Receipt-Free Universally-Verifiable Voting With Everlasting Privacy*

Tal Moran[1] and Moni Naor[1]**

Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel

Abstract. We present the first universally verifiable voting scheme that can be based on a general assumption (existence of a non-interactive commitment scheme). Our scheme is also the first receipt-free scheme to give "everlasting privacy" for votes: even a computationally unbounded party does not gain any information about individual votes (other than what can be inferred from the final tally).
Our voting protocols are designed to be used in a "traditional" setting, in which voters cast their ballots in a private polling booth (which we model as an untappable channel between the voter and the tallying authority). Following in the footsteps of Chaum and Neff [7,16], our protocol ensures that the integrity of an election cannot be compromised *even if the computers running it are all corrupt* (although ballot secrecy may be violated in this case).
We give a generic voting protocol which we prove to be secure in the Universal Composability model, given that the underlying commitment is universally composable. We also propose a concrete implementation, based on the hardness of discrete log, that is slightly more efficient (and can be used in practice).

Keywords: receipt-free voting human protocol universally composable formal proof

1 Introduction

One of the earliest secret election protocols is the source of the word "ostracism": when the citizens of ancient Athens believed a politician had too much power, they held a vote to determine if he should be exiled for a period of ten years. The vote was conducted by having each citizen write the name of the person he most hated on a pot shard (these shards were called *ostraca*) and place it in a vase. After the votes were cast, the shards were taken out of the vases and counted. Many modern voting systems follow a very similar protocol, replacing clay with paper and vases with ballot boxes.

* This work was partially supported by the Minerva Foundation

** Incumbent of the Judith Kleeman Professorial Chair

1.1 Challenges in Designing Voting Protocols

One of the main problems with traditional systems is that the accuracy of the election is entirely dependent on the people who count the votes. In modern systems, this usually consists of fairly small committees: if an entire committee colludes, they can manufacture their own results. Even worse, depending on the exact setup it may be feasible to stuff ballot boxes, destroy votes or perform other manipulations.

The problems with assuring election integrity were a large factor in the introduction of mechanical voting machines, and more recently, optical scan and "Direct Recording Electronic" (DRE) machines. These perform a function identical to a ballot box and paper ballots, using a different medium: the basic protocol remains the same. While alleviating some of the problems (such as ballot stuffing), in some cases they actually aggravate the main one: instead of relying on a large number of election committees (each of which has a limited potential for harm), their security relies on a much smaller number of programmers (who may be able to undetectably change the results of the entire election).

There has also been a large amount of more theoretical research, aimed at using cryptographic tools to define and solve the problems inherent in conducting secure elections. The most important advantage of cryptographic voting protocols over their physical counterparts is the potential for *universal verifiability*: the possibility that every voter (and even interested third parties) can verify that the ballot-counting is performed correctly. The challenge, of course, is satisfying this property while still maintaining the secrecy of individual ballots.

A problem that was first introduced with mechanical voting machines, and exacerbated in DRE and many cryptographic systems, is that part of the protocol must be performed by a machine (or computer), whose inner workings are opaque to most voters. This can have a serious impact on the trust a voter places in the results of the election (e.g., "how do I know that when I pushed the button next to candidate A the machine didn't cast a vote for B?"). One of the targets recently identified in the cryptographic literature is to design systems that can be trusted by human voters *even if the election computers are running malicious code*.

Another attack on both traditional and cryptographic voting systems is vote-buying and coercion of voters. To prevent this, we would like a voter to be unable to convince a third party of her vote *even if she wants to do so*. This property, called *receipt-freeness*, is strictly stronger than ballot secrecy, and seems even harder to achieve simultaneously with universal-verifiability. As is the case for election integrity, it is much more difficult to design a receipt-free protocol if the voter is required to perform secret calculations on a computer: the voter may be forced to use an untrusted computer to perform the calculations (or even one provided by the coercer), in which case the coercer can learn the secret.

There are also problems specific to the cryptographic case. One of these is that cryptographic protocols are often based on computational assumptions (e.g., the infeasibility of solving a particular problem). Unfortunately, some computational assumptions may not hold forever (e.g., Adi Shamir estimated that existing public-key systems will remain secure for less than thirty years [21]).

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's 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 et al. [1], we can say that information-theoretic privacy is *everlasting* privacy.

A related problem is that we would like to base our voting schemes on assumptions that are as strong as possible. Existing voting schemes generally require public-key encryption (or very specific computational assumptions, such as the hardness of computing discrete log in certain groups).

1.2 Our Results

In this paper, we present the first universally verifiable voting scheme that can be based on a general assumption (existence of a non-interactive commitment scheme).

Our protocol also satisfies the following properties:

- It has everlasting privacy (provided the commitment scheme is statistically hiding). To the best of our knowledge, only one published protocol has this property [9], and this protocol is not receipt-free.
- The protocol does not require human voters to perform any complex operations (beyond choosing a random string and comparing two strings)
- The integrity of the election is guaranteed even if the DRE is corrupted.
- It is receipt-free. We use a technique from Neff's voting scheme [16] to achieve receipt-freeness without requiring complex computation on the voter's part.

We give a rigorous proof that our protocol is secure in the Universally Composable model (given a universally-composable commitment scheme). This is a very strong notion of security. We also give a slightly more efficient protocol based on Pedersen Commitments (this protocol is secure, but not in the UC model, since Pedersen Commitments are not UC secure).

An additional contribution of this paper is a formal definition of receipt-freeness in the general multi-party computation setting (we also prove that our protocol satisfies this definition). Our definition is a generalization of Canetti and Gennaro's definition for an incoercible computation [4]. To the best of our knowledge, this is the first definition to capture receipt-freeness in the general case (most previous papers that deal with receipt-freeness do not provide a formal definition at all).

