New Techniques for Electronic Voting

Alan Szepieniec and Bart Preneel
ESAT/COSIC, KU Leuven and iMinds, Belgium
first.lastname@esat.kuleuven.be

2015-08-13

Abstract. This paper presents a novel unifying framework for electronic voting in the universal composability model that includes a property which is new to universal composability but well-known to voting systems: universal verifiability. Additionally, we propose three new techniques for secure electronic voting and prove their security and universal verifiability in the universal composability framework.

1. A tally-hiding voting system, in which the tally that is released consists of only the winner without the vote count. Our proposal builds on a novel solution to the millionaire problem which is of independent interest.
2. A self-tallying vote, in which the tally can be calculated by any observer as soon as the last vote has been cast — but before this happens, no information about the tally is leaked.
3. Authentication of voting credentials, which is a new approach for electronic voting systems based on anonymous credentials. In this approach, the vote authenticates the credential so that it cannot afterwards be used for any other purpose but to cast that vote. We propose a practical voting system that instantiates this high-level concept.

Note. This is the extended version of the paper with the same title [33], which appeared in the USENIX Journal of Election Technology and Systems (JETS) [35]. Please cite that version instead.

1 Introduction

For over two thousand years, voting systems have been in use. Recent developments in cryptography have enabled the transition from roll call votes or paper ballots to electronic voting systems. The application of cryptography to voting systems does not merely ensure correctness and security; on the contrary, it improves on these and offers new desirable properties when compared to its low-tech predecessors. However, the use of cryptography in this context leaves society no option but to trust a small number of cryptographers, and hardware and software vendors who develop and distribute these cryptographic voting systems.

Even more recently, the internet has revolutionized communication and the decision making process. What used to be a theoretical exercise is now becoming a practical reality: the internet allows us to hold votes and even state elections from our own personal computers.

However, the internet has had little effect on the theoretical foundation of electronic voting systems. The formalisms and techniques proposed in other research as well as in this paper make abstraction of the communication channel and apply equally to any voting system *not* conducted over the internet. On the other hand, there is one very important implication of internet voting that bears endless repeating: no mechanism can prevent the coercer from being physically

present when the voter casts his vote, and hence *internet voting is inherently coercible*. In order to guard against coercion, a fundamentally different adversarial model must be considered. While this is addressed in the literature [4, 7, 26, 31], coercive adversaries are beyond the scope of this paper.

The standard framework for assessing the security of cryptographic protocols is the universal composability (UC) framework proposed by Canetti [6]. While the UC framework imposes stringent constraints on the protocols and their design, it is beneficial for at least two reasons. First, the UC framework applies to *any* protocol and not just a predefined subset of them. Second, universal composability guarantees that the composition of UC-secure protocols is itself UC-secure, whether the components are invoked once or multiple times, recursively or not, serially or in parallel. This principle enables protocol designers to adopt a modular approach whereby the global protocol is composed of smaller subprotocols, each of which is proved UC-secure in its own right.

A *voting system* is a protocol like any other and hence the UC framework is applicable not just to the whole but also to the various subprotocols. Groth [22], for example, uses the UC framework to assess the security of some cryptographic voting systems. However, this assessment applies only to homomorphic aggregation voting systems and does not extend to mixnet-based or anonymous credential-based voting systems. Moreover, it does not take into account universal verifiability, *i.e.* the ability of any observer to verify the tally. Other formalisms, not explicitly taking UC-security into account, include the ones by Benaloh [3], Cramer *et al.* [14,15], Adida [1] and Chevallier-Mames [11].

Our contributions. We present a new formalism of voting systems which is more general than that of Groth as it contains homomorphic aggregation schemes as a special case. We also take the properties introduced in the other formalisms into account. Additionally, we formalize and prove a composability theorem for universal verifiability. We note that, at first sight, this notion of universal verifiability seems similar to the notion of public auditability for multiparty computation protocols recently and independently introduced by Baum, Damgård and Orlandi [2]. However, there are several important differences, which will be explained in due course.

Moreover, we introduce three new techniques for electronic voting protocols and subsequently prove their UC-security and, hence, their compatibility with our formalism. First, we propose a "Tally-Hiding Vote": a voting system where the result of the tallying process identifies the winner of the vote but leaks no information on the vote count. Our proposal uses a novel solution to Yao's millionaire problem [36] which depends on a well-known property of Damgård and Jurik's threshold cryptosystem [16], namely the ability to lift a ciphertext from one ciphertext space into another, larger space without modifying the plaintext. Our solution to the millionaire problem invokes this lifting procedure as an abstract black box. While Damgård and Jurik have proposed one instantiation of this protocol [17], we propose another one which is tailored to our particular use case.

Second, we propose a new "Self-Tallying" voting system, along the lines of Kiayias–Yung [27] and Groth [21]. In this type of voting system, the computationally expensive part of the tallying process is completed beforehand. The tally is known — or is easily computed by anyone — as soon as the last voter has cast his or her vote but not before. The obvious drawback is that one voter can boycott the entire procedure by refusing to cast his or her vote. The essence of our approach, like that of Kiayias and Yung, is that the randomizers used by the voters to encrypt their votes are not entirely random but in fact cancel out when aggregated. As long as

at least one voting authority is not corrupt, the authorities are unable to determine any one voter's particular random value or vote. Our system allows for a practically unlimited number of secure votes after just one initialization procedure of constant size, in contrast to the scheme of Kiayias and Yung where this initialization procedure is linear in the number of intended elections.

Third, we present a new paradigm for credential-based voting, namely "Authenticated Voting Credentials", along with a scheme that implements this concept. In contrast to previous schemes for electronic voting based on anonymous credentials, such as those of Chaum [10], Fujioka *et al.* [20], and Ohkubo *et al.* [29], our credentials are authenticated by the vote that is cast. In this setting, a man-in-the-middle cannot intercept a credential and append it to his own vote or just modify the vote; at most, he can prevent the voter's vote from being cast by blocking it entirely. While the concept applies to *any* credential system, we demonstrate it by applying it to the credential system by Ferguson [18], which draws from Chaumian blind signatures [9], and which we use in combination with Guillou and Quisquater's zero-knowledge proof of an RSA signature [23].

Outline. We start by summarizing universal composability in Section 2 after which we present our formalism of electronic voting and universal verifiability. In the subsequent sections, we cover our three new techniques: Tally-Hiding Vote in Section 3, Self-Tallying Vote in Section 4, and Authenticated Voting Credentials in Section 5. Finally, Section 6 concludes the text.

Notation and Preliminaries. Let $\kappa \in \mathbb{N}$ be a security parameter. We say a function $\epsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ of the security parameter is *negligible* if $\forall c \in \mathbb{R}_{>0} \,.\, \exists K \in \mathbb{N} \,.\, \forall \kappa \geq K \,.\, \epsilon(\kappa) \leq \kappa^{-c}$. A *public-key encryption scheme* consists of a tuple of possibly probabilistic algorithms $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ where G generates a matching public and private key; E is an algorithm that encrypts a given message using the public key and D decrypts a ciphertext provided that the matching private key is used. A public key cryptosystem is *semantically secure* if no non-trivial function of the plaintext can be computed given only the ciphertext and the public key. By $x \leftarrow y$ we denote the assignment of the value y to the variable x; by $x \xleftarrow{\$} S$ we denote a uniformly random selection from a finite set S and assignment of this value to the variable x. Long arrows with values on top denote messages sent between parties. For any positive integer $n \in \mathbb{N}$, we use $\mathbb{Z}_n$ to denote, depending on the context, either the ring of integers modulo n, or the set of integers $\{0, 1, \ldots, n-1\}$. Likewise, we use $\mathbb{Z}_n^*$ to denote either the group of integers under multiplication modulo n or else the subset of $\mathbb{Z}_n$ containing integers that are invertible for multiplication modulo n.

The Paillier cryptosystem is a semantically secure public-key scheme whereby the public key is given by n, a product of two large private primes, and the secret key is given by d, a number which satisfies $d = 0 \mod \varphi(n)$ and $d = 1 \mod n$. Encryption of a message m using randomness r is defined as $\mathsf{E}(m) = (1+n)^m r^n \mod n^2$. Decryption consists of first raising the ciphertext to the exponent d in order to cancel the randomizer, and secondly computing the easy discrete logarithm of the resulting value.

