Split-Ballot Voting: Everlasting Privacy With Distributed Trust *

Tal Moran
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel
tal.moran@weizmann.ac.il

Moni Naor†
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel
moni.naor@weizmann.ac.il

ABSTRACT

In this paper we propose a new voting protocol with 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 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

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:

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 both that their own votes were "cast as intended" and that all 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 [23]).

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 [2], 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

*This work was partially supported by the Israel Science Foundation

†Incumbent of the Judith Kleeman Professorial Chair

CCS'07, October 29–November 2, 2007, Alexandria, Virginia, USA.

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. 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, using a simulation-based definition of receipt-freeness previously introduced by the authors [16]. Helping to show that rigorous proofs of correctness are not just "formalism for the sake of formalism", we demonstrate a subtle attack against the receipt-freeness of the Punchscan voting system [9] (see Section 2.4).

1.2 Related Work

Voting Protocols. Chaum proposed the first published electronic voting scheme in 1981 [7]. Many additional protocols were suggested since Chaum's. Among the more notable are [13, 10, 3, 11, 12, 14].

Only a small fraction of the proposed voting schemes satisfy the property of receipt-freeness. Benaloh and Tuinstra [3] 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 [14]). 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 [8], and later Neff [18], proposed universally-verifiable receipt-free voting schemes that overcome this problem. Recently, Reynolds proposed another protocol similar to Neff's [21].

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 touchscreen 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 a truly an encryption of her intended vote.

Chaum's original protocol used Visual Cryptography [17] 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 [4, 22]. A newer idea of Chaum's is the Punchscan voting system [9], which we describe in more detail in Section 2.4.

Recently, the authors proposed a voting protocol, based on statistically-hiding commitments, that combines everlasting security and a human-centric interface [16]. 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 [1] 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 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.

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 voting 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, al-

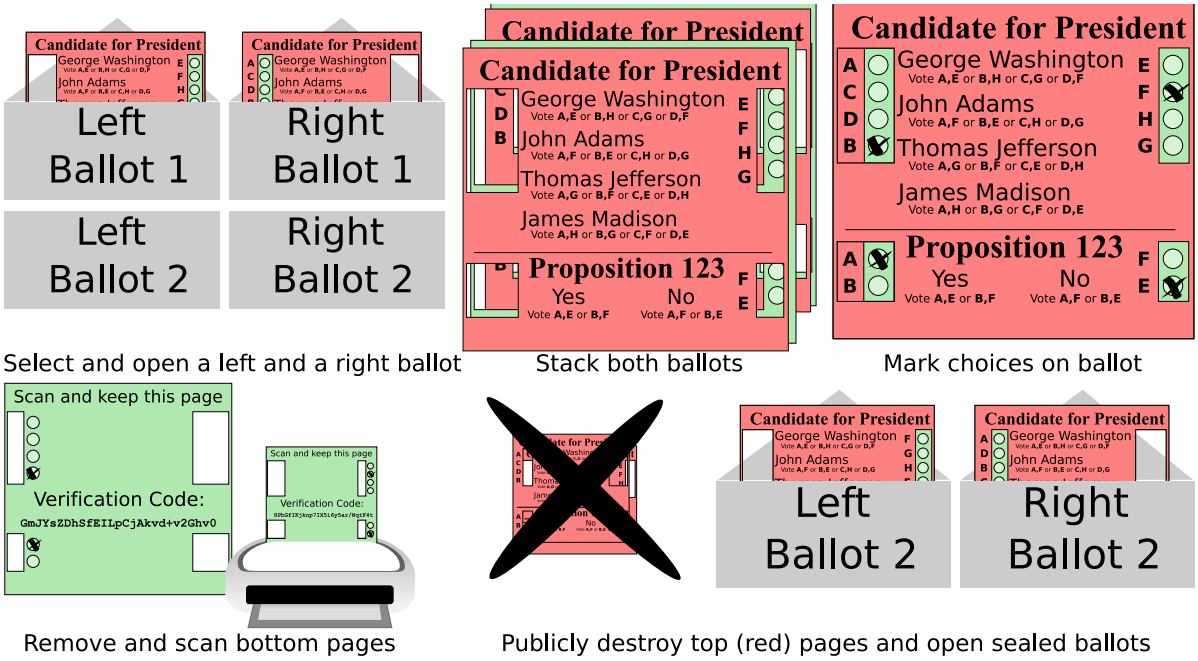

Figure 2.1: Illustrated Sample Vote

though 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 verification (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 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 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

The most questionable assumption we make with this protocol concerns human capability. It is essential to our protocol that the voter can do two things: randomly select a value from a set of values, and perform modular addition.

