on commitments (in this case, we can use the environment/adversary pair to break the commitment scheme), or it prepares bad ballots. If it prepares many bad ballots, it will be caught with high probability, since $\mathcal{I}$ simulates the honest voters correctly, and they choose to audit a bad ballot pair with probability $\frac{1}{2}$. So if we simply ignore the cases in which it was not caught, the distributions of the environment's view in the real and ideal world will still be statistically close.
(2) Unlike perfectly-hiding commitments, the encryptions send in the second tally phase cannot be retroactively changed in light of new information. This problem is solved by the restriction on the adaptiveness of the adversary (either it doesn't get to see the encryptions at all before it sees their contents, or it doesn't get to see the contents of the encryptions).
(3) The revealed values of the shuffled commitments aren't exactly the voters' choices — they are the voters' choices only up to multiples of m. For example, If $x_v = 0$, this value can be shared between the authorities as $0, 0$ (in which case the commitment $d_{3,v} = 0$), but also as $m-1, 1$, (in which case the commitment $d_{3,v} = m$). Since each authority knows its share of x_v, the revealed commitment can leak information about a specific voter. To prevent this, we would like to "rerandomize" the value by adding a uniformly random multiple of m. However, since it is likely that $m \nmid |\mathcal{M}|$, adding large multiples of m could change the value. Instead, authority A_2 (who shuffles first) adds multiples of m from a large range, but one that is guaranteed not to "overflow". To prevent A_2 from using the randomization step itself to gain information about specific voters, A_1 does the same thing with an even larger range. Thus, the revealed tallies leak only a negligible amount of information about the specific voters.
(4) Finally, we use the semantic security of the encryption scheme to show that the environment/adversary pair cannot use the encryptions sent in the tally phase to gain any advantage in differentiating the real and ideal worlds. If it can, we can use them to break the encryption scheme.

Below we describe the protocol for $\mathcal{I}$. In order to make the proof readable, we do not formally specify the entire simulation protocol. Instead, we focus on the points where $\mathcal{I}$ cannot simply follow the real protocol for its simulated parties (either because it lacks information the real parties have, or because $\mathcal{A}$ deviated from the protocol). We also omit the global mechanics of the simulation: whenever $\mathcal{A}$ corrupts a party, $\mathcal{I}$ also corrupts the corresponding ideal party; whenever $\mathcal{A}$

instructs a corrupted party to output a message, $\mathcal{I}$ instructs the corresponding ideal party to output the message as well. If $\mathcal{A}$ deviates from the protocol in a way that is evident to honest real parties (e.g., refuses to send messages, or sends syntactically incorrect messages to honest parties), $\mathcal{I}$ halts the simulation (and the parties output $\perp$). An exception is when corrupt voters deviate from the protocol in this way: in this case the voter would be ignored by honest authorities (rather than stopping the election), and $\mathcal{I}$ forces an abstention for that voter. Except for the noted cases, we explictly specify when $\mathcal{I}$ sends commands to $\mathcal{F}_{\text{vote}}$; the simulation is self-contained and only visible to $\mathcal{A}$ and the environment through the interactions of the simulated parties with the adversary.

6.1 Setup Phase

$\mathcal{I}$ chooses a random seed $s \in \{0,1\}^{\ell'}$ and simulates the random beacon, using $K'(s)$ to generate the part of the beacon used as input to the commitment-key generation algorithm, K (this will allow $\mathcal{I}$ to equivocate commitments). It then runs the ballot preparation stage by simulating A_1 and A_2 exactly according to protocol (or according to the real-world adversary's instructions, if one or both are corrupted). At the end of this stage, the adversary is committed to the contents of any ballots it sent.

6.2 Voting Phase

Honest Voter. When it receives a "v has voted" message from the voting functionality for an honest voter v, $\mathcal{I}$ begins simulating voter v running Protocol 1. The actions of $\mathcal{I}$ depend on whether the adversary has corrupted both A_1 and A_2:

Case 1: If both A_1 and A_2 are corrupt, $\mathcal{I}$ learns x_v from the voting functionality. It can then exactly simulate an honest voter voting x_v

Case 2: If at least one of the authorities is honest, $\mathcal{I}$ simulates a voter using a random value x'_v for the voter's choice value. This involves choosing a random commitment from the set received from A_1.

Corrupt Voter. Throughout this phase, $\mathcal{I}$ simulates a corrupt voter v by following $\mathcal{A}$'s instructions exactly. If both authorities were honest at the beginning of the voting stage (in particular, if the ballots were generated by $\mathcal{I}$), $\mathcal{I}$ computes $x_v = s_v + t_{(1,v)} + t_{(2,v)}$ and sends a **Vote** (v, x_v) command to $\mathcal{F}_{\text{vote}}$. If at least one authority was corrupt at the beginning ot the voting stage, $\mathcal{I}$ sends nothing to the ideal functionality (but in this case it will be able to cast a vote in the tally phase). Denote by W the number of voters for which $\mathcal{I}$ did *not* cast a vote during the voting phase.

Voter Corrupted During Protocol. If a voter v is honest at the beginning of the phase, and is corrupted during the protocol, $\mathcal{I}$ learns the real choice x_v when the voter is corrupted. If both authorities were already corrupt, $\mathcal{I}$ already knows x_v so this information will not have changed any of the views in the simulation. If at least one authority was honest and $x'_v \neq x_v$, $\mathcal{I}$ has to rewrite its history (both that of the honest authority and that of voter v) to be consistent with the new information. In this case, it uses its ability to equivocate commitments to rewrite the value of $t_{(a,v)}$, where a is the index of an honest authority (this will also change the randomness of the commitments). The new value will satisfy $x_v = s_v + t_{(1,v)} + t_{(2,v)}$.

Authority Corrupted During Protocol. If, at the beginning of the voting phase, both authorities are honest and later one is corrupted, $\mathcal{I}$ does not need to rewrite its history, since it is consistent with the real world simulation.

If, at the beginning the voting phase, at least one authority is honest and sometime later both of the authorities become corrupted, $\mathcal{I}$ learns the choices of all previous voters at that point. In this case, when the last authority is corrupted, $\mathcal{I}$ may be forced to rewrite its history as well as that of the honest voters whose choices didn't match the guesses made by $\mathcal{I}$. It does this by using its ability to equivocate commitments in order to change only the private parts of the ballots which were not seen by $\mathcal{A}$ before the corruption.

Case 1: If A_1 is corrupted last, $\mathcal{I}$ can rewrite the value $t_{(1,v)}$ to match the voter's choice by equivocating on the commitments $c_{(1,v),1}, \ldots, c_{(1,v),m}$

Case 2: If A_2 is corrupted last, $\mathcal{I}$ can rewrite the value $t_{(2,v)}$ to match the voter's choice by equivocating on the commitment $c_{(2,v)}$

6.3 Tally Phase

This phase begins when a voting authority sends the **Tally** command to $\mathcal{F}_{\text{vote}}$, or the adversary has corrupted an authority and decides to end the voting phase. $\mathcal{I}$ waits for $\mathcal{F}_{\text{vote}}$ to announce the tally. The simulator's strategy now depends on which of the voting authorities is corrupt (note that from this stage on the adversary is static with regard to corrupting the voting authorities). We can assume w.l.o.g. that $\mathcal{A}$ corrupted the relevant authorities at the start of the Setup phase: if only one authority is corrupted during the protocol, $\mathcal{I}$'s simulation is identical to the case where the authority was corrupt from the start, but chose to follow the protocol correctly. If both authorities are corrupted, $\mathcal{I}$ was required to rewrite its view at the time of the second corruption; however, the rewritten history is identical to a simulation in which both authorities were corrupted from the start and chose to follow the protocol honestly.

Case 1: **Neither voting authority is corrupt.** $\mathcal{I}$ generates vectors of random commitments for $d_{1,1}, \ldots, d_{1,n}$, $d_{2,1}, \ldots, d_{2,n}$, $d'_{1,1}, \ldots, d'_{1,n}$ and $d'_{2,1}, \ldots, d'_{2,n}$. Using its ability to equivocate, $\mathcal{I}$ can open the commitments to any value. In particular, it can pass all the zero-knowledge proofs. and open the commitments $d_{1,1}, \ldots, d_{1,n}$ to values that match the announced tally and are identically distributed to the outcome in the real world protocol.

Case 2: **Exactly one authority is corrupt.** $\mathcal{I}$ rewrites its provisional view for the honest authority to make it consistent with the announced tally, using a random permutation for the honest voters. It then simulates the honest authority according to protocol, using its provisional view. At the end of Protocol 3, $\mathcal{I}$ simulates the verifier running Protocol 5. If verification fails, $\mathcal{I}$ halts in the ideal world as well. Otherwise, $\mathcal{I}$ now knows the real-world tally: $\tau_0, \ldots, \tau_{m-1}$. This may be different from the ideal-world tally announced by $\mathcal{F}_{\text{vote}}$; If the ideal-world tally can be changed to the real-world one by adding W votes and changing k votes (where W is the number of corrupt voters for whom $\mathcal{I}$ did not cast a vote in the voting phase), $\mathcal{I}$ sends the appropriate **Vote** commands to $\mathcal{F}_{\text{vote}}$, then sends an **Announce** command with the updated tally. Otherwise, $\mathcal{I}$ outputs "tally failure" and halts. In this case the real and ideal worlds are

very different; however, we prove that this happens with negligible probability (see Claim 6.3).