2 Secure Electronic Voting

2.1 Universal Composibility

The key idea of universal composability [6] is the comparison between two worlds in both of which a set of participants compute a function of their inputs. In one world, they hand their

inputs to a trusted third party called the *ideal functionality* which calculates the result and hands the correct outputs to the respective parties. In the other world, the computation is performed by a protocol executed by the parties. In both worlds, there is an external environment $\mathcal{E}$, modeled as a polynomial-time program or Turing machine, which chooses the inputs of the participants and reads their outputs. The environment interacts with the adversary throughout the process.

A protocol is a *UC-secure realization* of the ideal functionality if for every adversary $\mathcal{A}$ interacting with the participants while they execute the protocol, there exists a simulator $\mathcal{S}$ which interacts with the ideal functionality and with the parties involved such that no external environment $\mathcal{E}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ or with $\mathcal{S}$. This implies that any attack that can be performed by $\mathcal{A}$ against the protocol can also be performed by $\mathcal{S}$ against the ideal functionality.

The power of the UC framework is demonstrated by the universal composability theorem: if $\mathcal{P}_1$ securely realizes an ideal functionality $\mathcal{F}$, and if a protocol $\mathcal{P}_2$ in which $\mathcal{F}$ is used as subprotocol is secure, then replacing $\mathcal{F}$ by $\mathcal{P}_1$ will not change the security of $\mathcal{P}_2$. This theorem allows us to design secure protocols by designing secure modules which realize smaller functionalities.

2.2 Voting

An *option* is any string of characters that is known by the voters to refer to a hypothetical situation. Let $\mathbf{O}$ denote the set of options for a given vote; and let $\mathsf{P}\mathbf{O}$ denote the set of permutations of $\mathbf{O}$, which corresponds to the set of expressible preferences with respect to $\mathbf{O}$.[1] A *vote* (in the first sense of the word) is an expressed preference with respect to the options and thus an element of $\mathsf{P}\mathbf{O}$.

A *tally* is any string of characters which directly implies a partial ordering on the set of options — an ordering where some subsets are allowed to have the same rank. A tally may be the encoding of a mapping of the options to their vote counts, or of a string which identifies only the winning option or of something else. Let $\mathbf{T}$ denote the set of tallies.

A *tallying function* f is any function that maps n votes to a tally:

$$f : (\mathsf{P}\mathbf{O})^n \to \mathbf{T}.$$

A *voting system* is an interactive protocol which evaluates a tallying function in the set of votes cast by the voters. We distinguish two classes of participants: voters $V_1, V_2, \ldots, V_n \in \mathbf{V}$, whose votes should be counted by the tallying function; and authorities $A_1, A_2, \ldots, A_k \in \mathbf{A}$, who bear the responsibility for the correct execution of the protocol and who may be expected to perform more work. The protocols are designed to work as long as a subset of size t of the authorities are not corrupt.

The second meaning of the word "vote" refers to one instantiation or execution of a voting system. In an electronic voting setting, all participants are assumed to have access to a public append-only bulletin board. In the Section 2.3, we model this bulletin board by an ideal functionality.

We aim to design voting systems which are *universally verifiable*, meaning that *anyone* can verify the correctness of the protocol execution after its execution by requesting a transcript which certifies this. Provided that verification succeeds, universal verifiability guarantees *correctness*, which means that the tally that results from the protocol conforms to the tallying

[1] Preferences are always necessarily ordinal. This implicit ordering is manifested when the ballot does not allow "no preference". This implicit ordering may be not encoded if the ballot does allow it.

function and the cast votes. Correctness, in turn, implies *completeness* — all valid votes are counted correctly; *soundness* — invalid votes are not counted; *eligibility* — only votes from eligible voters $V_i \in \mathbf{V}$ are counted; *unreusability* — no voter can vote in excess of his or her quota (possibly but not necessarily one vote per person); and *finality* — no voter can change his or her vote after the deadline has expired.

In addition to correctness, we want the voting system to ensure privacy. This feature is formalized by *perfect ballot secrecy*, which might also be called *full vote confidentiality*. This property guarantees that the adversary cannot compute any information on any one individual's vote beyond what follows from the tally (and from the votes of the corrupted parties). Moreover, it implies *fairness*: no partial tallies can be known before the vote casting deadline — indeed, vote confidentiality requires that no partial tallies can by known at *any* point in time.

Other desirable features are *receipt-freeness*, where a voter cannot prove that he voted one way or another. Related to this feature is *uncoercibility*, where a voter cannot be coerced into voting a specific way even by a coercer who interacts with them *while casting the vote*, for example by allowing the voter to re-cast at another time. These properties fall outside of the scope of this paper.

2.3 UC-Secure Voting Systems

A voting system can only be UC-secure if it is a secure realization of an *ideal functionality*. Hence, we define the ideal functionality to be realized as well as the ideal functionalities that are allowed to be invoked. We start with the second, from the bottom upwards.

Groth [22] correctly notes that the bulletin board as well as a public key generation mechanism are in fact ideal functionalities. While Groth presents the two functionalities as a single ideal functionality, we opt to separate the two. The reason is that one might want to use the one functionality without using the other. The bulletin board is formalized by the ideal functionality $\mathcal{F}_{\mathsf{BB}}$ and the system public key generation mechanism is formalized by the ideal functionality $\mathcal{F}_{\mathsf{SKG}}$. The functionalities run with voters $V_1, \ldots, V_n$ and authorities $A_1, \ldots, A_k$, and with either an adversary $\mathcal{A}$ or adversary-simulator $\mathcal{S}$ (but for the purpose of describing the ideal functionalities, these two are interchangeable).

The bulletin board is a public append-only database of messages (any string of characters) from any protocol participant. We extend the power of the adversary compared to Groth. In addition to blocking honest voters' messages, the adversary in our bulletin board model can even falsify their messages — and the same holds for the messages of authorities. This is accomplished as follows: the functionality forwards all messages not originating from the adversary, to the adversary; the adversary can choose to allow them to the bulletin board by instructing the ideal functionality to do so. In contrast, the adversary cannot prevent voters nor authorities from reading bulletin board messages. Access to the bulletin board is not anonymous as all messages pass through the adversary. However, in Section 5 we deviate from this model as we are only able to prove UC-security of a credential-based voting system on the premise that access to the bulletin board *is* anonymous, *i.e.*, the adversary cannot determine who is posting messages to the bulletin board or even whether this communication is taking place. The bulletin board functionality $\mathcal{F}_{\mathsf{BB}}$ is presented in Functionality 1.

The system key generation mechanism is straightforward. The functionality starts by generating a public key, matching private key shares and verification keys for those shares. The private key shares are distributed to the proper authorities. The public key as well as the verification keys are distributed to everyone such that anyone who assumes the role of the verifier

Functionality $\mathcal{F}_{\mathsf{BB}}$

- Upon receiving $(\textbf{message}, sid, m)$ from any participant $P_i \in \mathbf{V} \cup \mathbf{A}$, send $(\textbf{message}, sid, P_i, m)$ to $\mathcal{S}$.
- Upon receiving $(\textbf{post}, sid, m)$ from $\mathcal{S}$, store it.
- Upon receiving $(\textbf{read}, sid)$ from any participant $P_i \in \mathbf{V} \cup \mathbf{A} \cup \{\mathcal{S}\}$, send to P_i the list of all stored tuples $(\textbf{post}, sid, *)$ in chronological order.

Functionality 1: bulletin board.

can verify the correct execution of the authorities' distributed decryption protocol. The functionality is parameterized by the public key cryptosystem to instantiate. For example, in a homomorphic aggregation type voting system, an additively homomorphic public key encryption scheme is required while for a credential-based voting system, a blind signature scheme is necessary. The system key generation functionality is presented in Functionality 2.

Functionality $\mathcal{F}_{\mathsf{SKG}}$

- Upon invocation, generate keys for a public key threshold cryptosystem. Send $(\textbf{public key}, sid, pk)$, where pk is the public key, to all participants and to $\mathcal{S}$. Send to all participants and to $\mathcal{S}$ the tuple $(\textbf{verification keys}, sid, vk_1, \ldots, vk_k)$ where vk_j is the verification key for A_j's key share. Send to all authorities $A_j \in \mathbf{A}$ the message $(\textbf{secret share}, sid, sk_j)$ where sk_j is the secret key share for A_j.

Functionality 2: system key generation mechanism.

While more realistic than the alternative, allowing the adversary to also falsify voters' messages does come with a price. The authorities need a mechanism by which to separate eligible voters from non-eligible voters (or more precisely: separate messages from eligible voters from messages from an adversary masquerading as eligible voter). We achieve this property by introducing an ideal participant key generation functionality, $\mathcal{F}_{\mathsf{PKG}}$. The ideal participant key generation functionality generates two public and private keys for each participant — one pair for encryption and one pair for signatures. The private keys are communicated to the respective participants whereas the public keys are sent to all participants including the adversary. After invoking the participant key generation functionality, participants can authenticate all their outward communication if they should choose to do so by signing it with their private signature key. After this functionality invocation, anyone can distinguish between eligible and non-eligible voters based on the list of eligible voter public keys.