1.3 Previous Work on Voting Protocols

The first published electronic voting scheme was proposed by Chaum [5], based on *mixes*. Loosely speaking, a *mix* is a device that hides the correspondence between its inputs and outputs by accepting (encrypted) inputs in large batches and mixing them before output. This can be used to hide the correspondence between voters and their votes, allowing each voter to make sure her vote was counted (ensuring the integrity of the election) while preserving the secrecy of the vote. A strong advantage of this scheme over previous voting systems (e.g., putting paper slips in ballot boxes) is that the integrity of the vote no longer rests in the hands of a few trustees: every voter can verify that their vote was counted (i.e. it has *individual* verification). The *privacy* of the votes does depend on a small number of trustees (the mixing centers), though. Other advantages are convenience and speed: a voter can vote from any location with network access, and the votes are tabulated by computers immediately after they were all cast.

Many additional protocols were suggested since Chaum's. Almost all can be classified into one of three categories:

Mix-Type These are protocols based on mixes, such as Chaum's original protocol.

Blind Signatures These are protocols based on "blind signatures", introduced by Chaum in [6]. A blind signature allows a signer to digitally sign a document without knowing what was signed. In a voting scheme based on blind signatures, the general idea is that the voter has her ballot blindly signed by the voting authority, and later publishes the ballot using an anonymous channel. Although Chaum suggested the use of blind signatures for voting in his original paper, the first published protocol that makes use of blind signatures was by Fujioka et al. [11]. A major problem of blind signature schemes is that they require *anonymous channels* (so that the voter can publish her signed vote linking the vote to the voter).

Homomorphic A function E is homomorphic if for any x and y in its domain it satisfies $E(x)E(y) = E(x+y)$. The general idea of a homomorphic voting scheme is for each voter to encrypt her vote using a public-key homomorphic encryption function, where the public key is published before the election. Each voter must prove that her encrypted vote is an encryption of a *valid* vote (the voting schemes differ on the exact way in which this is done). The encrypted votes are summed using the homomorphic property of the encryption function (without decrypting them). Finally, a set of trustees cooperate to decrypt the final tally (the secret key for the encryption scheme is divided between the trustees). The advantages of using homomorphic schemes are efficiency and verifiability: many operations can be carried out on the encrypted votes, in public, so they are both verifiable and can be performed during the voting process (without interaction between the voting authorities). The first protocol of this type was devised by Cohen (Benaloh) and Fischer [8]. Additional examples of this type of scheme are [2,9,10,12].

Receipt-Free Voting Only a small fraction of the proposed voting schemes satisfy the property of receipt-freeness. Benaloh and Tuinstra [2] 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 [12]). Their protocol was based on homomorphic encryption rather than mixes. To satisfy receipt-freeness, Benaloh and Tuinstra also required a physical "voting booth": completely untappable channels between the voting authority and the voter. Sako and Kilian showed that a one-way untappable channel is enough [20], and gave a receipt-free mix-type voting scheme based on this assumption (our protocol makes this assumption as well). Other protocols were also devised, however the minimal assumption required by protocols that do not use a trusted third party device (e.g., a smart card) is the one-way untappable channel.

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 actually casting the ballot on her behalf is accurately reflecting her intentions. Chaum [7], and later Neff [16], proposed universally-verifiable receipt-free voting schemes that overcome this problem. Recently, Reynolds proposed another protocol similar to Neff's [18].

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 DRE. 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 a truly an encryption of her intended vote. After all the votes have been cast, the votes are shuffled and decrypted using mix-type schemes.

Chaum's protocol uses a two-part ballot. Together, both parts define the vote in a manner readable to a human. Either part separately, however, contains only an encrypted ballot. The voter chooses one part at random (after verifying that the ballot matches her intended choice), and this part becomes her receipt. The other part is destroyed. The ballots are constructed so that in an invalid ballot, at least one of the two parts must be invalid (and so with probability at least $\frac{1}{2}$ this will be caught at the tally stage). Chaum's original protocol used Visual Cryptography [15] to enable the human voter to read the complete (two-part) ballot, and so required special printers and transparencies. Bryans and Ryan showed how to simplify this part of the protocol to use a standard printer [3,19].

Neff's protocol makes ingenious use of zero-knowledge arguments. The idea is that a zero-knowledge argument system has a simulator that can output "fake" proofs indistinguishable from the real ones. The DRE performs an interactive zero-knowledge protocol with the voter to prove that the encrypted ballot corresponds to the correct candidate. The DRE uses the simulator to output a zero-knowledge proof for *every other* candidate. The proofs are of the standard "cut-and-choose" variety. In a "real" proof, the DRE commits, then the voter gives a random challenge and the DRE responds. In the "fake" proofs, the voter first gives the random challenge and then the DRE commits and responds. The voter only has to make sure that she gave the challenge for the real candidate *after* the DRE was committed, and that the challenge printed on the receipt matches what she gave. Everything else can be publicly checked outside the voting booth. Since no one can tell from the receipt in which order the commitments and challenges were made, the zero-knowledge property ensures that they cannot be convinced which of the proofs is the real one.

2 The Model

The physical setup of our system is very similar to many existing (non-electronic) voting schemes. Voters cast their ballots at polling stations. The votes are tabulated for each station separately, and the final tally is computed by summing the results for all stations.

2.1 Basic Assumptions

Human Capability An important property of our protocol is that its security is maintained even if the computers running the elections are corrupt (and only some of the human voters remain honest). Thus, we must define the operations we expect a human to perform. We make three requirements from human voters:

1. They can send messages to the DRE (e.g., using a keyboard). We require voters to send a few short phrases. This should be simple for most humans (but may be a slight problem for disabled voters).
2. They can verify that two strings are identical (one of which they chose themselves)
3. They can choose a random string. This is the least obvious of the assumptions we make. Indeed, choosing truly uniformly random bits is probably not something most humans can do. However, using physical aids (coins, dice, etc.) and techniques for randomness extraction it may be possible. In our security proofs, in order to clarify the presentation, we will ignore these subtleties and assume the voters can choose uniformly random strings.