If an honest voter v is corrupted during the tally phase, $\mathcal{I}$ learns x_v and must provide the voter's provisional history to $\mathcal{A}$. If $x'_v \neq x_v$, $\mathcal{I}$ chooses one of the remaining honest voters v' for which $x'_{v'} = x_v$ and rewrites both voters' views by equivocating on the commitments generated by the honest authority. In the new views $x'_{v'} \leftarrow x'_v$ and $x'_v \leftarrow x_v$. Note that there will always be such an honest voter, since the provisional views of the simulated honest voters are consistent with the ideal-world tally, which is consistent with the inputs of the ideal voters.

Case 3: **Both authorities are corrupt.** In this case, $\mathcal{I}$ knows the inputs of all voters, so it can create a completely consistent view. $\mathcal{I}$ follows the instructions of the real-world adversary until $\xi_1, \ldots, \xi_n$ and $\rho_1, \ldots, \rho_n$ are published. It then simulates the verifier running Protocol 5. If verification fails, $\mathcal{I}$ halts in the ideal world as well. Otherwise, $\mathcal{I}$ now knows the real-world tally: $\tau_0, \ldots, \tau_{m-1}$. As in the case of a single corrupt authority, if the ideal-world tally can be changed to the real-world one by adding W votes and changing k votes, $\mathcal{I}$ sends the appropriate **Vote** commands to $\mathcal{F}_{\text{vote}}$, then sends a **Announce** command with the real-world tally. Otherwise, $\mathcal{I}$ outputs "tally failure" and halts.

6.4 Indistinguishability of the Real and Ideal Worlds

To complete the proof of Theorem 5.1, we must show that the environment machine cannot distinguish between the real protocol executed in the real world, and $\mathcal{I}$'s simulation executed in the ideal world. This is equivalent to showing that the views of the environment in both cases are computationally indistinguishable.

To define the environment's view, it is helpful to explicitly describe the views of all the parties in the real world:

Verifier: The verifier's view consists of all the public information (and only that):

(1) The output of the random beacon: R.
(2) The commitment public key, $cpk = K(R)$, generated by A_1 in the Setup phase
(3) The proof of correctness for the commitment public key (the output of P_K)
(4) The audit bits of the voters: b_v for all $v \in [n]$.
(5) The public and private parts of all the audit ballots: $B_{\vec{w}}$ for all $\vec{w} = (a, v, i)$ such that $i = 1 - b_v$.
(6) Randomness for all the audit ballots: $r_{\vec{w}}$ for all $\vec{w} = (2, v, i)$ such that $i = 1 - b_v$ and $r_{\vec{w},j}$ for all $v \in [n]$, $j \in \mathbb{Z}_m$ and $\vec{w} = (1, v, 1 - b_v)$.
(7) Public part of the ballots for cast ballots: s_v,$c_{(1,v),j}$ and $c_{(2,v)}$ for all $v \in [n]$ and $j \in \mathbb{Z}_m$.
(8) Public proofs of tally correctness: $d'_{a,1}, \ldots, d'_{a,n}$, $\delta_{a,1}, \ldots, \delta_{a,n}$ for $a \in \{1, 2\}$ and the transcripts of the zero-knowledge proofs that these were generated correctly.
(9) The Opening of the shuffled commitments: $\xi_1, \ldots, \xi_n$ and $\rho_1, \ldots, \rho_n$.

Voter v: The voter's view includes all the public information (the verifier's view) in addition to:

(1) The voter's input: x_v

(2) The private part of the voter's ballots: $t_{(1,v)}$ and $t_{(2,v)}$

Authority A_1: The view of this authority consists of the verifier's view, and in addition:
(1) The public keys for the encryption schemes: $pk^{(\mathcal{M})}$ and $pk^{(\mathcal{R})}$.
(2) The secret keys for the encryption schemes: $sk^{(\mathcal{M})}$ and $sk^{(\mathcal{R})}$.
(3) The private parts of the voters' ballots: $t_{(1,v)}$ for all $v \in [n]$
(4) The randomness for the commitments in the voters' ballots: $r_{\vec{w},j}$ for all $v \in [n]$, $j \in \mathbb{Z}_m$ and $\vec{w} = (1, v, b_v)$
(5) The permutation σ_1 and the values $u_{1,1}, \ldots, u_{1,n}$, $u'_{1,1}, \ldots, u'_{1,n}$ and $z_{1,1}, \ldots, z_{1,n}$
(6) The secret random coins for the encryptions sent to A_2.
(7) The encryptions received from A_2: $e^{(\mathcal{M})}_{2,1}, \ldots, e^{(\mathcal{M})}_{2,n}$ and $e^{(\mathcal{R})}_{2,1}, \ldots, e^{(\mathcal{R})}_{2,n}$ and their contents.
(8) The secret random coins used in the zero-knowledge proofs in which A_1 was the prover.

Authority A_2: The view of this authority consists of the verifier's view, and in addition:
(1) The public keys for the encryption schemes: $pk^{(\mathcal{M})}$ and $pk^{(\mathcal{R})}$
(2) The private parts of the voters' ballots: $t_{(2,v)}$ for all $v \in [n]$
(3) The randomness for the commitments in the voters' ballots: $r_{\vec{w}}$ for all $v \in [n]$ and $\vec{w} = (2, v, b_v)$
(4) The permutation σ_2 and the values $u_{2,1}, \ldots, u_{2,n}$, $u'_{2,1}, \ldots, u'_{2,n}$ and $z_{2,1}, \ldots, z_{2,n}$
(5) The encryptions received from A_1: $e^{(\mathcal{M})}_{1,1}, \ldots, e^{(\mathcal{M})}_{1,n}$ and $e^{(\mathcal{R})}_{1,1}, \ldots, e^{(\mathcal{R})}_{1,n}$.
(6) The secret random coins for the encryptions sent to A_1.
(7) The secret random coins used in the zero-knowledge proofs in which A_2 was the prover.

$\mathcal{A}$: The real-world adversary's view consists of the verifier's view, any messages it receives from the environment and the view of any party it corrupts.

Environment: The environment's view consists of $\mathcal{A}$'s view, and in addition, the inputs of all the voters: $x_1, \ldots, x_n$ and its own random coins.

In the ideal world, the environment's view is the same, except that the views of the real-world parties are those simulated by $\mathcal{I}$. The other difference is that in the ideal world, the output of the verifier seen by the environment is the tally produced by $\mathcal{F}_{\text{vote}}$ (none of the other parties have output).

The following lemma completes the proof of Theorem 5.1:

LEMMA 6.1. *The enviroment's view in the real world is computationally indistinguishable from its view in the ideal world.*

PROOF. *Setup and Voting Phases.* First, note that in the Setup and Voting phases of the protocol, as long as $\mathcal{I}$ can perfectly equivocate commitments, the views are identically distributed. This is because the views of the simulated parties are always kept in a perfectly consistent state, given the knowledge $\mathcal{I}$ has about the voters' inputs. Whenever a simulated party is corrupted, $\mathcal{I}$ gains enough information to perfectly rewrite its view so that it is consistent with the previous view of the environment.

Thus, the only possible way the views could differ is if $\mathcal{I}$ cannot equivocate commitments. This can occur only if the commitment key published in the Setup

phase is not the one chosen by $\mathcal{I}$. But by the definition of the commitment scheme, the probability that A_1 can generate a key that passes verification but does not allow equivocation is negligible.

Tally Phase. In the Tally phase, $\mathcal{I}$ deviates from a perfect simulation only when at least one of the authorities is corrupt. The possible reasons $\mathcal{I}$'s simulation may be imperfect are:

(1) The **Vote** commands sent to $\mathcal{F}_{\text{vote}}$ by the simulator are not consistent with the tally in the simulation. This would cause the output of the verifier in the ideal world to differ from its output in the real world. Note that the inconsistency will be noticable only if the ideal-world tally is "far" from the real-world tally: If it can be modified by adding W arbitrary votes and changing up to k votes, $\mathcal{I}$ will be able to "fix" the ideal-world tally. The probability that $\mathcal{I}$ fails in this way is negligible, as we prove in Claim 6.3.