The ideal participant key generation functionality also turns out to be of crucial importance for universal verifiability. We will formally define the concept later but at this point we introduce a verifier $\mathcal{V}$, which is another party in the protocol. The public keys produced by $\mathcal{F}_{\mathsf{PKG}}$ are also sent to the verifier, who uses them to verify that genuine interaction between real parties has taken place. Since the adversary does not possess the private keys from honest parties, he cannot fake their communications. The verifier, too, can distinguish between votes from eligible

voters and votes from non-eligible voters based on the list of eligible voter public keys. $\mathcal{F}_{\mathsf{PKG}}$ is formally presented in Functionality 3.

Functionality $\mathcal{F}_{\mathsf{PKG}}$

- Upon invocation, for each voter $V_i \in \mathbf{V}$, generate a public/private key pair for encryption and a public/private key pair for signatures. Send the private keys to the V_i and the public keys to all participants including $\mathcal{S}$ and the verifier $\mathcal{V}$.
- Upon invocation, for each authority $A_j \in \mathbf{A}$, generate a public/private key pair for encryption and a public/private key pair for signatures. Send the private keys to the A_j and the public keys to all participants including $\mathcal{S}$ and the verifier $\mathcal{V}$.

Functionality 3: eligible participant key generation mechanism.

Having properly formalized the ideal functionalities which a voting system might invoke, we can turn to the voting system itself. The ideal voting system, $\mathcal{F}_{\mathsf{VS}}$, is presented in Functionality 4. Here, in contrast to the bulletin board functionality, the adversary is not allowed to read the cast votes or to falsify them. However, he *is* allowed to block them. In fact, the ideal functionality blocks the votes by default and requires the intervention of the adversary, in the form of a **no-block** message, to unblock the votes.

Functionality $\mathcal{F}_{\mathsf{VS}}$

- Upon receiving $(\textbf{vote}, sid, v_i)$ from voter $V_i \in \mathbf{V}$ and if v_i is a valid vote, store $(\textbf{vote}, sid, v_i, V_i)$ and send $(\textbf{vote}, sid, V_i)$ to $\mathcal{S}$. (Note that the vote v_i itself is *not* sent to the adversary.)
- Upon receiving $(\textbf{no-block}, sid, V_i)$ from $\mathcal{S}$, store it for future reference.
- Upon receiving $(\textbf{tally}, sid)$ from all the authorities, compute the tallying function on all the stored votes v_i from unique voters and for a matching **no-block** message was stored. If a voter V_i has multiple stored votes in excess of his quota, choose the most recent quota-sized subset of votes for which there is a matching **no-block** message. Calculate $t = f(v_1, \ldots, v_n)$ and send $(\textbf{result}, sid, t)$ to all parties including $\mathcal{S}$ and halt.

Functionality 4: voting system.

A UC-secure realization of $\mathcal{F}_{\mathsf{VS}}$ must necessarily satisfy the correctness and perfect ballot secrecy properties. Thus, if we prove that a particular voting system is a UC-secure realization of $\mathcal{F}_{\mathsf{VS}}$, then we prove that all the implied properties enumerated at the end of Section 2.2 are satisfied (i.e.: correctness — completeness, soundness, eligibility, unreusability, and finality — as well as perfect ballot secrecy). However, universal verifiability is less straightforward.

A naïve approach to universal verifiability might grant the verifier access to the bulletin board and demand that the transcript be of such a form that the verifier can easily catch any cheater. This approach is insufficient because it does not adequately capture the power of the

adversary who attacks the protocol and who might be spoofing the verifier's read access to the bulletin board or manipulate its contents after the protocol has been executed.

However, we can demand that there be a transcript — either because a correct execution of the protocol produced it, or else because the adversary is trying to fool someone with it. We can then formalize universal verifiability as follows.

Definition. (Universal verifiability)
Let $\mathcal{V}$ (verifier) be a probabilistic polynomial-time algorithm which takes as input the transcript T as produced by an adversary $\mathcal{A}$ attacking a protocol $\mathcal{P}$. $\mathcal{V}$ eventually outputs a bit $\tilde{b}$. Let b be a variable indicating whether the protocol was executed correctly by all parties — *i.e.*, the parties behaved honestly and were not corrupted by the adversary — and if the transcript is authentic, by assuming the value 1 if this is the case and 0 otherwise. Then the protocol $\mathcal{P}$ is *universally verifiable* if there exists a verifier $\mathcal{V}$ such that, for all adversaries $\mathcal{A}$ attacking the protocol, $\mathcal{V}$ has significant distinguishing power:

$$\left|\Pr\left[b = \tilde{b}\right] - \Pr\left[b \neq \tilde{b}\right]\right| \geq \frac{1}{2} \ ,$$

where the probabilities are taken over all random coins used by $\mathcal{V}$, $\mathcal{A}$ and $\mathcal{P}$.

In other words, a protocol is universally verifiable if no adversary attacking it can convince the verifier that it was executed correctly by all parties when it was not. Without loss of generality, we assume that the verifier $\mathcal{V}$ indicates the transcript is authentic and the protocol execution was correct by outputting $\tilde{b} = 1$ and indicates the opposite with $\tilde{b} = 0$. The constant $\frac{1}{2}$ captures the notion of "significant" distinguishing power but in effect, any constant not negligibly close to 0 or 1 will do.

The verifier $\mathcal{V}$ trusts ideal functionalities and hence does not need to verify their correct execution. This makes the composability theorem for universal verifiability rather trivial.

Theorem 1. ***If*** *(1) $\mathcal{P}_1$ is a hybrid protocol invoking the ideal functionality $\mathcal{F}_2$, and (2) $\mathcal{P}_1$ is universally verifiable for verifiers that trust $\mathcal{F}_2$, and (3) $\mathcal{P}_2$ is a universally verifiable protocol that realizes $\mathcal{F}_2$;* ***then*** *$\mathcal{P}_1$ is still universally verifiable if it invokes $\mathcal{P}_2$ rather than $\mathcal{F}_2$.*

Proof of Theorem 1.
If there is a protocol $\mathcal{P}_1$ which invokes a universally verifiable ideal functionality $\mathcal{F}_2$, then the verifier $\mathcal{V}_1$ for $\mathcal{P}_1$ will not need to verify $\mathcal{F}_2$ as this functionality is trusted anyway. If $\mathcal{F}_2$ is realized by a universally verifiable protocol $\mathcal{P}_2$, then there must exist a verifier $\mathcal{V}_2$ that verifies its correct execution given a transcript T_2. Rather than trusting $\mathcal{F}_2$, $\mathcal{V}_1$ now requests a transcript T_2 for $\mathcal{P}_2$ from the adversary and simulates $\mathcal{V}_2$. $\mathcal{V}_1$ returns 0 if $\mathcal{V}_2$ returns 0. This is repeated for every invocation of $\mathcal{P}_2$. □

We stress that the verifier $\mathcal{V}$ must be able to differentiate between simulated parties created by the adversary $\mathcal{A}$ and genuine protocol participants. As such, the verifier must be a recipient of the public keys from $\mathcal{F}_{\mathsf{PKG}}$.

This notion of universal verifiability is in essence very similar to that of Baum, Damgård and Orlandi [2], which they name *public auditability*. Rather than in terms of a game between the verifier and the adversary, Baum *et al.* define public auditability in terms of the root ideal functionality which is extended so that it responds honestly to any participant who requests an

audit. This response indicates whether or not parties in the protocol have behaved corruptly. This extension to the ideal functionality is realized by allowing anyone to have access to the bulletin board and view the transcript of the protocol. Anyone who wishes to audit, can retrieve a transcript of the protocol and decide themselves whether or not all participants behaved correctly. Only if the protocol is well-formed (for example, because the correctness of every step is proven by non-interactive zero-knowledge proofs), does the auditor stand a chance at deciding correctly, and only then can they be said to realize this extension to the ideal functionality.

Public auditability is dependent on the existence of the bulletin board, which must contain the transcript of the protocol. On the other hand, universal verifiability takes into account that the verifier's access to the bulletin board might be spoofed by the adversary. Instead, rather than relying on the existence of an append-only (even for the adversary) bulletin board, the verifier in the universal verifiability game requests the protocol transcript from the adversary. As such, universal verifiability is a stronger concept as it provides a similar notion of verifiable correctness in the face of a more powerful adversary.