Physical Commitment In order to achieve receipt-freeness, our protocol requires a commitment with an extremely strong hiding property: The verifier's view at the end of the commit stage is a deterministic function of her view before the commit stage (i.e., not only can the view not contain any information about the committed string, it cannot even contain randomness added by the committer). Such a commitment is not possible in the "plain" cryptographic model (even with computational assumptions), but can be easily implemented by physical means (for example, by covering part of the printer's output with an opaque shield, so that the voter can see that something has been printed but not what). Note that the security of the protocol does not depend on physical commitment, only the receipt-freeness.

2.2 Participating Parties

In our description, we consider only a single polling booth (there is no interaction between booths in our system, apart from the final, public summation of results). Formally, we consider a few classes of participants in our voting protocol:

Voters There are an arbitrary number of voters participating in the protocol (we will denote the number of voters by n). Each voter has a secret input (the candidate she wants to vote for).

DRE The protocol has only a single DRE party. The DRE models the ballot box: it receives the votes of all the voters and outputs the final tally at the end.

Verifier A Verifier is a party that helps verify that the voting protocols are being followed correctly. Although there can be many verifiers (and voters can be verifiers as well) the verifiers are deterministic and use only public information, so we model them as a single party.

Adversary The adversary attempts to subvert the voting protocol. We detail the adversarial model in Sections 2.4 and 2.5.

2.3 Protocol Structure and Communication Model

As is customary in universally-verifiable voting protocols, we assume the availability of a public *Bulletin Board*: a broadcast channel with memory. All parties can read from the board and all messages from the DRE are sent to the bulletin board.

Our voting protocols consist of three phases:

1. *Casting a Ballot.* In this phase, each voter communicates directly with the DRE over a private, untappable, channel (inside the voting booth). All communication from the DRE to the voter is through the bulletin board.
   In practice, the voter will not be able to access the bulletin board while in the voting booth. Thus, we assume there is a separate channel between the DRE and the voter, also with memory (e.g., a printer). The DRE outputs its messages both to the printer (the printed messages form the voter's receipt), and to the bulletin board. This implementation adds an additional stage in which the voter verifies that the contents of her receipt match the contents on the bulletin board.
   We need the physical commitment only at one point in the Ballot-Casting phase. One possible implementation is by a shield covering part of the printer's output (so that the voter can see something has been printed under the shield, but not what).

2. *Tallying the results*. This phase begins after all voters have cast their ballots, and in this phase the results of the vote are revealed. The tallying consists of an interactive protocol between the DRE and a random beacon, whose only messages are uniformly random strings. In practice, we can use the Fiat-Shamir heuristic, replacing the beacon's messages with a secure hash of the entire transcript to that point (in the analysis the hash function is modeled as a random oracle). The entire transcript of the tally phase is sent to the bulletin board.
3. *Universal Verification*. This phase consists of verifying the consistency of the messages on the bulletin board. This can be performed by any interested party, and is a deterministic function of the information on the bulletin board.

2.4 Universal Composability

We consider a number of different adversarial models. In the basic model, the adversary can adaptively corrupt the DRE and the voters (since there are arbitrarily many verifiers, and all are running the same deterministic function of public information, it is reasonably to assume not all of them can be corrupted). We formalize the capabilities of the adversary by defining them in the Universally Composable model, using the ideal voting functionality, $\mathcal{F}_V$. In the ideal world, The DRE does nothing unless it is corrupted. When the DRE is corrupt, ballots no longer remain secret (note that this must be the case in any protocol where the voter divulges her vote to the DRE). The integrity of the vote is always maintained. The verifiers have no input, but get the final output from the functionality (or $\perp$ if the functionality was halted by the adversary).

The adversary is allowed to intercept every outgoing message from the functionality, and has the choice of either delivering the message or sending the **Halt** command to the functionality (in which case the message will not be received by other parties).

The ideal functionality realized by our voting scheme accepts only one "honest" command and one "cheating" command (beyond the special **Halt** and **Corrupt** commands that can be sent by the adversary)

Vote c On receiving this command from voter v, the functionality verifies that this is the only **Vote** command sent by v. It then:
1. Increments counter c.
2. If the DRE is corrupt, the functionality outputs **Voted** v, c to the adversary.
3. Broadcasts the message **Voted** v.
4. If all n voters have sent a **Vote** command, the functionality outputs the final value of the counters to the verifiers.

ChangeVote c, c' This command can only be sent by a corrupt voter v and only if the DRE is also corrupt. On receiving this command, the functionality verifies that a **Vote** c command was previously sent by v. It then decrements counter c and increments counter c'. This command can be sent after the last voter has voted and before the final tally is output.

The security, privacy and robustness of our protocol is proven by showing that any attack against the protocol in the real world can be performed against the ideal functionality in the ideal world (where the possible attacks are explicitly defined). Due to space contraints, the formal description of the protocol and proof of security will appear only in the full version.

2.5 Receipt-Freeness

The property of receipt-freeness is not adequately captured by the UC model. In this model, in addition to corrupting parties, the adversary can *coerce* parties. A coercion attack models the real-life scenario in which voters are bribed or threatened to act according the adversaries wishes. The adversary can interrogate coerced parties and give them commands, but does not completely control them (a formal definition of receipt freeness can be found in Section 4). When considering the receipt-freeness of our protocol, we do not allow the adversary to coerce or corrupt the DRE. The latter is because corrupting the DRE reveals the voter's inputs, and so the protocol is trivially coercible. The former is because the DRE is a machine, so it does not make sense to bribe or threaten it.

It may make sense to coerce or corrupt the DRE's *programmers*, however. The difference between this situation and a corrupt DRE is that a corrupt DRE can communicate freely with the adversary, while a "maliciously programmed" DRE can communicate with the adversary only through the public communication channel (in one direction) and the voter's input (in the other direction). We briefly discuss this problem in Section 5.