(2) If A_1 is honest and A_2 is corrupt:
   (a) The encryptions $e_{1,1}^{(\mathcal{M})}, \ldots, e_{1,n}^{(\mathcal{M})}$ and $e_{1,1}^{(\mathcal{R})}, \ldots, e_{1,n}^{(\mathcal{R})}$ may be inconsistent with the environment's view in the ideal world: the encrypted values contain $\mathcal{I}$'s simulated inputs for the honest voters (which may differ from the real inputs of the voters). The environment, however, knows the real inputs of the voters. Although the environment's views may be statistically far, the semantic security of the encryption scheme implies that the encryptions of two different messages are computationally indistinguishable, even when the messages are known. Hence, the environment's views of the encryptions in the real and ideal world are computationally indistinguishable.
   (b) The distribution of $\xi_1 \ldots, \xi_n$ may be inconsistent with the environment's view in the ideal world. This can happen because $\xi_1, \ldots, \xi_n$ is a permutation of $x_1, \ldots, x_n$ only modulo m; the actual values depend on the permutations σ_1, σ_2 the values $t_{(1,1)}, \ldots, t_{(1,n)}$, $t_{(2,1)}, \ldots, t_{(2,n)}$, $z_{(1,1)}, \ldots, z_{(1,n)}$, $z_{(2,1)}, \ldots, z_{(2,n)}$ and on the order of the simulated votes chosen by $\mathcal{I}$ (which may differ from the actual order). However, the statistical distance between the views of $\xi_1, \ldots, \xi_n$ in the real and ideal worlds is negligible. Intuitively, this is because A_2 can only add multiples of m up to 2^k. No matter what it does, once A_1 adds a multiple of m uniformly chosen up to 2^{2k}, the output distribution will be nearly uniform (only an exponentially small fraction of the output values will have different probabilities). We prove this formally in Claim 6.2.

(3) If A_1 is corrupt and A_2 is honest, the distribution of $\xi_1 \ldots, \xi_n$ may be inconsistent with the environment's view in the ideal world. This can happen for exactly the same reason as it does when A_2 is corrupt and A_1 honest: the extra multiples of m in $\xi_1 \ldots, \xi_n$ may contain information about the order of the votes. As in the previous case, the statistical distance between the views of $\xi_1, \ldots, \xi_n$ in the real and ideal worlds is negligible (here the intuition is that A_1 can only set t_v to be at most $2m$, while A_2 will add a multiple of m uniformly chosen up to 2^k. We omit the formal proof, as it is almost identical to the proof of Claim 6.2.

Verification Phase. Since the verifier cannot be corrupted and uses only public information, the views of the environment in the verification phase are identical in

the real and ideal world, except in case of a tally failure (which we prove occurs with negligible probability). □

The proof of Lemma 6.1 is completed by the claims below:

CLAIM 6.2. *Fix any permutation σ_2 and values $t_{(1,1)},\ldots,t_{(1,n)} \in \mathbb{Z}_{2m}$, $t_{(2,1)},\ldots,t_{(2,n)} \in \mathbb{Z}_{2m}$, $z_{(2,1)},\ldots,z_{(2,n)} \in \mathbb{Z}_{2^k}$ and $x_1,\ldots,x_n \in \mathbb{Z}_m$. For a permutation π, let $\Xi_\pi = \xi_1\ldots,\xi_n$ be the output of the protocol when it is run with the fixed values, permuted voter inputs $x_{\pi(1)},\ldots,x_{\pi(n)}$ and when σ_1, $z_{(1,1)},\ldots,z_{(1,n)}$ are chosen randomly (following the protocol specification).*

Then for any two permutations π_1 and π_2, the statistical difference between Ξ_{π_1} and Ξ_{π_2} is at most $n2^{-k+1}$.

PROOF. Note that $\xi_i = x_{\pi(\sigma_2(\sigma_1(i)))} + (f_{\sigma_2(\sigma_1(i))} + z_{2,\sigma_1(i)} + z_{1,i})m$, where $f_i \in \mathbb{Z}_4$. First, we define a new random variable ${\Xi'}_\pi = ({\xi'}_1\ldots,{\xi'}_n)$, whose value is similar to Ξ_π, except without the fixed multiples of m: ${\xi'}_i = x_{\pi(\sigma_2(\sigma_1(i)))} + z_{1,i}m$. Note that $\pi\circ\sigma_2\circ\sigma_1$ is a random permution for any fixed π and σ_2, and $\{z_{1,i}\}$ are identically and independently distributed. Hence, for any two permutations π_1 and π_2, ${\Xi'}_{\pi_1}$ and ${\Xi'}_{\pi_2}$ are identically distributed. Next, we will show that for any permutation π, the statistical difference between ${\Xi'}_\pi$ and Ξ_π is at most $n2^{-k}$. Thus, by the triangle inequality, the statistical difference between Ξ_{π_1} and Ξ_{π_2} is at most $n2^{-k+1}$. Denote $\vec{\xi} \doteq (\xi_1\ldots,\xi_n)$. By definition, the statistical difference between the two distributions is:

$$\begin{aligned}
\Delta({\Xi'}_\pi,\Xi_\pi) &= \frac{1}{2}\sum_{\vec{\xi}}\left|\Pr[{\Xi'}_\pi=\vec{\xi}]-\Pr[\Xi_\pi=\vec{\xi}]\right| \\
&= \frac{1}{2n!}\sum_{\vec{\xi}}\left|\sum_{\sigma_1}\left[\Pr[{\Xi'}_\pi=\vec{\xi}\mid\sigma_1]-\Pr[\Xi_\pi=\vec{\xi}\mid\sigma_1]\right]\right| \\
&\le \frac{1}{2n!}\sum_{\vec{\xi},\sigma_1}\left|\Pr[{\Xi'}_\pi=\vec{\xi}\mid\sigma_1]-\Pr[\Xi_\pi=\vec{\xi}\mid\sigma_1]\right|.
\end{aligned}$$

Denote $p \doteq \Pr[\Xi_\pi=\vec{\xi}\mid\sigma_1]$ and $p' \doteq \Pr[{\Xi'}_\pi=\vec{\xi}\mid\sigma_1]$. Let $\zeta_i \doteq f_{\sigma_2(\sigma_1(i))} + z_{2,\sigma_1(i)}$ (the fixed multiples of m) and $\theta_i \doteq x_{\pi(\sigma_2(\sigma_1(i)))}$ (the permuted inputs). Note that since $f_{\sigma_2(\sigma_1(i))} < 4 \le 2^k$ we have $0 \le \zeta_i \le 2^{k+1}$. By our definition of p:

$$p = \Pr[\bigwedge_{i=1}^{n}(\theta_i + \zeta_i m + z_{1,i}m = \xi_i) \mid \sigma_1]$$

(where the probability is only over $\{z_{1,i}\}$). Hence, $p \neq 0$ only if for all $1 \le i \le n$, it holds that $\theta_i \equiv \xi_i \pmod m$ and $z_{1,i} = \frac{1}{m}(\xi_i - \theta_i) - \zeta_i$.

Since $z_{1,i}$ is uniformly chosen in $\mathbb{Z}_{2^{2k}}$ (and in particular $0 \le z_{1,i} \le 2^{2k}$), it follows that $p \neq 0$ only if for all $1 \le i \le n$:

$$\zeta_i \le \frac{1}{m}(\xi_i - \theta_i) \le 2^{2k} + \zeta_i.$$

When $p > 0$, then $p = 2^{-2kn}$ (since for every i there is exactly one "good" choice for $z_{1,i}$). Similarly, $p' \neq 0$ only if for all $1 \le i \le n$, it holds that $\theta_i \equiv \xi_i \pmod m$

and $\frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k}$. Hence, if $p \neq p'$ then for all $1 \leq i \leq n$: $\theta_i \equiv \xi_i \pmod m$ and either

Case 1: $p \neq 0, p' = 0$**:** for all $1 \leq i \leq n$, $\zeta_i \leq \frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k} + \zeta_i$ and there exists i such that $2^{2k} < \frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k} + \zeta_i$ or

Case 2: $p = 0, p' \neq 0$**:** for all $1 \leq i \leq n$, $\frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k}$ and there exists i such that $\frac{1}{m}(\xi_i - \theta_i) \leq \zeta_i$.