Moreover, the verifier in the universal verifiability framework is able to verify the authenticity of participant's messages if they are accompanied by signatures, as the verifier has received the public keys from $\mathcal{F}_{\mathsf{PKG}}$. On the other hand, the auditor in the public auditability framework trusts the bulletin board to record the identities of whoever posts a message — but if the bulletin board guarantees this, then it precludes anonymous access. Universal verifiability does not depend on the existence of a bulletin board and does not impose such stringent constraints. Hence, a protocol based on anonymous communication cannot be publicly auditable, while it can be universally verifiable — possibly by employing anonymous credentials.

Lastly, universally verifiable protocols are composable, and hence integrate nicely with universally composable protocols. On the other hand, public auditability was designed specifically for multiparty computation and does not explicitly take into account modular design of just any protocol in the same way the UC framework does.

3 Tally-Hiding Vote

If the goal is to preserve voter privacy, the logical extreme is to release only the winning option of the vote and not the number of votes each option received. This idea to "show the winner without the tally" was first introduced for electronic voting by Benaloh [12]. We present a relatively efficient instantiation of this idea.

The key primitive used is a novel solution to the millionaire problem. Given two Paillier ciphertexts where the secret key is distributed among the voting authorities, there exists an efficient protocol to determine which of the two plaintexts is larger. For now, we consider only the case where there are two options. This strategy is sufficient for computing the winner in a preferential election while keeping the particular vote counts secret, but this does leak the entire ranking of all the options. Later on, we show a solution to calculate the winner among r options all the while leaking no information either on the ranking of the other options or on the particular vote counts.

Our scheme builds on a well-known property of the Damgård-Jurik cryptosystem [16] (a generalization of Paillier's cryptosystem [30]) which allows the holders of the secret key to turn a ciphertext from one space, $\mathbb{Z}_{n^2}^*$ into a ciphertext from another space, $\mathbb{Z}_{n^3}^*$ such that it encrypts the same plaintext. Under certain conditions on the plaintext no information is leaked by this procedure (except that the plaintext satisfies said conditions). We invoke this *ciphertext lifting*

procedure as a black box. While Damgård and Jurik themselves were the first to propose an instantiation of this protocol, we propose an alternate one which is tailored to our needs.

First, we prove that, given access to an ideal functionality $\mathcal{F}_{\mathsf{MP}}$ which solves the millionaire problem for two ciphertexts, we can construct a UC-secure tally-hiding voting system. Then, we show how we can realize $\mathcal{F}_{\mathsf{MP}}$ using ciphertext lifting as a black box ideal functionality. In Appendix A we present our alternative to Damgård and Jurik's lifting procedure.

3.1 UC-Secure Tally-Hiding Vote

In the original millionaire problem [36], two millionaires want to decide who of them is richer without disclosing any other information about their wealth. In our case, there are k participants, each holding a share in the private key, who want to decide which one of two ciphertexts encrypts the larger plaintext without revealing any other information. For now, we assume we have a solution to this problem in the form of an ideal functionality and base a voting system on it. Later on, we present such a solution.

We define the ideal functionality $\mathcal{F}_{\mathsf{MP}}$ for the millionaire problem in Functionality 5. The functionality takes two ciphertexts c_1, c_2 as input and compares the corresponding plaintexts, i.e.: it outputs 1 if $\mathsf{D}(c_1) < \mathsf{D}(c_2)$ and 0 otherwise. The functionality runs with all authorities. This functionality may be considered part of $\mathcal{F}_{\mathsf{SKG}}$, the system key generation functionality.

Functionality $\mathcal{F}_{\mathsf{MP}}$

– Upon receiving $(\mathbf{mp}, c_1, c_2, sid)$ from all authorities A_j, first check if all messages are identical. If not, do nothing; otherwise respond as follows. If $\mathsf{D}(c_1) < \mathsf{D}(c_2)$, send $(\mathbf{smaller}, sid, c_1, c_2)$ to all authorities and $\mathcal{S}$. Otherwise, send $(\mathbf{not\ smaller}, sid, c_1, c_2)$ to all authorities and $\mathcal{S}$.

Functionality 5: millionaire problem solver.

Using the ideal functionalities $\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}}$ and $\mathcal{F}_{\mathsf{MP}}$, we can define a protocol $\mathcal{P}_{\mathsf{THVS}}$ which securely realizes $\mathcal{F}_{\mathsf{VS}}$ with a tally-hiding tallying function. This protocol is presented in Protocol 6. The protocol runs with authorities and voters. There are two options: "A" and "B". (See Section 3.3 for a generalization with more than two options.) Each voter publishes two ciphertexts, one corresponding to each option. In addition, the voter supplies a non-interactive zero-knowledge proof that one of his ciphertexts is an encryption of 1 and the other is an encryption of 0. Such a proof can be constructed by applying the subset proof technique from Cramer *et al.* [13] to a proof of plaintext knowledge which in the case of Paillier encryption amounts to a Shoup proof [32]. The authorities apply homomorphic aggregation to obtain two ciphertexts encoding the number of votes for each option. They subsequently invoke $\mathcal{F}_{\mathsf{MP}}$ twice to determine which option has received more votes.

$\mathcal{P}_{\mathsf{THVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$. We demonstrate this by showing that for every adversary $\mathcal{A}$ attacking an execution of the protocol, there exists an adversary-simulator $\mathcal{S}$ attacking the ideal process such that a computationally bounded external environment machine $\mathcal{E}$ cannot determine whether he is interacting with $\mathcal{A}$ and an execution of $\mathcal{P}_{\mathsf{THVS}}$ or with $\mathcal{S}$ and an execution of $\mathcal{F}_{\mathsf{VS}}$. Moreover, we must show that there exists a verifier

Protocol $\mathcal{P}_{\mathsf{THVS}}$

- Initiate by invoking $\mathcal{F}_{\mathsf{PKG}}$ followed by $\mathcal{F}_{\mathsf{SKG}}$.
- All voters $V_i \in \mathbf{V}$ use the system public key to encrypt their vote $v_i = (v_i^{(A)}, v_i^{(B)})$ and generate a proof that the encryption is the encryption of a valid vote. The encryption and the proof are signed and then sent to $\mathcal{F}_{\mathsf{BB}}$. (The adversary $\mathcal{A}$ is assumed to allow them to pass.)
- The authorities $A_j \in \mathbf{A}$ use the cryptosystem's homomorphic property to calculate the aggregates: encryptions c_1 and c_2 of the number of votes for "A" and for "B" respectively.
- The authorities invoke $\mathcal{F}_{\mathsf{MP}}$ twice. Once on input $(\mathbf{mp}, c_1, c_2, sid)$ and once on input $(\mathbf{mp}, c_2, c_1, sid)$.
- If $\mathcal{F}_{\mathsf{MP}}$ returns $\mathbf{smaller}, \mathbf{not\ smaller}$, all authorities send $(\mathbf{message}, sid, (A < B, sig))$ to $\mathcal{F}_{\mathsf{BB}}$. If $\mathcal{F}_{\mathsf{MP}}$ returns $\mathbf{not\ smaller}, \mathbf{smaller}$, all authorities send $(\mathbf{message}, sid, (A > B, sig))$ to $\mathcal{F}_{\mathsf{BB}}$. If $\mathcal{F}_{\mathsf{MP}}$ returns $\mathbf{not\ smaller}, \mathbf{not\ smaller}$, then all authorities send $(\mathbf{message}, sid, (A = B, sig))$ to $\mathcal{F}_{\mathsf{BB}}$. The adversary is assumed to allow all messages to pass. In these messages, sig represents a signature by the authority who sends the message on the tally.
- All voters V_i send $(\mathbf{read}, sid)$ to $\mathcal{F}_{\mathsf{BB}}$.

Protocol 6: tally-hiding voting system.

$\mathcal{V}$ who queries the adversary for a transcript of $\mathcal{P}_{\mathsf{THVS}}$ and who returns 1 with high probability if and only if the protocol was executed correctly and 0 otherwise.

Theorem 2. *Protocol $\mathcal{P}_{\mathsf{THVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ with a tally-hiding tally function in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}})$-hybrid model for non-adaptive adversaries corrupting up to $k - t$ authorities, where k is the number of authorities and t is the threshold of honest authorities.*

The proof is in Appendix B.1

3.2 Ciphertext Lifting and the Millionaire Problem

We now describe a secure protocol for solving the millionaire problem. Let us first summarize threshold decryption of a Paillier or Damgård-Jurik ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$. Here, n is the product of two large primes and s is a small integer (equal to 1 for the Paillier case).