2.6 Timing Attacks

Like any cryptographic proof, the security of our protocol is guaranteed only as far as the real-world matches the model on which our proof is based. One point we think is important to mention is the "timing" side-channel. Our model does not specify timing information for messages appearing on the bulletin board — only the order of the messages. However, in a real life implementation it may be possible to time the messages sent by the DRE. If the DRE actually does send messages simultaneously to the bulletin board and the voter, this timing information can be used to determine the voter's input (since the time it takes the voter to respond will be different). To prevent this attack, we specify that the DRE sends its output to the bulletin board only after the voter leaves the booth. One possible implementation (that also guarantees that the DRE can't leak information using timing, is that the DRE is not connected to a network at all. Instead, it prints the output to be sent to the bulletin board. The printout is given by the voter to the election officials on exiting the booth, who can scan it and upload the information to the bulletin board.

3 Informal Protocol Description

3.1 Overview

At the highest level, our voting scheme is extremely simple: the voter enters the voting booth and selects a candidate. The DRE uses a statistically-hiding commitment scheme to publicly commit to the candidate (e.g., by posting the commitment on a public bulletin board). It then proves privately to the voter that the commitment is really to the voter's candidate. After all voters have cast their ballots, the DRE publishes the final tally. It then proves, using a zero knowledge proof of knowledge, that the tally corresponds to the commitments published for each voter.

Since we know how to construct a ZK proof of knowledge for any NP language, and in particular we can construct such a proof system for any string commitment scheme, it would appear that we could use any such system for the private proof (the one that convices the voter that her ballot is being cast as she intended). The zero-knowledge property would ensure that the voter cannot use the proof to convince any third party of her vote.

The problem is that the voter is human, and the general zero-knowledge proof systems require complex computations that are infeasible to perform without the help of computers. Since the scheme must remain secure even if the DRE is malicious, the voter cannot trust the DRE to make these calculations. Allowing voters to use their own computers is not much better. Most voters do not know how to verify that their computer is actually running the correct code. Even worse, a coercive adversary could require a voter to use a computer supplied by the adversary, in which case it could easily learn the identity of the voter's candidate.

Our solution is that used in Neff's voting scheme: Neff observed that a standard cut-and-choose zero knowledge proof of some statement S has the following structure: the prover commits to two proofs P_0, P_1, the verifier makes a choice b, the prover reveals proof P_b and finally the verifier makes sure the revealed proof is correct. The protocol is constructed so that if both P_0 and P_1 are valid proofs, the statement S holds but, given b, anyone can construct a pair P'_0, P'_1 so that P'_b is correct (even if S does not hold). The insight is that a human can easily make the choice b without the aid of a computer. To keep the proof private, the DRE constructs a *fake* proof for all the other candidates by running the ZK simulator. The only difference between the real and fake proofs in this case is that in the real proof the DRE first commits, and then the voter chooses b, while in the fake proof the voter reveals b before the DRE commits. This *temporal* information cannot be inferred from the receipt. However, since the receipt is public, anyone can check (using a computer) that both P_b and P'_b are valid proofs. Even if the voter does not trust her own computer, it is enough that someone with a good implementation of the verification algorithm perform the check.

In the following subsections we present a concrete implementation of our generic protocol. This implementation is based on Pedersen commitments. In Section 3.2 we describe an example of a hypothetical voter's interaction with the system. Section 3.3 goes behind the scenes, and describes what is actually occurring during this session (as well as giving a brief description of the protocol itself).

3.2 A Voter's Perspective

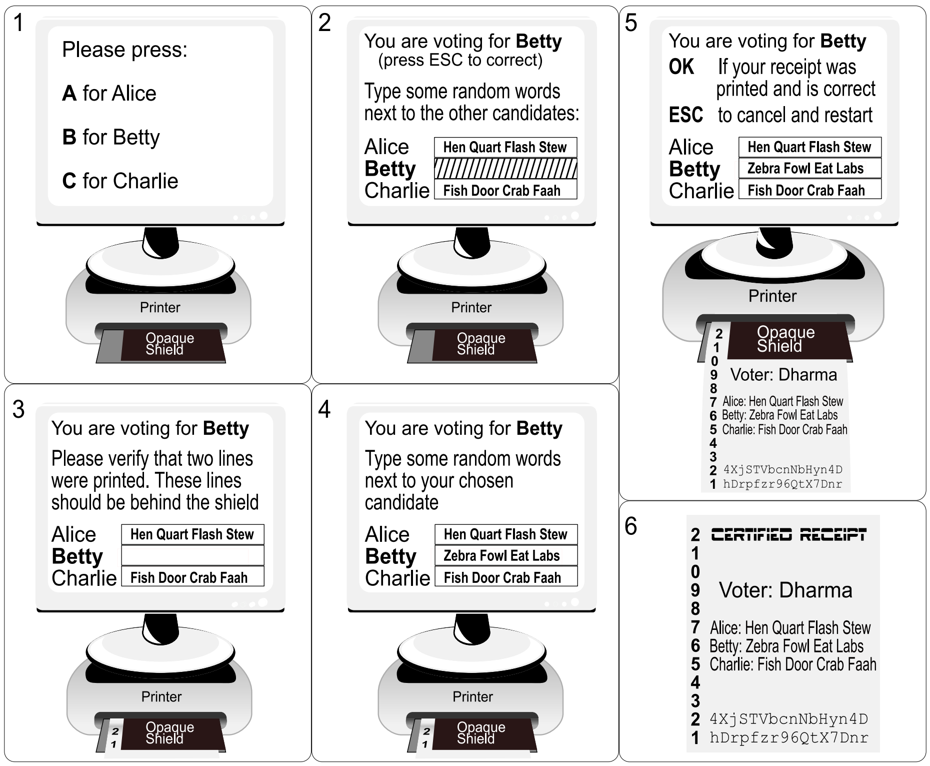

Fig. 3.1. Ballot for Betty

Figure 3.1 shows what Dharma, our hypothetical voter, would see while casting her ballot in an election between candidates Alice, Betty and Charlie. Dharma identifies herself to the election official at the entrance to the polling station and enters the booth.