The first is a fairly standard assumption. The second seems highly suspect.

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 two stacked papers. The top paper contains explanations, as well as a random permutation of letters. It also contains holes through which the bottom paper can be seen. Next to each letter on the top page, the bottom paper contains a scannable bubble that can be marked with a pencil or pen; when the two papers are stacked the bubbles are visible through holes in the top paper.

The voter selects a letter by marking the corresponding bubble. Each candidate on the ballot can be chosen by some combinations of two letters. We construct the ballots in such a way that when two ballots are stacked, one from each authority, the letters and bubbles from both ballots are visible. The voter then chooses one of the letter combinations for her desired candidate, and marks the bubbles.

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 2.1). The election is for the office of president, and also includes a poll on "Proposition 123". The presidential candidates are George, John, Thomas and James.

Sarah, the voter, enters the polling place and receives four ballots in sealed envelopes: two "Left" ballots and two "Right" ballots (we can think of the two voting authorities as the "Left" authority and the "Right" authority). She takes the ballots and enters the polling booth. She then randomly chooses one of the Left ballots and one of the Right ballots and opens their envelopes. She removes the ballots, each of which consists of a red (top) and green (bottom) page. She stacks all four pages together (the order doesn't matter). Sarah wants to vote for Thomas and to vote Yes on Proposition 123. She finds her candidate's name on the top paper, and sees that he is represented by the pairs **(A,G)**, **(B,F)**, **(C,E)**, and **(D,H)**. She randomly picks **(B,F)** and marks her ballot.[1] She sees that to vote Yes on Proposition 123 she can choose either **(A,E)** or **(B,F)**. She randomly chooses **(A,E)** and fills the appropriate bubbles.

Sarah then separates the papers. She scans both bottom pages. The scanner can give immediate output so she can verify that she filled the bubbles correctly, and that the scanner correctly identified her marks.[2] 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.

The top pages she destroys, then demonstrates she has done so to an election official (the official will not be allowed to see the text on those pages, of course). Finally, the election official verifies that the two unvoted ballot envelopes are still sealed, then allows Sarah to open them. The complete (unvoted) ballots will also be kept for verification, or given to a helper organization. At home, Sarah will make sure the complete ballots are published on the bulletin board.

[1] One possible way to make sure the choice is truly random would be to use a physical aid, such as a spinner, coins or dice

[2] Note that Sarah doesn't have to *trust* the scanner (or its software) in any way: These pages will be kept by Sarah as receipts which she can use to prove that her vote was not correctly tabulated (if this does occur).

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 2.2).

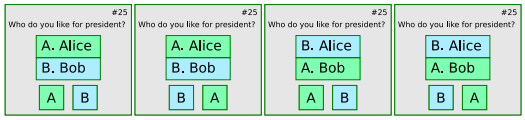

Figure 2.2: Punchscan Ballot Configurations

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 receipt), while the other is shredded (these steps are shown in Figure 2.3).

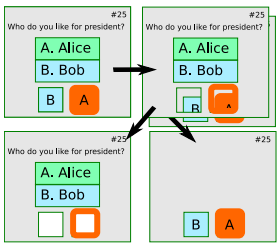

Figure 2.3: Punchscan Ballot

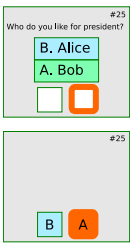

Figure 2.4: "Bad" Receipts

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 2.4 (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 2.2). 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 must note that the current version of Punchscan (as described in [19]) instructs the voter to commit to the layer she will take before entering the voting booth. The original purpose of this requirement was to prevent a different attack, but it suffices to foil the attack described above. The requirement does not appear in any other Punchscan literature, however, and demonstration elections using Punchscan did not enforce it (possibly because coercion was not considered a serious threat in that setting).

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 [9], Scratch&Vote [1]). 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.

Tamper-Evident Seals. In order to preserve privacy, ballots must be delivered to the voter sealed (so that no one else can see their contents). In order to preserve receipt-freeness, even the voter must not be able to see the contents of a ballot before opening it. Moreover, the voting authorities must be able to verify that the voter did not open the unvoted ballots.

To achieve this, we can make use of tamper-evident envelopes, or opaque sealed bags (which a voter cannot open undetected). A formal model for tamper-evident seals was previously developed by the authors [15]. The "distinguishable-envelope" model in [15] captures our requirements precisely.