The private decryption exponent, d, is distributed among k parties such that at least t of them must cooperate in order to perform a decryption. Every participant A_i (indexing starts from 1) is in possession of a value s_i, such that (i, s_i) is a point on the $(t-1)$-degree polynomial $f(x) \in \mathbb{Z}[x]$. This polynomial evaluates in zero to the secret exponent: $f(0) = d$ where d satisfies $d = 0 \mod \varphi(n)$ and $d = 1 \mod n$

In addition to a share s_i in the secret exponent, all parties have a public verification key $w_i = w^{\Delta s_i} \mod n^s$ for some fixed w. The verification key binds them to their secret share. Here, $\Delta = k!$.

In order to decrypt a ciphertext $c \in \mathbb{Z}^*_{n^s}$, the parties proceed as follows. Each party publishes a decryption share $c_i = c^{2\Delta s_i} \mod n^s$ along with a non-interactive zero-knowledge proof that $\mathrm{dlog}_{c^2} c_i = \mathrm{dlog}_w w_i$.

Once we have a set S of at least t decryption shares, we can combine them using Lagrange interpolation:

$$c' = \prod_{i \in S} c_i^{2\lambda_i} \mod n^s \quad \text{where } \lambda_i = \Delta \prod_{j \in S \setminus \{i\}} \frac{-j}{i-j} \ .$$

After combining the shares, the randomizer in the ciphertext will have been canceled and the resulting value will be of the form $c' = (1+n)^{4\Delta^2 m} \mod n^s$. Damgård and Jurik describe an algorithm to efficiently compute the plaintext $m \in \mathbb{Z}_{n^{s-1}}^*$ from this value.

Our lifting procedure is founded on the following observation. As soon as the authorities decide on an access structure Λ (a set of size t of indices of the authorities that will proceed with the protocol), these authorities are able to obtain a multiplicative sharing of the plaintext modulo n^2. Each authority A_i computes $c_i = c^{4\Delta^2 \prod_{j \in \Lambda \setminus \{i\}} \frac{-j}{i-j}} \mod n^2$ and then $(1+n)^m = \prod_{i \in \Lambda} c_i \mod n^2$ is guaranteed to hold. A little more juggling results in an additive sharing modulo n where $m = \sum_{i \in \Lambda} u_i \mod n$ is guaranteed to hold. Under favorable conditions on the plaintext, we can find an equivalent additive sharing where no overflow occurs: $m = r + \sum_{i \in \Lambda} u_i'$ (where u_i' remain secret). This sum is subsequently computed homomorphically in ciphertext-domain by first encrypting r and all the shares under the Damgård-Jurik cryptosystem ($s = 2$) and then multiplying all the summands. The protocol is described in full detail in Appendix A.

By contrast, Damgård and Jurik's solution [17] involves regarding a ciphertext $c \in \mathbb{Z}_{n^s}^*$ as though it were a ciphertext in $\mathbb{Z}_{n^{s+1}}^*$: $c = \mathsf{E}_s(m, r) = \mathsf{E}_{s+1}(m + tn^s, r')$. The protocol proceeds to compute t from this value — by first homomorphically adding sufficiently small noise R, then decrypting and lastly by performing the division by n^s on $m + R + tn^s$. The desired ciphertext is obtained by encrypting tn^s and homomorphically calculating $\mathsf{E}_{s+1}(m + tn^s - tn^s, r'')$. However, this solution is cumbersome as it requires expensive range proofs on m and R in order to guarantee on the one hand that no overflow will occur and on the other hand that no information on m is leaked.

In order to use this to solve the millionaire problem, we make the following observation. For the Paillier cryptosystem, multiplication of two ciphertexts modulo n^2 homomorphically corresponds to addition of the plaintexts modulo n. However, for the Damgård-Jurik extension, multiplication of two ciphertexts modulo n^3 homomorphically corresponds to addition of the plaintexts modulo n^2. But modulo n^2, subtraction of a larger value from a smaller value, both smaller than n, will result in a number larger than n. That is, the second digit (in base n) becomes nonzero (in fact, this digit will be equal[2] to $n-1$).

We use this principle to solve the millionaire problem in the following way. Let $\ominus$ denote multiplication with the inverse of the right hand side, which corresponds to subtraction of the right plaintext from the left plaintext. Let $\mathsf{Lift} : \mathbb{Z}_{n^2}^* \to \mathbb{Z}_{n^3}^*$ be the black box lifting procedure that maps $\mathbb{Z}_{n^2}^*$-ciphertexts onto $\mathbb{Z}_{n^3}^*$-ciphertexts while keeping the plaintext intact. Given two ciphertexts c_1 and c_2, we calculate $B = \mathsf{Lift}(c_1 \ominus c_2)$ and $b = \mathsf{Lift}(c_1) \ominus \mathsf{Lift}(c_2)$. Next, we decrypt $A = B \ominus b$. The result can either be different from zero, in which case $c_2 > c_1$, or else 0 and in this case $c_2 \leq c_1$. Aside from this relation, no information is leaked on c_1 and c_2.

This protocol, $\mathcal{P}_{\mathsf{CLMP}}$ is formally presented in Protocol 7. It runs with participants A_i holding shares in a t-out-of-k threshold Paillier cryptosystem. The input consists of two ciphertexts c_1 and c_2. While Lift is invoked as a function, in reality it represents an ideal functionality $\mathcal{F}_{\mathsf{Lift}}$.

This concludes the description of the ciphertext lifting protocol $\mathcal{P}_{\mathsf{CLMP}}$. What remains to be shown is that $\mathcal{P}_{\mathsf{CLMP}}$ is a secure realization of the ideal protocol $\mathcal{F}_{\mathsf{MP}}$ that solves the millionaire

[2] Unless our lifting procedure is employed and a particular edge case is triggered in which case the digit in question equals $n-2$. See Appendix A for details.

Protocol $\mathcal{P}_{\mathsf{CLMP}}$

- The authorities calculate $B = \mathsf{Lift}(c_1 \ominus c_2)$, $\mathsf{Lift}(c_1)$ and $\mathsf{Lift}(c_2)$.
- Next, the authorities compute $b = \mathsf{Lift}(c_1) \ominus \mathsf{Lift}(c_2)$ and $A = B \ominus b$.
- Lastly, the authorities decrypt A and output $\mathsf{D}(c_1) \geq \mathsf{D}(c_2)$ if $\mathsf{D}(A) = 0$ and $\mathsf{D}(c_1) < \mathsf{D}(c_2)$ otherwise.

Protocol 7: ciphertext lifting for the millionaire problem.

problem. This security holds in the hybrid model as the protocol implicitly relies on a system public key whose matching private key is distributed among the participants ($\mathcal{F}_{\mathsf{SKG}}$), as well on the *accessibility* of this public key, for instance by posting it on the bulletin board ($\mathcal{F}_{\mathsf{BB}}$). Moreover, the zero-knowledge proofs which are required for decryption (and possibly for the lifting procedure, depending on its particular instantiation) are made non-interactive and retain their soundness in the random oracle model. We formalize the random oracle model as an ideal functionality $\mathcal{F}_{\mathsf{RO}}$, along the lines of Hofheinz and Müller-Quade [25].

Theorem 3. *Protocol $\mathcal{P}_{\mathsf{CLMP}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{MP}}$ in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Lift}})$-hybrid model against non-adaptive adversaries.*

The proof is in Appendix B.2.

3.3 Winner among r Options

Thus far we have proposed a UC-secure protocol for calculating the winner without the vote count in a vote with two options. It turns out that our particular solution extends to multiple options.

The key insight is that a lifted ciphertext $c \in \mathbb{Z}^*_{n^3}$ is homomorphic in the following sense. Let D represent the Damgård-Jurik decryption and let D' be defined as:

$$\mathsf{D}' : \mathbb{Z}^*_{n^3} \to \{0,1\} \ : \ c \mapsto \begin{cases} 0 \text{ if } \mathsf{D}(c) = 0 \\ 1 \text{ if } \mathsf{D}(c) \neq 0 \end{cases} .$$

Then we have the following homomorphism for ciphertexts $c_1, c_2 \in \mathbb{Z}^*_{n^3}$ which holds with high probability for random $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_{n^3}$ (where $\vee$ represents the OR operation):

$$\mathsf{D}'(c_1^{r_1} \times c_2^{r_2} \mod n^3) = \mathsf{D}'(c_1) \vee \mathsf{D}'(c_2) \ .$$

We can exploit this property to solve the *richest* millionaire problem (as opposed to the *richer* millionaire problem, which is necessarily between two participants) by modifying $\mathcal{P}_{\mathsf{CLMP}}$ in the following way. For all pairs $\{c_i, c_j\}$ of vote counts, the authorities invoke $\mathcal{P}_{\mathsf{CLMP}}$ *up to the decryption of A* once on (c_i, c_j) and once on (c_j, c_i). Let the resulting ciphertexts be denoted $A_{i,j}$ and $A_{j,i}$.