Inside the booth is the DRE: a computer with a screen, keyboard and an ATM-style printer

1. The screen presents the choice of candidates: "Press *A* for Alice, *B* for Betty and *C* for Charlie". Dharma thinks Betty is the best woman for the job, and presses *B*.
2. The DRE now tells Dharma to enter some random words next to each of the other candidates (Alice and Charlie). For an even simpler experience, the DRE can prefill the random words, and just give Dharma the option of

changing them if she wants. At any time until the final stage, Dharma can change her mind by pressing **ESC**. In that case, the DRE spits out whatever has been printed so far (this can be discarded), and returns to stage 1.

3. The DRE prints two rows. The actual printed information is hidden behind a shield, but Dharma can verify that the two rows were actually printed.
4. Dharma enters a random challenge for her chosen candidate.
5. The DRE prints out the rest of the receipt. Dharma verifies that the challenges printed on the receipt are identical to the challenges she chose. If everything is in order, she presses **OK** to finalize her choice. If something is wrong, or if she changed her mind and wishes to vote for a different candidate, she presses **ESC** and the DRE returns to stage 1.
6. The DRE prints a "Receipt Certified" message on the final line of the receipt. Dharma takes her receipt and leaves the voting booth. At home, she verifies that the public bulletin board has an exact copy of her receipt, including the two lines of "gibberish" (the bulletin board can be viewed from the internet). Alternatively, she can give her receipt to an organization she trusts (e.g., "Betty's Popular People's Front"), who will perform this verification for her.

After all the voters have cast their ballots, the protocol moves to the Final Tally phase. Voters are not required to participate in this phase — it consists of a single broadcast from the DRE to the public bulletin board (here we assume we are using the Fiat-Shamir heuristic to make the final tally non-interactive). Note that we do not actually require the DRE to be connected to a network. The DRE can store its output (e.g., on a removable cartridge). After the DRE has written the final tally message, the contents of the cartridge can be uploaded to the internet. Anyone tampering with the cartridge would be detected.

Anyone interested in verifying the election results participates in the Universal Verification phase. This can include voters, candidates and third parties. Voters do not have to participate, as long as they made sure that a copy of their receipt appears on the bulletin board, and they trust that at least one of the verifying parties is honest.

Receipt-Freeness To get an intuitive understanding for why this protocol is receipt-free, suppose Eve tries to bribe Dharma to vote for Alice instead. There are only two things Dharma does differently for Alice and Betty: she presses B in the first step, and she fills in Alice's random words before the DRE prints the first two lines of the receipt, while filling in Betty's afterwards. Eve has no indication of what Dharma pressed (since the receipt looks the same either way). The receipt also gives no indication in what order the candidate's words were filled (since the candidates always appear in alphabetical order). Because the first two lines of the receipt are hidden behind the shield when Dharma enters the challenge for her chosen candidate, she doesn't gain any additional information as a result of filling out the challenges for Alice and Charlie; so whatever Eve asks her to do, she can always pretend she filled out the challenge for Alice after the challenge for Betty.

3.3 Behind the Scenes: An Efficient Protocol Based on the Discrete Log Assumption

Pedersen Commitments The concrete protocol we describe in this section is based on Pedersen commitments [17]; statistically hiding commitments whose security is based on the hardness of discrete-log. We briefly describe the Pedersen commitment, assuming discrete log is hard in some cyclic group G of order q, and $h, g \in G$ are generators such that the committer does not know $\log_g h$. To commit to $a \in \mathbb{Z}_q$, the committer chooses a random element $r \in \mathbb{Z}_q$, and sends $h^a g^r$. Note that if the committer can find $a' \neq a$ and r' such that $h^{a'} g^{r'} = h^a g^r$, then $h^{a'-a} = g^{r-r'}$, and so the commiter can compute $\log_g h = \frac{r-r'}{a'-a}$ (in contradiction to the assumption that discrete log is hard in G). Therefore the commitment is computationally binding. If r is chosen uniformly at random from $\mathbb{Z}_q$, then g^r is a uniformly random element of G (since g is a generator), and so for any a the commitment $h^a g^r$ is also a uniformly random element of G. So the commitment is perfectly hiding.

We'll now show what happened behind the scenes, assuming the parameters G, h and g of the Pedersen commitment scheme are decided in advance and known to all parties. For simplicity, we also assume a collision-resistant hash function $H : \{0,1\}^* \mapsto \mathbb{Z}_q$. This allows us to commit to an arbitrary string $a \in \{0,1\}^*$ by committing to $H(a)$ instead. Denote $P(a,r) = h^{H(a)} g^r$. To open $P(a,r)$, the committer simply sends a, r.

The security of the scheme depends on a security parameter, k. The probability that the DRE can change a vote without being detected is $2^{-k+1} + O(nk\epsilon)$, where n is the number of voters and ϵ is the probability of successfully breaking the commitment scheme.

Casting the Ballot. We'll go over the stages as described above.

1. (Dharma chose Betty). The DRE computes a commitment: $v = P(\text{Betty}, r)$ (where r is chosen randomly), and prepares the first step in a proof that this commitment is really a commitment to Betty. This step consists of computing, for $1 \leq i \leq k$, a "masked" copy of v: $b_i = vg^{r_{B,i}} = P(\text{Betty}, r + r_{B,i})$, where $r_{B,i}$ is chosen randomly.
2. (Dharma enters fake challenges). The DRE translates each challenge to a k bit string using a predetermined algorithm (e.g., by hashing). Let A_i be the i^{th} bit of the challenge for Alice. For each bit i such that $A_i = 0$ the DRE computes a commitment to Alice: $a_i = P(\text{Alice}, r_{A,i})$, while for each bit such that $A_i = 1$ the DRE computes a masked copy of the real commitment to Betty: $a_i = vg^{r_{A,i}}$. Note that $a_i = P(\text{Betty}, r + r_{A,i})$. The set of commitments $a_1, \ldots, a_k$ will form a fake proof that v is a commitment to Alice (we'll see why we construct them in this way in the description of universal verification phase step 3.3). The DRE also computes $c_1, \ldots, c_k$ in the same way for Charlie.
   The DRE now computes a commitment to everything it has calculated so far: $x = P([v, a_1, \ldots, a_k, b_1, \ldots, b_k, c_1, \ldots, c_k], r_x)$. It prints x on the receipt (this is what is printed in the first two lines).