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. In order to guarantee everlasting privacy, communication between the voting authorities is assumed to take place using untappable private channels. This is a fairly reasonable assumption: the voting authorities can be physically close and connected by direct physical channels. Note that if this assumption is not satisfied, the protocol is still computationally private (but is no longer UC-secure or information-theoretically private).

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 [20], 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 knowl-

edge 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 space of commitments $(\mathcal{C}, \cdot)$. To reduce clutter, we omit the key parameter when it is obvious from context (i.e., we write $C(m, r)$ instead of $C_{cpk}(m, r)$).

P_K and V_K are a "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 PPTs K^*, P_K^* (corresponding 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.
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, $C(m_1, r_1) \cdot C(m_2, r_2) = C(m_1 + m_2, r_1 + r_2)$.

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, and for every $l \in \{0,1\}^{\ell}$ and $m \in \mathcal{M}$ and $c \in \mathcal{C}$, $C_{K(K'(l))}(m, C'(l, c, m)) = c$ (i.e., it is possible to generate a public-key that is identical to a normal public key, such that it is possible to 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 $\left(KG^{(\mathcal{M})}, E^{(\mathcal{M})}, D^{(\mathcal{M})}\right)$ and $\left(KG^{(\mathcal{R})}, E^{(\mathcal{R})}, D^{(\mathcal{R})}\right)$, 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 clarify the presentation, below we omit the randomness and the superscript for the encryption schemes where it can be understood from the context (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 approached for defining security.

The basic functionality is defined and proved in Canetti's Universal Composability framework [5]. 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.

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 more) of the voting authorities is 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).[3] 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 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 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.

Tally On receiving this command from a voting authority, the functionality computes $s_i = |\{(v, x_v) \in S \mid x_v = i\}|$ for all $i \in \mathbb{Z}_m$. If neither of the voting authorities is corrupt, the functionality sends the tally $s_0, \ldots, s_{m-1}$ to the verifier and halts (this is a successful termination). Otherwise, it sends the tally, $s_0, \ldots, s_{m-1}$, to the adversary.

Announce $s'_0, \ldots, s'_{m-1}$ On receiving this command from the adversary, the functionality verifies that the **Tally** command was previously received. It then computes $d = \sum_{i=0}^{m-1} |s_i - s'_i|$. If $d < k$ (where k is the security parameter) it outputs the tally $s'_0, \ldots, s'_{m-1}$ 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 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.

We can now state our main theorem:

THEOREM 4.1. *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*.

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.

[3]This is a fairly common assumption in cryptographic voting protocols (appearing in [8, 4, 22, 9], among others).

Due to space constraints, we defer the proof of Theorem 4.1 to the full version of the paper.[4]

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 [16], an extension of Canetti and Gennaro's *incoercible computation* [6]. 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 [16], the adversary can force a party to abstain. We weaken this definition slightly, and allow the adversary to force a party to vote randomly (in most voting systems, a random vote is effectively the same as an abstention, so this is not much weaker). Under this definition:

THEOREM 4.2. *The Split-Ballot voting protocol is receipt-free, for any adversary that does not corrupt any of the voting authorities.*

Due to space constraints, we defer the formal proof of this theorem to the full version of the paper. However, the intuition behind it is apparent from the coercion-resistance strategy (described in Section 5.2).

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 de-

[4]An up to date version of the paper (with additional details) can be found in `http://www.wisdom.weizmann.ac.il/~naor/onpub.html`

scribe 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 $(\mathcal{C}, \cdot)$. 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_2 runs $KG^{(\mathcal{M})}$ and $KG^{(\mathcal{R})}$, producing $\left(pk^{(\mathcal{M})}, sk^{(\mathcal{M})}\right)$ and $\left(pk^{(\mathcal{R})}, sk^{(\mathcal{R})}\right)$ (which it sends over the private channel to A_1. 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_2 now runs P_K using the random beacon to replace the verifier (this produces a public proof that the commitment key was generated correctly).
2. Ballot preparation. Each voting authority prepares at least $2n$ ballots. Informally, each ballot contains commitments to the numbers 0 through $m-1$ in a random order (each number corresponds to a candidate). We identify a ballot by the tuple $\overline{w} = (a, i, b) \in \{0,1\} \times [n] \times \{0,1\}$, where A_a is the voting authority that generated the ballot, i is the index of the voter to whom it will be sent and b a ballot serial number. The ballot $B_{\overline{w}}$ consists of a random permutation $\pi_{\overline{w}}$: $\mathbb{Z}_m \mapsto \mathbb{Z}_m$ and a vector of commitments: $c_{\overline{w},\pi_{\overline{w}}(0)}, \ldots, c_{\overline{w},\pi_{\overline{w}}(m-1)}$, where