Then, for each original ciphertext $c_i \in \mathbf{O}$, the authorities combine all comparison ciphertexts, each of which is randomized by exponentiation with a random $r_j \xleftarrow{\$} \mathbb{Z}_{n^3}$:

$$A_i \leftarrow A_{i,1}^{r_1} \times \ldots \times A_{i,j}^{r_j} \times \ldots \mod n^3$$

That ciphertext c_i whose A_i decrypts to $\mathsf{D}'(A_i) = 0$ encrypts the largest value because this value is as great or greater than that of any c_j. If there are more than one such A_i, then they are tied winners.

The tally-hiding voting system, $\mathcal{P}_{\mathsf{THVS}}$ invokes this modified version of the richest millionaire protocol on all the ciphertexts c_i that encrypt the total number of votes on option O_i. The richest millionaire protocol $\mathcal{P}_{\mathsf{CLMP}}$ will determine the winner of the vote. The security proofs are easily adapted from the two option case.

4 Self-Tallying Vote

A self-tallying voting system is a voting system in which any interested third party can compute the tally after the last vote has been cast. This can be desirable in small-scale votes (*e.g.*: boardroom votes) where all of the voters or a large part of them are simultaneously voting authorities. The price to pay for this increase in efficiency is that one voter can boycott the vote by refusing to participate. The concept was introduced by Kiayias and Yung [27] and was later improved on by Groth [22].

We follow the scheme of Kiayias–Yung where the randomizers used for ballot encryption are not entirely random, but designed to cancel out when aggregated. In this way, when all ballots are put together, the randomizers disappear and we are left with the tally. We overcome the problem in which the last voter can see the tally before he votes by introducing a control voter. This control voter casts a dummy vote only after all voters have cast their votes. It turns out that this control voter is essential for the UC-security of the scheme.

In the systems of both Kiayias–Yung and Groth, all voters are authorities. In our scheme, however, we assume that some voters may not be authorities and that some authorities may not be voters. As long as at least one authority remains honest, no voter's randomizer can be computed.

We describe the protocol for Paillier encryption with the slight modification that the randomizer is generated deterministically from a public seed r to make a cancellation possible. However, it should be noted that our protocol applies to any group in which the discrete logarithm is hard. (Although, eventually, the tally is obtained by computing a discrete logarithm; we selected the Paillier cryptosystem precisely because it offers a subgroup in which this is easy.)

Every voter V_i possesses a secret exponent x_i, such that $\sum_i x_i = 0$. When all ballots are combined, the randomizers cancel:

$$\prod_i (1+n)^{v_i} r^{x_i n} = (1+n)^{\sum_i v_i} \mod n^2 \ .$$

At the start of the protocol, the authorities decide on a base $w \in \mathbb{Z}_n$ for the verification keys. Next, every authority A_j chooses an $x_{i,j}$ for every voter V_i such that $\sum_i x_{i,j} = 0$. This value is sent to voter V_i while $w^{x_{i,j}} \mod n$ is made public. After all authorities have done this, the voter obtains the secret exponent $x_i = \sum_j x_{i,j}$ and any scrutineer can determine that voter's verification key $w_i = \prod_j w^{x_{i,j}} \mod n$.

In order to start voting, the voters must first select a public seed r, for example by taking the hash value of all the messages on the bulletin board up until that point. There is a new public seed for every round of vote casting. It should be noted that $\mathrm{dlog}_w r$ must not be knowable by any subset of authorities, as knowledge of this value can be used to recover individual votes.

A vote is encrypted using the system public key. A non-interactive proof is generated that the vote is correctly formed and the secret exponent was used (these two claims may be proved in a single proof). The result is signed and sent to the bulletin board. A concise overview of this protocol, $\mathcal{P}_{\mathsf{STVS}}$, is presented in Protocol 8.

Protocol $\mathcal{P}_{\mathsf{STVS}}$

- Initiate by invoking $\mathcal{F}_{\mathsf{PKG}}$ followed by $\mathcal{F}_{\mathsf{SKG}}$.
- All authorities A_j choose random values $x_{i,j} \xleftarrow{\$} \mathbb{Z}_{n^2}$ for all voters V_i such that $\sum_i x_{i,j} = 0$. They send these values to $\mathcal{F}_{\mathsf{BB}}$ but encrypted with that voter's public key. Also, they publish $w^{x_{i,j}}$ for each $x_{i,j}$.
- The scrutineers check that $\sum_i x_{i,j} = 0$ for all j by calculating $\prod_i w^{x_{i,j}} \stackrel{?}{=} 1$. Also, the scrutineers calculate every voter's verification key $w_i = w^{\sum_j x_{i,j}} = \prod_j w^{x_{i,j}}$.
- All voters V_i read their messages from the authorities and determine their own secret exponent $x_i = \sum_j x_{i,j}$.
- In order to vote, the voters select a public random seed r, for example by querying the random oracle. Next, each voter encrypts his vote using r^{x_i} for randomizer. This encryption is posted to $\mathcal{F}_{\mathsf{BB}}$ along with a proof that the ballot encrypts an allowable vote and that the correct secret exponent was used, and with a signature on the previous two objects. When all real voters have cast their vote, the control voter casts his dummy vote and proves that it does not alter the tally and that it uses the correct secret exponent.
- All participants send a $(\mathbf{read}, sid)$ message to $\mathcal{F}_{\mathsf{BB}}$ and combine all the votes and calculate the tally t.

Protocol 8: self-tallying voting system.

We can only prove that $\mathcal{P}_{\mathsf{STVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ under an additional assumption: the control voter V_n is uncorruptable. The reason is that the honest simulated voters V_i must cast their votes *before* the authorities instruct $\mathcal{F}_{\mathsf{VS}}$ to tally. Therefore, their votes cannot be made to correspond to the tally unless they cast their votes *after* the invocation of $\mathcal{F}_{\mathsf{VS}}$ — in which case the simulated adversary $\mathcal{A}$ can pick up on the time and order difference. To prevent this, the simulated voters cast random votes, which are then "corrected" by the control voter.

Theorem 4. *$\mathcal{P}_{\mathsf{STVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ with an additive tallying function against a non-adaptive adversary who has not corrupted the control voter V_n in the $(\mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{RO}})$-hybrid model.*

The proof is in the Appendix B.3.

5 Authenticated Voting Credentials

One of the first proposals for the construction of anonymous voting systems was by using *anonymous credentials* [10]: strings of characters whose membership to a certain class of strings is

easy to verify. However, without some trapdoor information pertaining to that class, generating new or other members is intractable.

The idea is as follows. The authorities generate one credential for each eligible voter and securely communicate them to their intended recipients. During voting, the voters log in to the bulletin board via an anonymous channel and cast their votes anonymously. Only those votes that are accompanied by a valid credential are tallied. In an internet setting, the anonymous channel can be approximated by using an anonymization network such as Tor.

However, in recent years credential-based voting systems have fallen out of favor due to their weak security properties. First, the anonymous access to the bulletin board is difficult to realize. The adversary who monitors a target's outgoing traffic as well as the bulletin board's incoming traffic is able to correlate the two and recover the voter's vote. Moreover, to date, there exists no security guarantee for credential-based voting systems. While there do exist UC-secure credential systems [5, 8, 34], the inclusion of a voting procedure introduces a whole new challenge.

Our contribution is twofold. First, at a conceptual level we introduce the concept of "authenticated voting credentials" for credential-based voting systems. The idea is to protect against the adversary who intercepts the credential or modifies the vote, or who modifies the protocol transcript before presenting it to the verifier. The voter proves his eligibility to vote not by releasing the credential, but by proving knowledge of it. This interactive proof doubles as a MAC on the vote that is cast. The adversary is not able to modify the proof so that it is authenticated by a different vote.

Second, on the level of protocol design, we propose an instantiation of this idea. It is based on Chaumian blind RSA signatures [9], using a credential withdrawal protocol introduced by Ferguson [18]. A credential consists of a preimage a, a one-way function of the preimage $A = ag^{f(h^a)} \mod n$, and an RSA signature on this value $S = A^{1/v} \mod n$. In these expressions, g and h are publicly known base values in $\mathbb{Z}_n^*$; f is a suitable one-way function; n is an RSA modulus whose factorization is known only to the bank (or, in our case, to the authorities); v is the public RSA exponent and $1/v = v^{-1} \mod \varphi(n)$ is the private RSA exponent.