3. (Dharma enters the real challenge) The DRE translates this challenge into a k bit string as in the previous step. Denote B_i the i^{th} bit of the real challenge.
4. (The DRE prints out the rest of the receipt). The DRE now computes the answers to the challenges: For every challenge bit i such that is $X_i = 0$ (where $X \in \{A, B, C\}$), the answer to the challenge is $s_{X,i} = r_{X,i}$. For $X_i = 1$, the answer is $s_{X,i} = r + r_{X,i}$. The DRE stores the answers. It then prints the candidates and their corresponding challenges (in alphabetical order), and the voter's name (Dharma).
5. (Dharma accepts the receipt). The DRE prints a "Receipt Certified" message on the final line of the receipt. (the purpose of this message is to prevent voters from changing their minds at the last moment, taking the partial receipt and then claiming the DRE cheated because their receipt does not appear on the bulletin board). It then sends a copy of the receipt to the public bulletin board, along with the answers to the challenges and the information needed to open the commitment x: $(s_{A,1}, \ldots, s_{A,k}, s_{B,1}, \ldots, s_{B,k}, s_{C,1}, \ldots, s_{C,k})$ and $([v, a_1, \ldots, a_k, b_1, \ldots, b_k, c_1, \ldots, c_k], r_x)$.

Final Tally. The DRE begins the tally phase by announcing the final tally: how many voters voted for Alice, Betty and Charlie. Denote the total number of voters by n, and $v_i = P(X_i, r_i)$ the commitment to voter i's choice (X_i) that was sent in the Ballot Phase. The DRE now performs the following proof k times:

1. The DRE chooses a random permutation π of the voters, and n "masking numbers" $m_1, \ldots, m_n$. It then sends the permuted, masked commitments of the voters:
   $v_{\pi(1)} g^{m_{\pi(1)}}, \ldots, v_{\pi(n)} g^{m_{\pi(n)}}$
2. The random beacon sends a challenge bit b
3. If $b = 0$, the DRE sends π and $m_1, \ldots, m_n$ (unmasking the commitments to prove it was really using the same commitments it output in the Ballot-Casting phase). If $b = 1$, the DRE opens the masked commitments (without revealing π, the correspondence to the original commitments). It sends: $(X_{\pi(1)}, r_{\pi(1)} + m_{\pi(1)}), \ldots, (X_{\pi(n)}, r_{\pi(n)} + m_{\pi(n)})$

Universal Verification (and security proof intuition). The purpose of the universal verification stage is to make sure that the DRE sent well-formed messages and correctly opened all the commitments. For the messages from the Ballot-Casting phase, the verifiers check that:

1. $x = P([v, a_1, \ldots, a_k, b_1, \ldots, b_k, c_1, \ldots, c_k], r_x)$ This ensures that the DRE committed to v and $b_1, \ldots, b_k$ (in Dharma's case) before Dharma sent the challenges $B_1, \ldots, B_k$ (because x was printed on the receipt before Dharma sent the challenges).
2. For every commitment x_i (where $x \in \{a, b, c\}$), its corresponding challenge X_i, and the response $s_{X,i}$, the verifiers check that x_i is a good commitment to X when $X_i = 0$ (i.e., $x_i = P(X, s_{X_i})$) and that x_i is a masked version of v if $X_i = 1$ (i.e., $vs_{X_i} = x_i$). Note that if x_i is both a masked version of v *and* a good commitment to X, then v must be a good commitment to X

(otherwise the DRE could open v to two different values, contradicting the binding property of the commitment). This means that if v is a commitment to some value other than the voter's choice, the DRE will be caught with probability at least $1 - 2^{-k}$: every commitment x_i can be either a good masked version of v or a good commitment to X, but not both. So for each of the k challenges (which are not known in advance to the DRE), with probability $\frac{1}{2}$. The DRE will not be able to give a valid response.

For the final tally phase, the verifiers also check that all commitments were opened correctly (and according to the challenge bit). As in the Ballot-Casting phase, if the DRE can correctly answer a challenge in both directions (i.e., the commitments are a permutation of good masked versions of commitments to the voter's choices, and also when opened they match the tally), then the tally must be correct. So the DRE has probability at least $\frac{1}{2}$ of getting caught for each challenge if it gave the incorrect tally. If the DRE wants to change the election results, it must either change the final tally, change at least one of the voter's choices or break the commitment scheme. Since the protocol uses $O(nk)$ commitments (note that cheating on the commitments in the fake proofs doesn't matter), the total probability that it can cheat is bounded by $2 \cdot 2^{-k} + O(nk\epsilon)$.

Protocol Complexity. We can consider both the time and communication complexity of the protocol. In terms of time complexity, the DRE must perform $O(knm)$ commitments in the Ballot Casting phase (where m is the number of candidates), and $O(kn)$ commitments in the Final Tally phase (the constants hidden in the O notation are not large in either case). Verifiers have approximately the same time complexity (they verify that all the commitments were opened).

The total communication complexity is also of the same order. In this case, the important thing is to minimize the DRE's communication to the voter (since this must fit on a printed receipt). Here the situation is much better: the receipt only needs to contain a single commitment and the challenges sent by the voter (each challenge has k bits). Note that we do not use any special properties of the commitment on the receipt (in practice, this can be the output of a secure hash function rather than a Pedersen commitment).