$$c_{\overline{w},\pi_{a,i}(j)} \doteq C\left(\pi_{\overline{w}}(j), r_{\overline{w},\pi_{\overline{w}}(j)}\right),$$

and $r_{\overline{w},0}, \ldots, r_{\overline{w},m-1} \in_R \mathcal{R}$ is a vector of m random values chosen by the authority.

5.2 Voting

The voter receives two ballots from each of the voting authorities. Denote the ballots received by voter $v \in \{1, \ldots, n\}$: $B_{1,v,0}$, $B_{1,v,1}$, $B_{2,v,0}$ and $B_{2,v,1}$, and 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. Each share is sent to a different authority (by choosing the corresponding commitment from the ballot). More formally:

1. The voter receives ballots $B_{1,v,0}$, $B_{1,v,1}$, $B_{2,v,0}$ and $B_{2,v,1}$ from the authorities and enters the "voting booth". The voter chooses, uniformly at random, two bits $b_{v,1}, b_{v,2} \in_R \{0,1\}$ and a value $t_v \in_R \mathbb{Z}_m$. The value t_v is one "secret share" of the vote, the other will be $x_v - t_v$. Bit $b_{v,a}$ determines which ballot received from authority A_a will be used for voting (the other is used only for verification). The voter opens ballots $B_{1,v,b_{v,1}}$ and $B_{2,v,b_{v,2}}$ and leaves the other two ballots sealed.
2. To vote, the voter selects $s_{1,v} \doteq c_{1,v,b_{v,1},t_v}$ (i.e., the commitment to t_v) and $s_{2,v} \doteq c_{2,v,b_{v,2},x_v-t_v}$ (the commitment to $x_v - t_v$, where $x_v - t_v$ is computed in $\mathbb{Z}_m$).[5]
3. The voter then physically deletes the description of $\pi_{1,v,b_{v,1}}$ from $B_{1,v,b_{v,1}}$ and the description of $\pi_{2,v,b_{v,2}}$ from $B_{2,v,b_{v,2}}$. After this step the voter "leaves the voting booth".
4. The voter sends $s_{1,v}$ to A_1 and $s_{2,v}$ to A_2 ("sending" the ballot can consist of running it through a scanner at the polling place). The voting authorities verify that the proper erasures were performed and that two of the ballots are still sealed.
5. The voter opens the two sealed (unvoted) ballots.
6. Authority A_1 publishes the tuple $(1, v, s_{1,v})$ on the bulletin board and authority A_2 publishes the tuple $(2, v, s_{2,v})$.
7. For $a \in \{1,2\}$, authority A_a publishes $B_{a,v,1-b_{v,a}}$ and $r_{a,v,1-b_{v,a},0}, \ldots, r_{a,v,1-b_{v,a},m-1}$ to the public bulletin board (i.e., it opens the commitments for the ballot that *wasn't* used to cast the vote).
8. The voter verifies that commitments for the voted ballots have been correctly published (they match the values sent in step 2), and that both unvoted ballots were correctly published in their entirety.

5.2.1 Coercion-Resistance Strategy. We assume the adversary cannot observe the voter between steps 1 and 3 of the voting phase (these steps are performed while the voter is "in the voting booth").

If the voter is coerced before step 1, the voter follows the adversary's strategy precisely, but uses random permutations 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 permutations or not. By using random permutations, the end result is that voter votes randomly (coercing a voter to vote randomly is an attack we explicitly allow).

If the voter is coerced at step 1 or later (after entering the voting booth), she follows the regular voting protocol in steps 1 through 3. Even if she is coerced before step 3, she lies to the adversary and pretends the coercion occurred at step 3 (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 chooses random permutations that are consistent with the fake input and her chosen commitments, and pretends these were the permutations revealed on the opened ballots. Using the example in Figure 2.1, if Sarah was trying to convince a coercer that she actually voted for John (instead of Thomas), she would choose randomly one of the options for John (e.g., C,H), then claim that the left permutation she saw had C as the fourth value and the right permutation had H on the second (ordering the others randomly). Note that if the adversary forces the voter to invalidate her ballot, she will do so (but this is a forced abstention, which we explicitly do not prevent).