Note that both p and p' depend only on $\vec{\xi}$ (and not on σ_1). So we can write

$$\Delta(\Xi'_\pi, \Xi_\pi) \leq \frac{1}{2} \sum_{\vec{\xi}} |p(\vec{\xi}) - p'(\vec{\xi})| = 2^{-2kn-1} \left| \left\{ \vec{\xi} \mid p(\vec{\xi}) \neq p'(\vec{\xi}) \right\} \right| \tag{1}$$

We can list all such $\vec{\xi}$ (possibly overcounting) in the following way:

(1) Choose $i \in [n]$ (n possibilities)
(2) Choose one of the two cases above for which $p \neq p'$ (2 possibilities):
(3) Depending on which is chosen, either:
Case 1: choose ξ_i such that $2^{2k} + \zeta_i \geq \frac{1}{m}(\xi_i - \theta_i) > 2^{2k}$ or
Case 2: choose ξ_i such that $\frac{1}{m}(\xi_i - \theta_i) \leq \zeta_i$
(in either case, since $m \geq 2$, there are $\frac{1}{m}\zeta_i \leq 2^k$ possibilities)
(4) For all $1 \leq j \leq n, j \neq i$:
Case 1: choose ξ_j such that $\zeta_j \leq \frac{1}{m}(\xi_j - \theta_j) \leq 2^{2k} + \zeta_j$ or
Case 2: choose ξ_j such that $\frac{1}{m}(\xi_j - \theta_j) \leq 2^{2k}$
(in either case, there are $\prod_j \frac{1}{m} 2^{2k} \leq 2^{2(k-1)(n-1)} \leq 2^{2kn-k}$ possibilities)

Thus, the number of $\vec{\xi}$s that satisfy $p(\vec{\xi}) \neq p'(\vec{\xi})$ is bounded by $n2^{2kn-k+1}$. Plugging this in to inequality 1, we get $\Delta(\Xi'_\pi, \Xi_\pi) \leq 2^{-2kn-1} \cdot nn2^{2kn-k+1} \leq n2^{-k}$. □

Claim 6.3. *The probability that the ideal simulator terminates the simulation by outputting "tally failure" is a negligible function of k.*

Proof. $\mathcal{I}$ outputs "tally failure" only if the simulated verifier did not abort, and the ideal tally (computed by $\mathcal{F}_{\text{vote}}$) cannot be changed to the real tally (the one output by the simulated verifier) by adding W votes and changing k votes (where $W = 0$ if both authorities were honest at the beginning of the voting phase, and otherwise W is the number of voters who were corrupt during the voting phase).

Assume, in contradiction, that the environment/real-world adversary pair can cause a tally failure with probability ϵ. We will show how to use such an environment/adversary pair to break the binding property of the commitment scheme.

That is, there exists a machine M that can produce, with probability polynomial in ϵ, a commitment c and m_1, r_1, m_2, r_2, $m_1 \neq m_2$ such that $c = C(m_1, r_1) = C(m_2, r_2)$.

M works by simulating the entire ideal world (including the enviroment machine, $\mathcal{F}_{\text{vote}}$ and the ideal-world adversary, $\mathcal{I}$). After each of the public zero-knowledge proofs of knowledge in the Tally phase (steps 9, 10 and 13 in Protocol 2), M rewinds the environment/real-adversary pair and runs their corresponding knowledge extractors. Let ϵ' be the probability that the knowledge extractors succeed for all the proofs. Since each knowledge extractor succeeds with probability polynomial in the probability that the verifier accepts, and the verifier accepts all the

proofs with probability at least ϵ (otherwise it would not cause a tally failure), $\epsilon' = poly(\epsilon)$.

The proofs of knowledge allow M to extract the permutations for the shuffles performed by the authorities, along with the randomizing values (and ensure that the extracted values are in the correct ranges). Since the final opened commitments are consistent with the real tally, the extracted value of the original commitment vector (before both shuffles and randomization) must also be consistent with the real tally. Let $(x''_v, (r'')_v)$ denote the opening of the commitment $d_{3,v}$ that is output by M.

For any execution of the protocol in the ideal world, the ideal tally consists of the tally of all voters (if both authorities were honest at the beginning of the Voting phase) or only of the honest voters (otherwise) This is because $\mathcal{I}$ does not send a **Vote** command on behalf of corrupt voters if one of the authorities was corrupt at the beginning of the Voting phase). A tally failure means that even after adding W votes to the ideal tally, it differs from the real tally by more than k votes.

Case 1: **All the ballots were generated by honest authorities.** Since the honest authorities are simulated by $\mathcal{I}$ (which is, in turn, simulated by M), this means all the ballots were generated by M. In particular, there exists a voter v such that $x_v \neq x''_v$, and for which M knows $r_{(1,v)}, r_{(2,v)}$. Since $d_{3,v} = C(x_v, r_{(1,v)} + r_{(2,v)})$ by the construction of d_3, and $d_{3,v} = C(x''_v, r''_v)$ by the knowledge extraction, M can open $d_{3,v}$ in two different ways.

Case 2: **At least some of the ballots were generated by a corrupt authority.** In this case, there must still be at least k honest voters $v_1, \ldots, v_k$ such that $x_{v_i} \neq x''_{v_i}$. (since W is the number of corrupt voters). To break the commitment scheme, M rewinds the ideal-world to the end of the setup phase (after all the ballots have been committed), then reruns the simulation with new randomness for $\mathcal{I}$. In particular, the honest voters' audit bits are new random bits. The probability that both simulations end with tally failures is at least ϵ'^2. In particular, when this occurs, the authorities correctly opened all the audited commitments in the second simulation. The probability that all k of the voters will have the same audit bits in both simulations is 2^{-k}. If this does not occur, the corrupt authorities will publish $r_{(1,v_i)}, r_{(2,v_i)}$ such that $d_{3,v_i} = C(x_{v_i}, r_{(1,v_i)} + r_{(2,v_i)})$ for at least one of the honest voters. Again, this will allow M to open d_{3,v_i} in two different ways.

Taking a union bound, in the worst case the probability that M succeeds is at least $\epsilon'^2 - 2^{-k} = poly(\epsilon) - 2^{-k}$. If ϵ is non-negligible, then M can break the binding property of the commitment with non-negligible probablity. □

7. PROOF OF RECEIPT-FREENESS (THEOREM 5.2)

The definition of receipt-freeness shares many elements with the security definitions of the universal composability framework, hence the proofs will also be very similar. In both cases, we must show that the view of an adversary in the real world is indistinguishable from its view of a simulated protocol execution in an ideal world, where there exists an ideal simulator, $\mathcal{I}$. There is one main difference: the adversaries (in both the real and ideal worlds) can perform an additional action: coercing a party. Each party has, in addition to their input, a "coercion-response"

bit and a fake input (both also hidden from the adversary). The coercion-response bit determines how they will respond to coercion. When the bit is 1, a coerced party behaves exactly as if it were corrupted. When the bit is 0, however, instead it executes a "coercion-resistance strategy". In the ideal world, the strategy is to send its real input to $\mathcal{F}_{\text{vote}}$, but lie to the adversary and claim its input was the fake input. In the real world, the coercion-resistance strategy is specified as part of the protocol (see Section 5.2).

Like the proof of security in the UC model, the proof of Theorem 5.2 has two parts: the description of the simulation run by $\mathcal{I}$ in the ideal world, and a proof that the adversary's views in the real and ideal worlds are indistinguishable. The simulation is almost identical to the simulation in the proof of Theorem 5.1. The difference is what happens when $\mathcal{A}$ coerces a voter (this does not occur in the original simulation). $\mathcal{I}$ handles coercions of voters exactly like corruptions (i.e., they do not run the coercion-resistance strategy), with the following modifications:

(1) $\mathcal{I}$ coerces rather than corrupts the ideal voter.
(2) If voter v was coerced before step 4 of the Voting phase (i.e., before entering the voting booth), $\mathcal{I}$ sends a **Vote** $v, *$ command to $\mathcal{F}_{\text{vote}}$ (signifying a forced random vote), instead of a standard vote command.
(3) If the adversary causes the voter to behave in a way that would invalidate her vote (e.g., send syntactically incorrect commands, or abort before casting a ballot), $\mathcal{I}$ sends a **Vote** $v, \perp$ command to $\mathcal{F}_{\text{vote}}$ (signifying a forced abstention).

7.1 Indistinguishability of the Real and Ideal Worlds

To complete the proof of Theorem 5.2, we prove the following lemma:

LEMMA 7.1. *The adversary's view in the real world is identically distributed to its view in the ideal world.*

PROOF. Note that we only need to consider the case where both voting authorities are honest. Hence, the adversary's view consists only of the verifier's view (the random beacon and the information on the bulletin board) and the views of corrupted voters and coerced voters (which, in the real world, may be fake, depending on the value of their coercion-response bit).

We can assume the adversary also determines the inputs, fake inputs, and coercion-response bits of all the voters (these are not given to $\mathcal{I}$).

First, note that the generated ballots always consist of uniformly random values, independet of the voters' inputs, and the private part of the ballot is independent (as a random variable) of the public part of the ballot (since the commitments are perfectly hiding). Thus, the view generated by the coercion-resistance strategy is identically distributed to the view of an honest voter in the real world, which in turn is identically distributed to the view simulated by $\mathcal{I}$.

Since, when authorities are honest, corrupt voters can have no effect on other voters (except by changing the final tally), we can assume w.l.o.g. that the adversary does not corrupt voters (it can simply determine the inputs for honest voters). The only remaining possibility for a difference between the adversary's views is the joint distribution of the inputs, fake inputs, coercion-resistance bits and the final tally. However, since $\mathcal{I}$ can perfectly equivocate on commitments, the simulated tally it

produces will always be consistent with the ideal tally, and distributed identically to the tally in the real world. □

8. DISCUSSION AND OPEN PROBLEMS

Multiple Questions on a Ballot. As shown in the "illustrated example", our voting protocol can be easily adapted to use multiple questions on the same ballot. If there are many questions, the pattern of votes on a single ballot may uniquely identify a voter, hence tallying the questions together may violate voter privacy. In this case, the tally protocol should be performed separately for each question (or for each small group).