A simplified version of Ferguson's protocol for credential withdrawal is given by Protocol 9. In this figure, H represents a random oracle (instantiated by a hash function) and f a generic one-way function. Compared to Ferguson's original version, the first step has been made non-interactive using the Fiat-Shamir heuristic [19].

In a voting system, the private RSA exponent would be distributed among the authorities. The RSA signature is generated using Shoup's protocol [32]. The message from $\mathcal{B}$ to $\mathcal{A}$ actually consists of several messages as each of the authorities communicate their share in $\bar{A}$, the blind signature, to the recipient.

In order to use the credential, the voter must not release the entire string but rather prove knowledge of it. Guillou and Quisquater have proposed an interactive proof of knowledge of an RSA preimage [23], which perfectly matches our purposes. For the sake of completeness, this proof, Protocol 11, is included in the appendix. The prover possesses an RSA signature and proves that he knows it while not releasing the signature itself. In Camenisch-Stadler notation:

$$\mathrm{ZKPoK}\{(S) \,:\, S^v = A \mod n\} \ .$$

The transcript of the protocol in which the credential is spent, is unlinkable to the transcript of the protocol that created it. The reason is that the transcript of the withdrawal protocol contains no information on the credential. A given withdrawal transcript $(b, c, \bar{A})$ may have

Protocol $\mathcal{P}_{\mathsf{CW}}$

Public knowledge: n, v, g, h.
Private knowledge for $\mathcal{B}$: $1/v = v^{-1} \mod \varphi(n)$.

$\mathcal{A}$		$\mathcal{B}$
$a_1, \gamma, \sigma \xleftarrow{\$} \mathbb{Z}_n^*$		
$b \leftarrow \gamma^v a_1 g^\sigma \mod n$		
$a_2 \leftarrow \mathsf{H}(b)$		
$a \leftarrow a_1 a_2 \mod n$		
$c \leftarrow f(h^a) - \sigma$		
	$\xrightarrow{b, c}$	
		$a_2 \leftarrow \mathsf{H}(b)$
		$\bar{A} \leftarrow (b a_2 g^c)^{1/v} \mod n$
	$\xleftarrow{\bar{A}}$	
$S \leftarrow \bar{A}\gamma^{-1} \mod n$		
$S^v \stackrel{?}{=} a g^{f(h^a)} \mod n$		

Protocol 9: protocol for credential withdrawal.

created *any* valid credential (a, A, S). To see this, choose $\sigma = f(h^a) - c$; $\gamma = \bar{A}S^{-1} \mod n$ and $a_1 = b\gamma^{-1} g^{-\sigma} \mod n$.

We now describe the entire authenticated voting credential protocol. The protocol is started by invocations of the key generation mechanisms. $\mathcal{F}_{\mathsf{PKG}}$ generates a public and private key pair for encryption and signing for every protocol participant and distributes the public keys such that each participant has a list of public keys of authorities and a list of public keys of eligible voters. $\mathcal{F}_{\mathsf{SKG}}$ generates an RSA public key and distributes shares in the private key to the authorities along Shoup's protocol [32].

In the second phase of the protocol, the voters withdraw their credential. They initiate the credential withdrawal protocol $\mathcal{P}_{\mathsf{CW}}$ (in the role of $\mathcal{A}$) and sign their first message. The authorities (in the role of $\mathcal{B}$) assign credentials only to eligible voters — whom they can tell from non-eligible voters by the signature which must match a public key in the list of eligible voter public keys. Each participant performs the instructions of $\mathcal{B}$ from $\mathcal{P}_{\mathsf{CW}}$ separately, returning not the signature but the signature share on A and encrypting it with that voter's public key, along with a proof of correct exponentiation. If at least t authorities do this honestly, the voter will be able to reconstruct the signature $S = A^{1/v} \mod n$ which is an essential part of the credential.

At this point, every eligible voter (or those who have initiated the withdrawal process) is in possession of a valid credential $C = (a, A, S)$ which is unlinkable to the transcript of the withdrawal process that created it. During the voting phase, the voters log in to the bulletin board anonymously and post one message containing all of the following. First, their vote v_i. Second, the credential preimage a. And last, a non-interactive proof of knowledge of $S = A^{1/v} = (a g^{f(h^a)})^{1/v} \mod n$. This non-interactive proof is generated by applying the Fiat-Shamir heuristic to the proof of RSA signature knowledge $\mathcal{P}_{\mathsf{RSA}}$, where the challenge value e is

computed as the hash of the proof claim, the proof's first message, and the vote that is cast:

$$e = \mathsf{H}(A \parallel D \parallel v_i) \ .$$

Anyone can compute the tally by counting only those votes that are accompanied by a valid and unique credential. A credential cannot be reused as the value a is fixed. Only eligible voters possess credentials because the authorities only assume their role in the credential withdrawal protocol when eligible voters request a credential (and they make sure not to issue multiple credentials to any one voter). Since access to the bulletin board is anonymous and since the credentials cannot be linked to the withdrawal process that created them, voter anonymity is preserved. Lastly, the votes that were cast conform to the credential holder's intention because otherwise the proof will not check out. The protocol is formally presented in Protocol 10.

Protocol $\mathcal{P}_{\mathsf{AVCS}}$

- Initiate by invoking $\mathcal{F}_{\mathsf{PKG}}$ followed by $\mathcal{F}_{\mathsf{SKG}}$.
- All voters V_i that wish to vote post to $\mathcal{F}_{\mathsf{BB}}$ the pair of values b and c, which constitutes the first message in $\mathcal{P}_{\mathsf{CW}}$. This pair is signed by the voter.
- All authorities A_j, upon seeing this initial message from V_i, compute $\bar{A}_j = (b\mathsf{H}(b)g^c)^{s_j} \mod n$ (where s_j is the share in the private exponent for authority A_j). They send this value, along with a proof of correct exponentiation to $\mathcal{F}_{\mathsf{BB}}$ after encrypting it with V_i's public key.
- All voters V_i compute the signature in their credentials by calculating $\bar{A} \leftarrow \prod_j \bar{A}_j^{2\Delta s_j \mu_j} \mod n$ where μ_j is the Lagrange coefficient, and then $S \leftarrow \bar{A}\gamma^{-1} \mod n$.
- All voters cast their votes anonymously by sending (v_i, a, Z) to $\mathcal{F}_{\mathsf{BB}}$, where Z is the transcript of the proof of knowledge of $S = A^{1/v} = (ag^{f(h^a)})^{1/v} \mod n$ made non-interactive by setting the challenge value to $e = \mathsf{H}(A \parallel D \parallel v_i)$.
- Anyone, including the scrutineers, can compute the tally by sending a $(\mathbf{read}, sid)$ message to $\mathcal{F}_{\mathsf{BB}}$ and tallying the votes from all the messages (v, a, Z) where Z is a valid proof for v and a (and where a was not used before).

Protocol 10: authenticated voting credential system.

We note that this protocol is not *fair*: voters can observe the current tally before casting their own vote. Knowledge of this partial tally may influence the content of their vote or even whether or not they choose to cast it. This drawback is addressed by universal composability. If the times at which the votes are cast, can influence which votes are cast, then the protocol adversary $\mathcal{A}$ can exploit this property and generate votes where the tallies are significantly different from the tallies that would have been produced by $\mathcal{F}_{\mathsf{VS}}$.

In order to make the voting system fair — and, hence, in order for the system to possibly be universally composable — voters must not be able to read the bulletin board during the voting phase. Conceptually, we modify $\mathcal{F}_{\mathsf{BB}}$ to offer this property. In particular: if $\mathcal{F}_{\mathsf{BB}}$ receives a $(\mathbf{close}, sid)$ from at least one honest authority, $\mathcal{F}_{\mathsf{BB}}$ starts ignoring all $(\mathbf{read}, sid)$ requests. When at least one honest authority sends a $(\mathbf{open}, sid)$ message, $\mathcal{F}_{\mathsf{BB}}$ starts responding again.

This can be implemented on top of $\mathcal{F}_{\mathsf{BB}}$ by introducing an interfacing layer that relays messages between $\mathcal{F}_{\mathsf{BB}}$ and the external environment but blocks **read** messages when it is not readable.