[5]This selection can be implemented, for example, by having the voter mark a physical ballot with a pen (such as the method described in Section 2.2).

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, both authorities know $s_{1,1}, s_{2,1}, \ldots, s_{1,n}, s_{2,n}$ (this was published on the public bulletin board in the voting phase). Authority A_1 also knows $t_1, \ldots, t_n$, while authority A_2 knows $x_1 - t_1, \ldots, x_n - t_n$. Below, we give a simplified version of the tally protocol that reveals the sums x_i without a modular reduction by m. This may be sufficient when the number of voters is much larger than the number of candidates. At the cost of a few extra steps (masking the sums by adding random multiples of m), this small information leakage can be avoided. Note: the tally stage uses as subprotocols some zero-knowledge proofs. These are based on standard techniques, and we omit them here due to lack of space.

1. Authority A_1 computes, for all $1 \leq i \leq n$: $d_i \doteq s_{1,i} \cdot s_{2,i} = C\left(x_i, r_{1,i,b_{i,1},t_i} + r_{2,i,b_{i,2},x_i-t_i}\right)$. It chooses a random permutation $\sigma_1 \colon [n] \mapsto [n]$, random values $u_{1,1}, \ldots, u_{1,n} \in_R \mathcal{R}$ and publishes to the public bulletin board, for all $1 \leq i \leq n$: $d'_i \doteq d_{\sigma_1(i)} \cdot C(0, u_{1,i})$.
2. Authority A_1 proves in zero-knowledge that $d'_1, \ldots, d'_n$ is a valid shuffle of $d_1, \ldots, d_n$.
3. Authority A_2 sends, for all $1 \leq i \leq n$: $e_i^{(\mathcal{M})} \doteq E_{pk}(x_i - t_i)$ and $e_i^{(\mathcal{R})} = E_{pk}(r_{2,i,x_i-t_i})$ to A_1 (note that A_2 uses *its own* public-keys for encryption). It also proves in zero knowledge that $e_i^{(\mathcal{M})}$ is an encryption of a value in the range $\{0, \ldots, m-1\}$.
4. For all $1 \leq i \leq n$, authority A_1 sends over the private channel to A_2: $e'^{(\mathcal{M})}_i = e^{(\mathcal{M})}_{\sigma_1(i)} \cdot E_{pk}\left(t_{\sigma_1(i)}\right) = E_{pk}\left(x_{\sigma_1(i)}\right)$ and $e'^{(\mathcal{R})}_i = e^{(\mathcal{R})}_{\sigma_1(i)} \cdot E_{pk}\left(r_{1,\sigma_1(i),t_{\sigma_1(i)}} + u_{1,i}\right) = E_{pk}\left(r_{2,\sigma_1(i),x_{\sigma_1(i)}-t_{\sigma_1(i)}} + r_{1,\sigma_1(i),t_{\sigma_1(i)}} + u_{1,i}\right)$. (i.e. A_1 reconstructs the complete votes and commitment randomness using the homomorphic property of the encryption, then shuffles them in the same way it originally shuffled the commitments).
5. Authority A_2 chooses a random permutation $\sigma_2 \colon [n] \mapsto [n]$, random values $u_{2,1}, \ldots, u_{2,n} \in_R \mathcal{R}$ and publishes to the public bulletin board, for all $1 \leq i \leq n$: $d''_i \doteq d'_{\sigma_2(i)} \cdot C(0, u_{2,i})$.
6. Authority A_2 proves in zero-knowledge shuffle that $d''_1, \ldots, d''_n$ is a valid shuffle of $d'_1, \ldots, d'_n$.
7. For all $i \in [n]$, authority A_2 decrypts $e'^{(\mathcal{M})}_{\sigma_2(i)}$ and $e'^{(\mathcal{R})}_{\sigma_2(i)}$, getting the values $x_{\sigma_1(\sigma_2(i))}$ and $r_{2,\sigma_1(\sigma_2(i)),x_{\sigma_1(\sigma_2(i))}-t_{\sigma_1(\sigma_2(i))}} + r_{1,\sigma_1(\sigma_2(i)),t_{\sigma_1(\sigma_2(i))}} + u_{1,\sigma_2(i)}$, respectively. The decrypted values are the messages and randomness of the commitments $d'_1, \ldots, d'_n$. Authority A_2 can now compute the messages and randomness of the commitments $d''_1, \ldots, d''_n$ by itself: A_2 publishes to the public bulletin board $\xi_i \doteq x_{\sigma_1(\sigma_2(i))}$ and $\rho_i \doteq r_{2,\sigma_1(\sigma_2(i)),x_{\sigma_1(\sigma_2(i))}-t_{\sigma_1(\sigma_2(i))}} + r_{1,\sigma_1(\sigma_2(i)),t_{\sigma_1(\sigma_2(i))}} + u_{1,\sigma_2(i)} + u_{2,i}$. Note that $\xi_1, \ldots, \xi_n$ is equivalent (mod m) to a permutation of the votes.