More than Two Authorities. We described the protocol using two authorities. The abstract protocol can be extended to an arbitrary number of authorities (although this may require finding a threshold version of the encryption scheme). However, a major stumbling block is the human element: even for two authorities this protocol may be difficult for some users. Dividing a vote into three parts will probably be too complex without additional ideas in the area of human interface.

Receipt-Freeness with a Corrupt Authority. The current protocol is not receipt-free if even one of the authorities is corrupt. Note that this is not a problem in the proof, but in the protocol itself (if the voter does not know which authority is corrupt): the voter can't tell which of the ballots the coercer will have access to, so she risks getting caught if she lies about the value she erased from the ballot. It is an interesting open question whether this type of attack can be prevented.

Better Human Interface. Probably the largest hurdle to implementing this protocol is the human interface. Devising a simple human interface for modular addition could prove useful in other areas as well.

APPENDIX

A. HOMOMORPHIC COMMITMENT AND ENCRYPTION SCHEMES OVER IDENTICAL GROUPS

Our voting scheme requires a perfectly private commitment scheme with "matching" semantically-secure encryption schemes. The commitment scheme's message and randomizer spaces must both be groups, and the commitment scheme must be homomorphic (separately) in each of the groups. There must be a matching encryption scheme for each group, such that the encryption scheme's message space is homomorphic over that group.

To meet these requirements, we propose using the standard Paillier encryption scheme. The Paillier encryption public key consists of an integer $N = p_1 p_2$, where p_1 and p_2 are safe primes, and an element $e \in \mathbb{Z}^*_{N^2}$. The private key is the factorization of N. The encryption plaintext is in the group $\mathbb{Z}_n$

For the commitment scheme, we propose a modified version of the Pedersen commitment scheme where both messages and randomness are also in the group $\mathbb{Z}_N$. The commitment public key consists of N (the same value as the encryption public key) along with random generators g, h in the order N subgroup of $\mathbb{Z}^*_{4N+1}$. Below we give the details of this construction.

A.1 Modified Pedersen

The abstract version of Pedersen commitment has a public key consisting of a cyclic group G and two random generators $g, h \in G$ such that $\log_g h$ is not known to the committer. The cryptographic assumption is that $\log_g h$ is infeasible to compute.

The message and randomizer spaces for this scheme are both $\mathbb{Z}_{|G|}$. $C(m,r) \doteq g^m h^r$. Since g and h are both generators of the group, for any m, when r is chosen at random $g^m h^r$ is a random group element. Therefore, this scheme is perfectly hiding. If an adversary can find $(m_1, r_1) \neq (m_2, r_2)$ such that $g^{m_1} h^{r_1} = g^{m_2} h^{r_2}$, then it can compute $\log_g h = \frac{m_2 - m_1}{r_1 - r_2}$, violating the cryptographic assumption. Hence the scheme is computationally binding. It is easy to see that the scheme is homomorphic.

Finally, if we choose $g, h = g^x$, where g is chosen randomly and x is chosen randomly such that g^x is a generator, we get an identically distributed public key, but knowing x it is easy to equivocate.

In the "standard" implementation of Pedersen, G is taken to be the order q subgroup of $\mathbb{Z}_p^*$, where $p = 2q+1$ and both p and q are prime (i.e., p is a safe prime). g and h are randomly chosen elements in this group. The discrete logarithm problem in G is believed to be hard when p is a safe prime chosen randomly in $(2^n, 2^{n+1})$.

Our modified version of Pedersen takes G to be the order $N = p_1 p_2$ subgroup of $\mathbb{Z}_{4n+1}^*$, where p_1 and p_2 are safe primes and $4n+1$ is also prime (we can't use $2n+1$, since that is always divisible by 3 when p_1 and p_2 are safe primes). The computational assumption underlying the security of the commitment scheme is that, when p_1 is a random safe prime and g and h are random generators of G, computing $\log_g h$ is infeasible. Note that it is not necessary to keep the factorization of N secret (in terms of the security of the commitment scheme), but knowing the factorization is not required for commitment.

A.2 Choosing the Parameters

The connection between the keys for the commitment and encryption schemes makes generating them slightly tricky. On one hand, only one of the authorities can know the private key for the encryption scheme (since its purpose is to hide information from the other authority). On the other hand, the security of the commitment must be publicly verifiable (even if both authorities are corrupt), hence we cannot allow the authorities to choose the parameters themselves. Moreover, for the commitment to be binding, N must have a large *random* prime factor, and g and h must be chosen randomly.

Below, we sketch our proposed protocol for verifiably generating the system parameters. Protocol 6 generates the parameters for the Paillier encryption, and Protocol 7 the parameters for the modified Pedersen commitment. The basic idea is that A_1 can use zero-knowledge proofs to show that the modulus $N = p_1 p_2$ is product of two safe primes, and to prove that p_1 is a *random* safe prime: basically, that it is the outcome of a coin-flipping protocol that A_1 conducts with the random beacon (this is accomplished by the loop at step 1 in Protocol 6). Technically, the proofs should be output by Protocol 7 rather than 6 (since they are required to prove the security of the commitment scheme). However, to clarify the presentation we have included them in the encryption key-generation protocol.

The generators g, h for the order N subgroup of $\mathbb{Z}^*_{4N+1}$, needed for the Pedersen scheme, are simply random elements of $\mathbb{Z}_{4N+1}$ (chosen using the random beacon). This works because a random element $g \in_R \mathbb{Z}_{4N+1}$ will be an element of $\mathbb{Z}^*_{4N+1}$ with order $o(g) \in \{N, 2N, 4N\}$ except with negligible probability $(O(1/\sqrt{N}))$, assuming p_1 and p_2 are of order $O(\sqrt{N})$). If the order of g is $2N$ (resp. $4N$), then g^2 (resp. g^4) will have order N (this computation can be replicated by the verifiers).

The protocols require integer commitments that have an efficent zero-knowledge proof of multiplication. We need an integer commitment scheme whose setup can be performed using a random beacon (rather than a trusted party). One such possibility is the Damgård-Fujisaki scheme [Damgård and Fujisaki 2002], when instantiated using class groups rather than an RSA modulus. For the zero-knowledge proofs that p_1 and p_2 are safe primes, we can use the techniques of Camenisch and Michels [1999].

Note that if we had a trusted third party to help with setup, we could significantly simplify it. Even a third party that is only trusted *during the setup* could help (for instance, by allowing us to use Damgård-Fujisaki with an RSA modulus generated by the third party).

Protocol 6 Key Generation for Encryption (KG)

Input: Security parameter k

repeat {Generate a verifiable commitment C_1 to a random safe prime: p_1}
  Choose a random $p' \in_R \mathbb{Z}_{2^k}$
  Publish a commitment C' to p'
  Interpret the next output of the random beacon as a number $p'' \in_R \mathbb{Z}_{2^k}$
  if $p_1 = p' + p'' \pmod{2^k}$ is a safe prime **then**
    Publish a commitment C_1 to p_1
    Prove in zero-knowledge (using the random beacon) that C_1 is a commitment to a safe prime, and that it is the sum of p'' and the committed value of C'. {if the commitment is statistically hiding, this will be a zero-knowledge proof of knowledge}.
  else
    Publicly open the commitment C', revealing p'.
  end if
until p_1 is a safe prime
Privately choose a random k-bit safe prime p_2, such that $4p_1p_2 + 1$ is prime.
Publish $N = p_1p_2$
Publish a commitment C_2 to p_2
Prove in zero-knowledge that C_2 is a commitment to a safe prime, and that the product of the values committed to in C_1 and C_2 is N.
Run the standard Paillier key generation using N as the modulus.

B. ZERO-KNOWLEDGE PROOFS OF KNOWLEDGE

Our protocols require proving statements in zero-knowledge about committed values. Since we use perfectly-hiding commitments, proving that there exists an opening of a commitment with some property is meaningless: there exist openings of

Protocol 7 Key Generation for Commitment (K)

Input: Modulus N output by KG (Protocol 6)

1: Interpret the next output of the random beacon as elements $g, h \in Z^*_{4N+1}$.
2: **for** $x \in \{g, h\}$ **do**
3: **while** $x^N \not\equiv 1 \pmod{N^2}$ **do**
4: $x \leftarrow x^2$
5: **end while**
6: **end for**
7: Output g, h and N.

the commitment to every value. Instead, we use zero-knowledge proofs of knowledge [Bellare and Goldreich 1992]. Roughly, there exists an efficient "knowledge extractor" that, given oracle access to a prover that succeeds with with some non-negligible probability, can output a value consistent with what the prover claims to know.