3.4 Using Generic Commitment

The protocol we described above makes use of a special property of Pedersen commitment: the fact that we can make a "masked" copy of a commitment. The essence of our zero knowledge proof is that on the one hand, we can prove that a commitment is a masked copy of another without opening the commitment. On the other hand, just by seeing two commitments there is no way to tell that they are copies, so opening one does not give us information about the other.

Our generic protocol uses the same idea, except that we implement "masked copies" using an arbitrary commitment scheme. The trick is to use a double commitment. Denote $C(a, r)$ a commitment to a with randomness r.

The idea is to create all the "copies" in advance (we can do this since we know how many copies we're going to need): a "copyable" commitment to a, assuming we're going to need k copies, consists of $v = C(C(a, r_1), s_1), \ldots, C(C(a, r_k), s_k)$. The i^{th} copy of the commitment is $C(a, r_i)$. The hiding property of C ensures that there is no way to connect $C(a, r_i)$ to v. On the other hand, the binding property of C ensures that we cannot find any s' such that $C(C(a, r_i), s')$ is in v unless this was the original commitment.

Unlike the case with Pedersen commitments, when using the double commitment trick an adversary *can* "copy" a commitment to a different value (the adversary can always use different values in the inner commitments). The insight here is that the adversary still has to commit in advance to the locations of the errors. After the DRE commits, we randomly permute the indices (so $v = C(C(a, r_{\sigma(1)}), s_{\sigma(1)}), \ldots, C(C(a, r_{\sigma(k)}), s_{\sigma(k)})$, for some random permutation σ). If the DRE did commit to the same value at every index, this permutation will not matter. If the DRE committed to different values, we show that it will be caught with high probability. Intuitively, we can consider each commitment in the tally phase as a row of a matrix whose columns correspond to the voters. By permuting the indices of the copies, we are effectively permuting the columns of the matrix (since in the i^{th} tally step we force the DRE to use the i^{th} copy. The DRE can pass the test only if all the rows in this matrix have the same tally. But when the columns are not constant, this occurs with small probability[1]. The formal protocol description and proof of security, as well as a proof that our protocol is receipt-free, will appear in the full version.

4 Incoercibility and Receipt-Freeness

Standard "secure computation" models usually deal with two types of parties: honest parties with a secret input that follow the protocol, and corrupt parties that are completely controlled by the adversary. In voting protocols, we often need to consider a third type of player: a party that has a secret input, but is threatened (or bribed) by the adversary to behave in a different manner. Such a "coerced" player differs from a corrupt party in that she doesn't do what the adversary wishes if she can help it; if she can convince the adversary that she's following its instructions while actually following the protocol using her secret input, she will.

Benaloh and Tuinstra [2] were the first to introduce this concept. Most papers concerning receipt-free voting (including Benaloh and Tuinstra), do not give a rigorous definition of what it means for a protocol to be receipt-free, only the intuitive one: "the voter should not be able to convince anyone else of her vote".

Canetti and Gennaro considered this problem for general multiparty computation (of which voting is a special case), and gave a formal definition of *incoercibility* [4]. Their definition is weaker than receipt-freeness, however: the adversary is only allowed to coerce a player after the protocol is complete (i.e.,

[1] for technical reasons, the technique we use in practice is slightly different, however the idea is the same

it cannot require a coerced player to follow an arbitrary strategy, or even specify what randomness to use).

Juels, Catalano and Jakobsson also give a formal definition of *coercion-resistance* [13]. Their definition has a similar flavor, but is specifically tailored to voting in a public-key setting. It is stronger than receipt-freeness in that a coercion-resistant protocol must also prevent an *abstention-attack* (preventing a coerced voter from voting). However, this strong definition requires anonymous channels from voters to tallying authorities (otherwise an abstention-attack is always possible).

Our 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 our definition is incoercible as well). Loosely, both definitions require 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). The difference is the adversarial model we consider in the real world. Canetti and Gennaro allow the adversary to give coerced players an alternative input. In our definition, the adversary can give coerced players an arbitrary strategy (i.e. commands they should follow instead of the real protocol interactions).

4.1 Formal Definition

Formally, a receipt-free protocol must specify, in addition to the strategy for honest parties, a *coercion strategy*. When a party is coerced by the adversary, it begins following the coercion strategy instead of the honest strategy. The coercion strategy tells the party how to fake its responses to the adversary.

Ideal Setting. Consider the "ideal setting" for computing a function f. In this setting we have n players, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$ 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. The only thing an adversary can do with a coerced party is to force it to use the input $\perp$ (signifying a forced abstention).

$\mathcal{I}$ 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 and the output of the ideal functionality $f(x_1, \ldots, x_n)$ (where for corrupted parties x_i is the input chosen by the adversary).

Real-World Setting. In the "real-world setting", we also have the n parties, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$. Each party has a protocol specification, consisting of a pair of interactive turing machines (H_i, C_i). There is also a "real" adversary $\mathcal{A}$.

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

1. to corrupt a subset of the parties. In this case the adversary learns the entire past view of the party and completely controls its actions from that point.
2. to coerce a subset of the parties. The adversary supplies each coerced party with a "fake" input x'_i. From this point on it can interactively query and send commands to the coerced party. The coerced party switches to the coercion strategy C_i, which specifies how to respond to $\mathcal{A}$'s queries and commands.

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

Definition 1. *A protocol is receipt-free if there exists an ideal adversary $\mathcal{I}$, such that for any real adversary $\mathcal{A}$ (with which $\mathcal{I}$ can communicate as a black-box) and any input vector $x_1, \ldots, x_n$, $\mathcal{I}$'s output in the ideal world is indistinguishable from $\mathcal{A}$'s view of the protocol in the real world with the same input vector (where the distributions are over the random coins of $\mathcal{I}$, $\mathcal{A}$ and the parties).*

It is important to note that even though a protocol is receipt-free by our 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).