5.4 Universal Verification and Output

The verification can be performed by anyone with access to the public bulletin board.

1. The verifier checks that the commitment key was generated correctly.
2. The verifier checks for all unvoted ballots that the opened commitments match the published permutations
3. The verifier computes the vector $d_1, \ldots, d_n$ (by performing the computation $d_i = s_{1,i} \cdot s_{2,i}$ with the published commitments $s_{1,i} \cdot s_{2,i}$)
4. The verifier checks that the HVZK proofs of commitment shuffle published in steps 2 and 6 of the tally phase are correct.
5. The verifier checks that $d''_i = C(\xi_i, \rho_i)$ for all $i \in [n]$.
6. If an error was found, the verifier outputs $\perp$. Otherwise, the verifier computes and outputs the tally $s_0, \ldots, s_{m-1}$, where $s_j = |\{i \in [n] \mid \xi_i \equiv j \pmod m\}|$.

6. 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 easily be extended to an arbitrary number of authorities. However, the main 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 permutation 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.

7. REFERENCES

[1] B. Adida and R. L. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In *WPES '06*, pages 29–40, 2006.

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

[3] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *STOC '94*, pages 544–553, 1994.

[4] J. W. Bryans and P. Y. A. Ryan. A simplified version of the Chaum voting scheme. Technical Report CS-TR 843, University of Newcastle, 2004.

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

[6] R. Canetti and R. Gennaro. Incoercible multiparty computation. In *FOCS '96*, pages 504–513.
[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.
[8] D. Chaum. E-voting: Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, Jan./Feb. 2004.
[9] D. Chaum, 2006. `http://punchscan.org/`.
[10] J. D. Cohen(Benaloh) and M. J. Fischer. A robust and verifiable cryptographically secure election scheme. In *FOCS '85*, pages 372–382.
[11] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *Eurocrypt '96*, pages 72–83.
[12] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Eurocrypt '97*, pages 103–118.
[13] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT '92*, pages 244–251.
[14] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Eurocrypt '00*, pages 539+.
[15] T. Moran and M. Naor. Basing cryptographic protocols on tamper-evident seals. In *ICALP '05*, pages 285–297.
[16] T. Moran and M. Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO*, pages 373–392, 2006. `http://www.wisdom.weizmann.ac.il/~talm/papers/MN06-voting.pdf`.
[17] M. Naor and A. Shamir. Visual cryptography. In *Eurocrypt '94*, volume 950 of *LNCS*, pages 1–12, 1995.
[18] C. A. Neff. Practical high certainty intent verification for encrypted votes, October 2004. `http://www.votehere.net/vhti/documentation/vsv-2.0.3638.pdf`.
[19] S. Popoveniuc and B. Hosp. An introduction to punchscan, 2006. `http://punchscan.org/papers/popoveniuc_hosp_punchscan_introduction.pdf`.
[20] M. O. Rabin. Transaction protection by beacons. *J.Computer and System Sciences*, 27(2):256–267, 1983.
[21] D. J. Reynolds. A method for electronic voting with coercion-free receipt. Presentation: `http://www.win.tue.nl/~berry/fee2005/presentations/reynolds.ppt`.
[22] P. Y. A. Ryan. A variant of the Chaum voter-verifiable scheme. In *WITS '05*, pages 81–88, 2005.
[23] A. Shamir. Cryptographers panel, RSA conference, 2006. Webcast: `http://media.omediaweb.com/rsa2006/1_5/1_5_High.asx`.

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, where the plaintext is in the group $\mathbb{Z}_n$ for $n = p_1 p_2$, a product of two safe primes. For the commitment scheme, we propose a modified version of the Pedersen commitment scheme where both messages and randomness are also in the group $\mathbb{Z}_n$. 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, $g^m h^r$ is a random group element when r is chosen at random. 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. Using a random beacon, this can be done securely, but we leave the details to the full version of the paper.