In this section we briefly describe the zero-knowledge proof subprotocols. These are all honest-verifier, public-coin, zero-knowledge proofs of knowledge, using standard cut-and-choose techniques. When they are used publicly (i.e., on the bulletin board), the verifier's coins are taken from the random beacon, hence the honest-verifier assumption makes sense. Although more efficient protocols exist for these applications [Boudot 2000; Groth 2002], for the purpose of this paper we concentrate on simplicity and ease of understanding.

B.1 Proof That Two Commitments Are Equivalent

In step 7 of Protocol 3, authority A_1 must prove that an encryption it generated has the same value as a previously published commitment. The following subprotocol works for any two homomorphic commitment schemes (in this case we can consider the encryption a commitment scheme), as long as their message groups are isomorphic. Since our commitment scheme is symmetric (we can consider it a commitment to the randomness), this protocol works for that case as well.

We will assume two commitment schemes C_1 and C_2, with message space $\mathcal{M}$, commitment spaces $\mathcal{C}_1, \mathcal{C}_2$ (resp.) and randomness groups $\mathcal{R}_1, \mathcal{R}_2$ (resp.).

Let $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$ be the commitments for which we are proving "equivalence". Formally, what the protocol proves is that the prover knows a value $x \in \mathcal{M}$ and values $r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2$ such that $c_1 = C_1(x, r_1)$ and $c_2 = C_2(x, r_2)$. The complete protocol consists of k repetitions of Protocol 8; the probability that the prover can cheat successfully is exponentially small in k.

B.2 Proof of Commitment Shuffle

We say a vector of commitments is "a valid shuffle" of a second vector if, whenever the prover can open one vector, it can open both vectors to permutations of the same set of values. Note that this property is an equivalence relation (with respect a single prover).

An important point is that we do not require the prover to show that it can *open* either of the vectors of commitments. This property is necessary, because our voting protocol requires a voting authority to shuffle commitments that it does not

Protocol 8 Zero-Knowledge Proof That Two Committments Are Equivalent

Input: Verifier receives $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$, Prover receives $x \in \mathcal{M}$ and $r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2$ such that $c_1 = C_1(x, u_1)$ and $c_2 = C_2(x, u_2)$

Prover chooses values $s \in_R \mathcal{M}$ and $u_1 \in_R \mathcal{R}_1, u_2 \in \mathcal{R}_2$
Prover sends to verifier: $d_1 \doteq C_1(x + s, r_1 + u_1)$ and $d_2 \doteq C_2(x + s, r_2 + u_2)$
Verifier sends to the prover a random bit $b \in_R \{0, 1\}$
if $b = 0$ **then**
    Prover sends to the verifier: s, u_1 and u_2.
    Verifier checks that $d_1 = c_1 C(s, u_1)$ and $d_2 = c_2 C(s, u_2)$
else
    Prover sends to the verifier: $x + s$, $r_1 + u_1$ and $r_2 + u_2$
    Verifier checks that $d_1 = C(x + s, r_1 + u_1)$ and $d_2 = C(x + s, r_2 + u_2)$
end if

know how to open.

To construct a zero-knowledge proof, we use a standard cut-and-choose technique. Roughly, the prover publishes a *third* vector of commitments, then, according to the verifier's choice, it either shows that this third vector is a valid shuffle of the first, or that third vector is a valid shuffle of the second. If it is both, the first vector must be a valid shuffle of the second (and vice-versa).

Formally, let $c_1, \ldots, c_n \in \mathcal{C}$ and $c'_1, \ldots, c'_n \in \mathcal{C}$ be commitments. The prover must show that it knows a permutation $\sigma\colon [n] \mapsto [n]$ and values $r_1, \ldots, r_n \in \mathcal{R}$ such that for all $i \in [n]$: $c'_i = c_{\sigma(i)} \cdot C(0, r_i)$.

The protocol consists of k repetitions of Protocol 9 (where k is the security parameter).

Protocol 9 Zero-Knowledge Proof of Valid Shuffle

Prover chooses a random permutation $\pi\colon [n] \mapsto [n]$
Prover chooses values $r'_1, \ldots, r'_n \in_R \mathcal{R}$.
for $1 \le i \le n$ **do**
    Prover sends to the verifier: $d_i \doteq c'_{\pi(i)} \cdot C(0, r'_i)$
end for
Verifier sends to the prover a random bit $b \in_R \{0, 1\}$
if $b = 0$ **then**
    Prover sends to the verifier: π
    Prover sends to the verifier: $r'_1, \ldots, r'_n$
    for $1 \le i \le n$ **do**
        Verifier checks that $d_i = c'_{\pi(i)} \cdot C(0, r'_i)$
    end for
else
    Prover sends to the verifier: $\sigma \circ \pi$
    for $1 \le i \le n$ **do**
        Prover sends to the verifier: $s_i \doteq r_{\pi(i)} + r'_i$
        Verifier checks that $d_i = c_{\sigma \circ \pi(i)} \cdot C(0, s_i)$
    end for
end if

B.3 Proof that a Committed Value is in $\mathbb{Z}_{2^k}$

In steps 9 and 10 of Protocol 2, each authority must prove that a committed value "is in an appropriate range".

Formally, $z \in \mathcal{C}$ be a commitment. The prover must show that it knows values $x \in \mathcal{M}$ and $u \in \mathcal{R}$ such that $z = C(x, u)$ and $x \in \mathbb{Z}_{2^k}$ (i.e., that it knows how to open the commitment to a value in the range).

Roughly, the idea behind the protocol is to show that the binary representation of z has only k bits, by homomorphically constructing an equivalent commitment from k commitments to binary values. The protocol itself appears as Protocol 10.

Protocol 10 Zero-Knowledge Proof that a Committed Value is in $\mathbb{Z}_{2^k}$

Input: Verifier receives $z \in \mathcal{C}$, Prover receives $x \in \mathcal{M}$ and $u \in \mathcal{R}$ such that $z = C(x, u)$, $x < 2^k$.

1: Denote: $c_0 \doteq C(0, 0)$ and $c_1 \doteq C(1, 0)$

2: Denote: $b_0, \ldots, b_{k-1}$ the binary representation of x.

3: Prover chooses values $r_1, \ldots, r_{2k} \in_R \mathcal{R}$.

4: **for** $1 \leq i \leq 2k$ **do**

5: Prover computes and sends to verifier:

$$d_{i-1} \doteq \begin{cases} C(b_{i-1}, r_i) & \text{if } i \leq k \\ C(1 - b_{i-k-1}, r_i) & \text{if } i > k \end{cases}$$

6: **end for**

7: Prover proves to verifier (using Protocol 9) that $d_0, \ldots, d_{2k-1}$ is a valid shuffle of $\underbrace{c_0, \ldots, c_0}_{\times k}, \underbrace{c_1, \ldots, c_1}_{\times k}$ {note that this is indeed the case, since there are exactly k commitments to 0 and k commitments to 1}

8: Prover and verifier both compute:

$$z' \doteq \prod_{i=0}^{k-1} d_i^{2^i} = C\left(\sum_{i=0}^{k-1} 2^i b_i, \sum_{i=0}^{k-1} 2^i r_i\right)$$

9: Prover proves to verifier (using Protocol 8) that z' and z are commitments to the same value. {Note that this is the case, since by the definition of $b_0, \ldots, b_{k-1}$, $x = \sum_{i=0}^{k-1} 2^i b_i$}

C. A FORMAL DEFINITION OF RECEIPT-FREENESS

This section is taken almost verbatim from [Moran and Naor 2006]. This formalization of receipt-freeness is a generalization of Canetti and Gennaro's definition (and so can be used for any secure function evaluation), and is strictly stronger (i.e., any protocol that is receipt-free under this definition is post-factum incoercible as well). The difference is the adversarial model we consider. Canetti and Gennaro only allow the adversary to query coerced players players after the protocol execution is complete.

Unfortunately, this "perfect" receipt-freeness is impossible to achieve except for trivial computations. This is because for any non-constant function, there must exist some party P_i and some set of inputs to the other parties such that the output of the function depends on the input used by x_i. If the adversary corrupts all parties except for P_i, it will be able to tell from the output of the function what input what used by P_i, and therefore whether or not P_i was a puppet.

This is the same problem faced by Canetti and Genaro in defining post-factum incoercibility. Like theirs, this definition sidesteps the problem by requiring that any "coercion" the adversary can do in the real world it can also do in an ideal world (where the parties' only interaction is sending their input to an ideal functionality that computes the function). Thus, before we give the formal definition of receipt-freeness, we must first describe the mechanics of computation in the ideal and real worlds. Below, f denotes the function to be computed.

C.1 The Ideal World

The ideal setting is an extension of the model used by Canetti and Genaro (the post-factum incoercibility model). As in their model, there are n parties, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$. Each party also has a "fake" input; they are denoted $x'_1, \ldots, x'_n$. The "ideal" adversary is denoted $\mathcal{I}$.