5 Discussion

Splitting the Vote. Our scheme suffers a large drawback compared to many of the published universally-verifiable schemes: we do not know how to distribute the vote between multiple authorities. In our protocol, the DRE acts as the only tallying authority. Thus, a corrupt DRE can reveal the voters' ballots. This is also the case with Chaum and Neff's schemes, however (as well as traditional DRE schemes in use today). It seems a fairly difficult problem to solve while taking into account the limited abilities of voters (e.g., most secret sharing schemes cannot be performed by unaided humans).

Robustness of the Voting Scheme. The *robustness* of a voting scheme is its ability to recover from attacks without requiring the election to be canceled. Because the DRE is the sole tallying authority, a corrupt DRE can clearly disrupt the elections (e.g., by failing to output the tally). The UC proof shows that voters cannot disrupt the elections just by interacting with the DRE (the only thing a voter can do in the ideal model is either vote or abstain). However, our model does not prevent *false* accusations against the DRE. For example, a corrupt voter that is able to fake a receipt can argue that the DRE failed to publish it on the bulletin board. Using special paper for the receipts may help, but preventing this and similar attacks remains an open problem.

Traditional Paper Trail as Backup. Many critics of non-cryptographic DRE systems are pushing for a "voter verified paper-trail": requiring the DRE to print a plaintext ballot that the voter can inspect, which is then placed in a real ballot box. In a non-cryptographic system, the paper trail can help verify that a DRE is behaving honestly, and act as a recovery mechanism when it is not. In our system, a paper trail can be used for the recovery property alone: if the DRE is misbehaving, our protocol ensures it will be detected with high probability (without requiring random audits of the paper trail). In that case, we can perform a recount using the paper ballots.

Randomness and Covert channels. One problem that also plagues Neff's scheme, and possibly Chaum's [14], is that a corrupt DRE can convey information to an adversary using subliminal channels. In this case, an adversary only needs access to the bulletin board in order to learn all the voters choices. The source of the problem is that the DRE uses a lot of randomness (e.g., the masking factors for the commitments and the random permutations in the final tally phase).

In our scheme based on Pedersen commitments, we have a partial solution to this problem. It requires the output of the DRE to be sent through a series of "filters" before reaching the printer or bulletin board. The idea is that for each Pedersen commitment of the form $x = h^a g^r$ received by a filter, it will choose a random masking factor s, and output xg^s. If the DRE opens x by sending (a, r), the filter will send instead $(a, r + s)$. In a similar way the filter can mask the permutations used in the final-tally phase by choosing its own random permutations and composing them. Note that the filter does not need to know the value of the commitments or the original permutations in order to perform its operation. If the DRE is honest, the filter receives no information and so cannot covertly send any. If the filter is honest, any randomness sent by the DRE is masked, so no information embedded in that randomness can leak. By using a series of filters, each from a different trustee, we can ensure that the DRE does not utilize covert channels (as long as at least one of the filters is honest). A general solution to this problem is still an interesting open problem.

An even stronger adversary may be able to both coerce voters and maliciously program the DRE. Our protocol is vulnerable in this case. For example, a coercer can require a voter to use a special challenge, which is recognized by the DRE (a coercer can verify that the voter used the required challenge, since it appears on the public bulletin board). Once it knows a voter is coerced, the DRE can change the vote as it wishes (since the coerced voter will not be able to complain). Possibly mitigating the severity of this attack is the fact that, in order to significantly influence the outcome of an election, the adversary must coerce many voters. This makes it much harder to keep the corruption secret.

Separate Tallying. Our scheme requires the tally to be performed separately for each DRE. This reveals additional information about voter's choices (in many real elections this information is also available, however). An open problem is to allow a complete tally without sacrificing any of the other properties of our scheme (such as receipt-freeness and everlasting privacy).

References

1. Y. Aumann, Y. Z. Ding, and M. O. Rabin. Everlasting security in the bounded storage model. *IEEE Transactions on Information Theory*, 48(6):1668–1680, 2002.
2. J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *STOC '94*, pages 544–553, 1994.
3. 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.
4. R. Canetti and R. Gennaro. Incoercible multiparty computation. In *FOCS '96*, pages 504–513, 1996.
5. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.
6. D. Chaum. Blind signature systems. In *CRYPTO '83*, page 153, 1983.
7. D. Chaum. E-voting: Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, Jan./Feb. 2004.
8. J. D. Cohen(Benaloh) and M. J. Fischer. A robust and verifiable cryptographically secure election scheme. In *FOCS '85*, pages 372–382, 1985.
9. R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *EUROCRYPT '96*, pages 72–83, 1996.
10. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Eurocrypt '97*, pages 103–118, 1997.
11. A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT '92*, volume 718 of *LNCS*, pages 244–251, 1993.
12. M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Eurocrypt 2000*, volume 1807 of *LNCS*, pages 539+, 2000.
13. A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *WPES '05*, pages 61–70, 2005.
14. C. Karlof, N. Sastry, and D. Wagner. Cryptographic voting protocols: A systems perspective. In *USENIX Security '05*, pages 33–50, 2005.
15. M. Naor and A. Shamir. Visual cryptography. In *Eurocrypt '94*, volume 950 of *LNCS*, pages 1–12, 1995.
16. C. A. Neff. Practical high certainty intent verification for encrypted votes, October 2004. http://www.votehere.net/vhti/documentation/vsv-2.0.3638.pdf.
17. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*, volume 576 of *LNCS*, pages 129–140, 1991.
18. D. J. Reynolds. A method for electronic voting with coercion-free receipt. FEE '05. Presentation: http://www.win.tue.nl/~berry/fee2005/presentations/reynolds.ppt.
19. P. Y. A. Ryan. A variant of the Chaum voter-verifiable scheme. In *WITS '05*, pages 81–88, 2005.
20. K. Sako and J. Kilian. Receipt-free mix-type voting schemes. In *EUROCRYPT '95*, volume 921 of *LNCS*, pages 393–403, 1995.
21. A. Shamir. Cryptographers panel, RSA conference, 2006. Webcast: http://media.omediaweb.com/rsa2006/1_5/1_5_High.asx.