In our model we add an additional input bit to each party, $c_1, \ldots, c_n$. We call these bits the "coercion-response bits". A trusted party collects the inputs from all the players, computes $f(x_1, \ldots, x_n)$ and broadcasts the result. In this setting, the ideal adversary $\mathcal{I}$ is limited to the following options:

(1) Corrupt a subset of the parties. In this case the adversary learns the parties' real inputs and can replace them with inputs of its own choosing.
(2) Coerce a subset of the parties. A coerced party's actions depend on its coercion-response bit c_i. Parties for which $c_i = 1$ will respond by sending their real input x_i to the adversary (we'll call these "puppet" parties). Parties for which $c_i = 0$ will respond by sending the fake input x'_i to the adversary.
    At any time after coercing a party, the adversary can provide it with an alternate input x''_i. If $c_i = 1$, the coerced party will use the alternate input instead of its real one (exactly as if it were corrupted). If $c_i = 0$, the party will ignore the alternate input (so the output of the computation will be the same as if that party were honest). There is one exception to this rule, and that is if the alternate input is one of the special values $\perp$ or $*$, signifying a forced abstention or forced random vote, respectively. In this case the party will use the input $\perp$, or choose a new, random, input regardless of the value of c_i.

$\mathcal{I}$ can perform these actions iteratively (i.e., adaptively corrupt or coerce parties based on information gained from previous actions), and when it is done the ideal functionality computes the function. $\mathcal{I}$'s view in the ideal case consists its own random coins, the inputs of the corrupted parties, the inputs (or fake inputs) of the coerced parties and the output of the ideal functionality $f(x_1, \ldots, x_n)$ (where for corrupted and puppet parties x_i is the input chosen by the adversary).

Note that in the ideal world, the only way the adversary can tell if a coerced party is a puppet or not is by using the output of the computation – the adversary has no other information about the coercion-response bits.

C.2 The Real World

Our real-world computation setting is also an extension of the real-world setting in the post-factum incoercibility model. We have n players, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$ and fake inputs $x'_1, \ldots, x'_n$. The adversary in the real-world is denoted $\mathcal{A}$ (the "real" adversary).

The parties are specified by interactive Turing machines restricted to probabilistic polynomial time. Communication is performed by having special communication tapes: party P_i sends a message to party P_j by writing it on the (i, j) communication tape (we can also consider different models of communication, such as a broadcast tape which is shared by all parties). Our model does not allow erasure; communication tapes may only be appended to, not overwritten. The communication is synchronous and atomic: any message sent by a party will be received in full by intended recipients before the beginning of the next round.

We extend the post-factum incoercibility model by giving each party a private communication channel with the adversary and a special read-only register that specifies its corruption state. This register is initialized to the value "honest", and can be set by the adversary to "coerced" or "corrupted". In addition, each party receives the coercion response bit c_i. We can think of the ITM corresponding to each party as three separate ITMs (sharing the same tapes), where the ITM that is actually "running" is determined by the value of the corruption-state register. Thus, the protocol specifies for party P_i a pair of ITMs (H_i, C_i), corresponding to the honest and coerced states (the corrupt state ITM is the same for all protocols and all parties).

The computation proceeds in steps: In each step $\mathcal{A}$ can:

(1) Corrupt a subset of the parties by setting their corresponding corruption-state register to "corrupted". When its corruption-state register is set to "corrupted", the party outputs to the adversary the last state it had before becoming corrupted, and the contents of any messages previously received. It then waits for commands from the adversary and executes them. The possible commands are:

    Copy to the adversary a portion of one of its tapes (input, random, working or communication tapes).

    Send a message specified by the adversary to some subset of the other parties.

    These commands allow the adversary to learn the entire past view of the party and completely control its actions from that point on. We refer to parties behaving in this manner as executing a "puppet strategy".

(2) Coerce a subset of the parties by setting their corresponding corruption-state register to "coerced". From this point on $\mathcal{A}$ can interactively query and send commands to the coerced party as it can to corrupted parties. The coerced party's response depends on its coercion-response bit c_i. If $c_i = 1$, the party executes the puppet strategy, exactly as if it were corrupted. If $c_i = 0$, it runs the coercion-resistance strategy C_i instead. The coercion-resistance strategy specifies how to respond to $\mathcal{A}$'s queries and commands.

(3) Send commands to corrupted and coerced parties (and receive responses).

$\mathcal{A}$ performs these actions iteratively, adaptively coercing, corrupting and interacting with the parties. $\mathcal{A}$'s view in the real-world consists of its own randomness, the inputs, randomness and all communication of corrupted parties, its communications with the coerced parties and all public communication.

C.3 A Formal Definition of Receipt-Freeness

Definition C.1. A protocol is receipt-free if, for every real adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{I}$, such that for any input vector $x_1, \ldots, x_n$, fake input vector $x'_1, \ldots, x'_n$ and any coercion-response vector $c_1, \ldots, c_n$:

(1) $\mathcal{I}$'s output in the ideal world is indistinguishable from $\mathcal{A}$'s view of the protocol in the real world with the same input and coercion-response vectors (where the distributions are over the random coins of $\mathcal{I}$, $\mathcal{A}$ and the parties).
(2) Only parties that have been corrupted or coerced by $\mathcal{A}$ in the real world are corrupted or coerced (respectively) by $\mathcal{I}$ in the ideal world.

It is important to note that even though a protocol is receipt-free by this definition, it may still be possible to coerce players (a trivial example is if the function f consists of the player's inputs). What the definition does promise is that if it is possible to coerce a party in the real world, it is also possible to coerce that party in the ideal world (i.e. just by looking at the output of f).

ACKNOWLEDGMENTS

We would like to thank Tal Rabin for suggesting that all the randomness for the secret sharing can be chosen by the voting authorities rather than the voter. This significantly simplified the protocol's user inteface. We would also like to thank the anonymous reviewers for their helpful comments.

REFERENCES

Adida, B. and Rivest, R. L. 2006. Scratch & vote: self-contained paper-based cryptographic voting. In *Proceedings of WPES '06, the 5th ACM workshop on Privacy in electronic society* (Alexandria, VA, USA), J. Stern, Ed. ACM Press, New York, NY, USA, 29–40.

Aumann, Y., Ding, Y. Z., and Rabin, M. O. 2002. Everlasting security in the bounded storage model. *IEEE Trans. on Information Theory 48,* 6, 1668–1680.

Bellare, M. and Goldreich, O. 1992. On defining proofs of knowledge. In *Proceedings of CRYPTO 1992, 12th Annual International Cryptology Conference* (Santa Barbara, CA, USA), E. F. Brickell, Ed. LNCS, vol. 740. Springer-Verlag Inc., New York, NY, USA, 390–420.

Benaloh, J. and Tuinstra, D. 1994. Receipt-free secret-ballot elections. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing* (Montréal, Québec, Canada). ACM Press, New York, NY, USA, 544–553.

Boudot, F. 2000. Efficient proofs that a committed number lies in an interval. See Preneel [2000], 431–444.

Bryans, J. W. and Ryan, P. Y. A. 2004. A simplified version of the Chaum voting scheme. Tech. Rep. CS-TR 843, University of Newcastle. May.

Camenisch, J. and Michels, M. 1999. Proving in zero-knowledge that a number is the product of two safe primes. In *Proceedings of EUROCRYPT 1999, International Conference on the Theory and Application of Cryptographic Techniques* (Prague, Czech Republic), J. Stern, Ed. LNCS, vol. 1592. 107–122.

Canetti, R. 2000. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067.

Canetti, R. and Gennaro, R. 1996. Incoercible multiparty computation. In *37th Annual Symposium on Foundations of Computer Science* (Burlington, VT, USA). IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 504–513.

Chaum, D. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24,* 2, 84–88.

Chaum, D. 2004. E-voting: Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy 2,* 1 (Jan./Feb.), 38–47.

Chaum, D. 2006. `http://punchscan.org/`.

Cohen(Benaloh), J. D. and Fischer, M. J. 1985. A robust and verifiable cryptographically secure election scheme. In *Proceedings of the 26th annual Symposium on Foundations of Computer Science* (Portland, OR, USA). IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 372–382.

Cramer, R., Franklin, M., Schoenmakers, B., and Yung, M. 1996. Multi-authority secret-ballot elections with linear work. In *Proceedings of EUROCRYPT 1996, International Conference on the Theory and Application of Cryptographic Techniques* (Saragossa, Spain), U. Maurer, Ed. LNCS, vol. 1070. Springer-Verlag Inc., New York, NY, USA, 72–83.

Cramer, R., Gennaro, R., and Schoenmakers, B. 1997. A secure and optimally efficient multi-authority election scheme. In *Proceedings of EUROCRYPT 1997, International Conference on the Theory and Application of Cryptographic Techniques* (Konstanz, Germany), W. Fumy, Ed. Vol. 1233. Springer-Verlag Inc., New York, NY, USA, 103–118.

Damgård, I. B. and Fujisaki, E. 2002. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Proceedings of ASIACRYPT 2002, International Conference on the Theory and Application of Cryptology and Information Security* (Queenstown, New Zealand), Y. Zheng, Ed. LNCS, vol. 2501. Springer, New York, NY, USA, 125–142.

Fujioka, A., Okamoto, T., and Ohta, K. 1992. A practical secret voting scheme for large scale elections. In *Proceedings of AUSCRYPT 1992, Workshop on the Theory and Application of Cryptographic Techniques* (Gold Coast, Queensland, Australia), J. Seberry and Y. Zheng, Eds. LNCS, vol. 718. Springer-Verlag Inc., New York, NY, USA, 244–251.

Groth, J. 2002. A verifiable secret shuffle of homomorphic encryptions. In *Proceedings of PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography* (Miami, FL, USA), Y. Desmedt, Ed. Lecture Notes in Computer Science, vol. 2567. Springer, 145–160.

Hirt, M. and Sako, K. 2000. Efficient receipt-free voting based on homomorphic encryption. See Preneel [2000], 539+.

Moran, T. and Naor, M. 2006. Receipt-free universally-verifiable voting with everlasting privacy. In *Proceedings of CRYPTO 2006, 26th Annual International Cryptology Conference* (Santa Barabara, CA, USA), C. Dwork, Ed. LNCS, vol. 4117. Springer, New York, NY, USA, 373–392. `http://www.wisdom.weizmann.ac.il/~talm/papers/MN06-voting.pdf`.

Naor, M. and Shamir, A. 1994. Visual cryptography. In *Proceedings of EUROCRYPT 1994, Workshop on the Theory and Application of Cryptographic Techniques* (Perugia, Italy), A. De Santis, Ed. LNCS, vol. 950. Springer-Verlag Inc., New York, NY, USA, 1–12.

Neff, C. A. 2004. Practical high certainty intent verification for encrypted votes. `http://www.votehere.net/vhti/documentation/vsv-2.0.3638.pdf`.

Popoveniuc, S. and Hosp, B. 2006. An introduction to punchscan. `http://punchscan.org/papers/popoveniuc_hosp_punchscan_introduction.pdf`.

Preneel, B., Ed. 2000. *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceedings* (Bruges, Belgium). LNCS, vol. 1807. Springer, New York, NY, USA.

Rabin, M. O. 1983. Transaction protection by beacons. *J.Computer and System Sciences 27,* 2, 256–267.

Reynolds, D. J. 2005. A method for electronic voting with coercion-free receipt. Presentation: `http://www.win.tue.nl/~berry/fee2005/presentations/reynolds.ppt`.

Ryan, P. Y. A. 2005. A variant of the Chaum voter-verifiable scheme. In *Proceedings of WITS '05, 2005 Workshop on Issues in the Theory of Security* (Long Beach, CA, USA). ACM Press, New York, NY, USA, 81–88.

Shamir, A. 2006. Cryptographers panel, RSA conference. Webcast: http://media.omediaweb.com/rsa2006/1_5/1_5_High.asx.

Nomenclature

$(pk^{(\mathcal{X})}, sk^{(\mathcal{X})})$ The public/secret keys (resp.) for the encryption scheme with message space $\mathcal{X}$, where $\mathcal{X} \in \{\mathcal{M}, \mathcal{R}\}$, page 16

ℓ Length of random inputs to K (in bits), page 11, 12

$e_{1,i}^{(\mathcal{M})}$ Encryption of authority A_1's share of voter i's choice $(E^{(\mathcal{M})}(x_i - t_{(2,i)}))$, page 20

$e_{1,i}^{(\mathcal{R})}$ Encryption of authority A_1's share of the commitment randomness for voter $i(E^{(\mathcal{R})}(r_{(1,i)}))$, page 20

$\mathcal{F}_{\text{vote}}$ The ideal voting functionality, page 13

$\mathcal{C}$ Commitment group for commitment scheme (group operation is "$\cdot$"), page 11

$\mathcal{E}^{(\mathcal{X})}$ Group of encrypted messages, where $\mathcal{X} \in \{\mathcal{M}, \mathcal{R}\}$. The group operation is $\cdot$, page 12

$\mathcal{K}$ Set of public keys for commitment scheme., page 11

$\mathcal{R}$ Randomizer group for commitment scheme (group operation is "+"), page 11

ρ_i The decrypted randomness for the commitment $d_{1,i}$, page 20

σ_a Private permutation (shuffle) of voters chosen by authority A_a. The composition of the permutations is the one eventually revealed, page 19

τ_i The tally for candidate i, page 14

$\vec{w}$ Tuple, (a, i, b), identifying ballot part, where A_a is the generating authority, i is the voter id and b is a bit used as a serial number, page 16

ξ_i The decrypted message for the commitment $d_{1,i}$, page 20

A_a Voting authority a, page 13

b_v Serial number of ballot parts chosen by voter to be cast ($1 - b_v$ is the serial number of the ballot parts used for auditing), page 17

$B_{\vec{w}}$ The ballot part identified by the tuple $\vec{w}$, page 16

C Commitment function, parameterized by the public key. $C(x, r) \in \mathcal{C}$ is a commitment to x with secret randomness r, page 11

$c_{(1,v)}$ Shorthand for $c_{(1,v,b_v),s_v}$, the commitment to the value $x_v - t_{(2,v)} \pmod m$ (this is chosen by the voter from the set of commitments published in $B_{(1,v)}$), page 17

$c_{(2,v)}$ The commitment to the masking value $t_{(2,v)}$, page 17

$c_{\vec{w},j}$ A commitment to $t_{\vec{w}} + j \pmod m$; in ballot parts generated by A_1, page 16

$c_{\vec{w}}$ A commitment to $t_{\vec{w}}$, page 16

cpk The public key for the commitment scheme, page 16

$D^{(\mathcal{M})}$ Decryption function with message space $\mathcal{X}$, where $\mathcal{X} \in \{\mathcal{M}, \mathcal{R}\}$, page 12

$d_{3,v}$ A commitment to voter v's choice generated by homomorphically adding the commitments to the voter's masked choice and the commitments to the masking value, page 18

$d_{a,1}, \ldots, d_{a,n}$ A vector of commitments to the voters' choices published by authority A_a who generates it by shuffling and rerandomizing the vector $d_{a+1,1}, \ldots, d_{a+1,n}$, page 18

$E^{(\mathcal{M})}$ Encryption function with message space $\mathcal{X}$, where $\mathcal{X} \in \{\mathcal{M}, \mathcal{R}\}$; $E(x, r)$ is an encryption of x with randomness r, page 12

K Public key generation algorithm for commitment scheme., page 11

k Security parameter; we consider 2^{-k} negligible, page 13

K' Equivocable secret key generation algorithm for commitment scheme (the output of K' is used as an input to K), page 12

$KG^{(\mathcal{X})}$ Key generation algorithm for encryption scheme with message space $\mathcal{X}$, where $\mathcal{X} \in \{\mathcal{M}, \mathcal{R}\}$, page 12

m Number of candidates, page 3

n Number of voters, page 13

P_K, V_K Prover and Verifier algorithms, resp. for the commitment key generation algorithm, page 11

$r_{\vec{w},j}$ The secret randomness for the commitment $c_{\vec{w},j}$; in ballot parts generated by A_1, page 16

$r_{\vec{w}}$ The secret randomness for the commitment $c_{\vec{w}}$, page 16

s_v The masked choice for voter v: $s_v \doteq x_v - t_{(1,v)} - t_{(2,v)} \pmod{m}$, page 17

$t_{(a,v)}$ Shorthand for $t_{(a,v,b_v)}$: the masking value used by authority A_a for voter v's choice, page 17

$t_{\vec{w}}$ A random masking value ($t_{\vec{w}} \in \mathbb{Z}_m$) for ballot-part $\vec{w}$, page 16

$u_{a,i}$ The randomizing value added to the randomness of the commitment $d_{a+1,\sigma_a(i)}$ in the shuffle performed by authority A_a, page 19

v Used to represent the voter whose index is v, page 13

W The number of corrupt voters for which $\mathcal{I}$ did not cast a vote during the voting phase, page 23

x_v Voter v's choice of candidate ($x_v \in \mathbb{Z}_m$), page 13

$z_{a,i}$ The randomizing value added to the message of the $d_{a+1,\sigma_a(i)}$ in the shuffle performed by authority A_a. For A_2 this value is uniformly chosen from $\mathbb{Z}_{2^k}$, while for A_1 it is chosen from $\mathbb{Z}_{2^{2k}}$. The actual value added is $z_{a,i}m$, page 19

x'_v Random value chosen by $\mathcal{I}$ in place of unknown choice of voter v, page 23

$\mathcal{I}$ The "ideal adversary" in the UC model, page 21

$\mathcal{A}$ The "real-world adversary" in the UC model, page 21