The adversary who can read, block or falsify all incoming traffic to $\mathcal{F}_{\mathsf{BB}}$ breaks the security of the protocol because he can trivially see which vote is coming from which voter. We can only prove security under a far weaker adversarial model. In this model, the adversary can no longer read, block or falsify messages sent by voters to the bulletin board. In other words, except for instructing the corrupted parties, the adversary has become *passive*. In order to account for this adversary, we modify the specification of the bulletin board functionality $\mathcal{F}_{\mathsf{BB}}$ to instantly store any message by any participant without reference to the sender and without letting it pass through the adversary. In this modified $\mathcal{F}_{\mathsf{BB}}$ model, we are able to prove the UC-security and universal verifiability of $\mathcal{P}_{\mathsf{AVCS}}$.

Theorem 5. *Protocol $\mathcal{P}_{\mathsf{AVCS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ with an additive tallying function in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}})$-hybrid model for a passive, non-adaptive adversary corrupting up to $k - t$ authorities, where k is the number of authorities and t is the threshold of honest authorities.*

The proof is in Appendix B.4.

Universal composability does not require authenticated voting credentials. As long as the bulletin board offers anonymous, append-only access and is unreadable during voting but otherwise readable by all participants, any credential system can be used for UC-secure voting. On the other hand, if the adversary is able to control what the bulletin board shows to readers during the tallying process, then authenticated voting credentials are necessary for UC-security. More importantly, they are necessary for universal verifiability. Since the adversary controls the transcript that is verified by the verifier, he could otherwise pretend the (unauthenticated) credentials were used to cast different votes and so convince the verifier of a false tally. This is impossible if the credentials are authenticated by the votes.

6 Conclusion

In this paper we have introduced three new techniques for electronic voting as well as a unifying framework for their analysis. In particular, we have proposed a tally-hiding voting system, a self-tallying voting system and an authenticated voting credential system and we have proved the UC-security of each of these systems as well as their universal verifiability.

The UC-security of the systems is proved under various adversarial models. Our tally-hiding system is secure against the strongest adversary: one who can observe, block or falsify any voter's messages and corrupt any number of voters and up to $k - t$ authorities, where k is the number of authorities and t is the threshold of honest authorities. In the self-tallying vote, the adversary is nearly as strong. The only additional constraint is that there is a control voter who casts the last dummy vote and who cannot be corrupted. In both cases, he is allowed to read, block and falsify voters' messages, extending the power of the adversary compared to Groth's model [22]. On the other hand, Groth's model includes protection against adaptive adversaries in the erasure and in the erasure-free model. The adversary in the authenticated voting credential system is far weaker as he is not able to observe voters' messages — not even indirectly by observing the bulletin board as this may be closed for reading. In all cases, we assume the adversary is computationally bounded and that his power to corrupt is somehow constrained.

We offer a formalism of universal verifiability which is compatible with universal composability in two senses: first, the verifier (from the universal verifiability paradigm) interacts with the adversary (from the universal composability paradigm); second, universally verifiable protocols are composable just as universally composable protocols are. While universal verifiability is independent of the ideal bulletin board functionality, it is not independent of the participant key generation functionality nor of the random oracle functionality. The verifier $\mathcal{V}$ needs to verify the authenticity of, at the very least, the voters' messages. The random oracle is necessary to make the proofs non-interactive. While the adversary-simulator $\mathcal{S}$ may simulate a random oracle to convince a real protocol adversary $\mathcal{A}$ of false claims, he cannot convince a skeptical verifier in the same way. In order to convince a verifier of the correctness of a protocol, the adversary must invoke a random oracle that is trusted by the verifier. To make this non-interactive, a cryptographic hash function is a reasonable approximation.

Acknowledgements

The authors would like to thank the anonymous reviewers for their useful comments and countless colleagues for fruitful discussions and critical proof-readings. This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). Alan Szepieniec is supported by a doctoral grant from the Agency for Innovation by Science and Technology in Flanders (IWT). In addition, this work was supported by the Flemish Government, FWO Beveiligde Uitvoering G.0130.13 and by the European Commission through the ICT programme under contract H2020-ICT-2014-645421 ECRYPT CSA, H2020-MSCA-ITN-2014-643161 ECRYPT NET and FP7-ICT-2013-645622 PRACTICE.

References

1. Adida, B.: Advances in cryptographic voting systems. Ph.D. thesis, Massachusetts Institute of Technology (2006)
2. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. IACR Cryptology ePrint Archive 2014, 75 (2014)
3. Benaloh, J.: Verifiable secret-ballot elections. Ph.D. thesis, Yale University (1987)
4. Benaloh, J.C., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: Leighton, F.T., Goodrich, M.T. (eds.) STOC. pp. 544–553. ACM (1994)
5. Camenisch, J., Casati, N., Groß, T., Shoup, V.: Credential authenticated identification and key exchange. In: Rabin, T. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6223, pp. 255–276. Springer (2010)
6. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. pp. 136–145. IEEE Computer Society (2001)
7. Canetti, R., Gennaro, R.: Incoercible multiparty computation (extended abstract). In: FOCS. pp. 504–513. IEEE Computer Society (1996)
8. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 4117, pp. 78–96. Springer (2006)
9. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO. pp. 199–203. Plenum Press, New York (1982)
10. Chaum, D.: Elections with unconditionally-secret ballots and disruption equivalent to breaking rsa. In: Günther [24], pp. 177–182

11. Chevallier-Mames, B., Fouque, P.A., Pointcheval, D., Stern, J., Traoré, J.: On some incompatible properties of voting schemes. In: Towards Trustworthy Elections. pp. 191–199 (2010)
12. Cohen, J.D.: Improving privacy in cryptographic elections. Citeseer (1986)
13. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 839, pp. 174–187. Springer (1994)
14. Cramer, R., Franklin, M.K., Schoenmakers, B., Yung, M.: Multi-autority secret-ballot elections with linear work. In: Maurer, U.M. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1070, pp. 72–83. Springer (1996)
15. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multiauthority election scheme. In: Fumy, W. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1233, pp. 103–118. Springer (1997)
16. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Kim, K. (ed.) Public Key Cryptography. Lecture Notes in Computer Science, vol. 1992, pp. 119–136. Springer (2001)
17. Damgård, I., Jurik, M.: Client/server tradeoffs for online elections. In: Naccache and Paillier [28], pp. 125–140
18. Ferguson, N.: Single term off-line coins. In: Helleseth, T. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 765, pp. 318–328. Springer (1993)
19. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
20. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT. Lecture Notes in Computer Science, vol. 718, pp. 244–251. Springer (1992)
21. Groth, J.: Efficient maximal privacy in boardroom voting and anonymous broadcast. In: Juels, A. (ed.) Financial Cryptography. Lecture Notes in Computer Science, vol. 3110, pp. 90–104. Springer (2004)
22. Groth, J.: Evaluating security of voting schemes in the universal composability framework. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS. Lecture Notes in Computer Science, vol. 3089, pp. 46–60. Springer (2004)
23. Guillou, L.C., Quisquater, J.J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: Günther [24], pp. 123–128
24. Günther, C.G. (ed.): Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings, Lecture Notes in Computer Science, vol. 330. Springer (1988)
25. Hofheinz, D., Müller-Quade, J.: Universally composable commitments using random oracles. In: Naor, M. (ed.) TCC. Lecture Notes in Computer Science, vol. 2951, pp. 58–76. Springer (2004)
26. Jonker, H.L., de Vink, E.P.: Formalising receipt-freeness. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC. Lecture Notes in Computer Science, vol. 4176, pp. 476–488. Springer (2006)
27. Kiayias, A., Yung, M.: Self-tallying elections and perfect ballot secrecy. In: Naccache and Paillier [28], pp. 141–158
28. Naccache, D., Paillier, P. (eds.): Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings, Lecture Notes in Computer Science, vol. 2274. Springer (2002)

29. Ohkubo, M., Miura, F., Abe, M., Fujioka, A., Okamoto, T.: An improvement on a practical secret voting scheme. In: Mambo, M., Zheng, Y. (eds.) ISW. Lecture Notes in Computer Science, vol. 1729, pp. 225–234. Springer (1999)
30. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999)
31. Sako, K., Kilian, J.: Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In: Guillou, L.C., Quisquater, J.J. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 921, pp. 393–403. Springer (1995)
32. Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1807, pp. 207–220. Springer (2000)
33. Szepieniec, A., Preneel, B.: New techniques for electronic voting. In: Wallach and Mebane [35], pp. 46–69, https://www.usenix.org/conference/jets15
34. Trolin, M.: A universally composable scheme for electronic cash. In: Maitra, S., Madhavan, C.E.V., Venkatesan, R. (eds.) INDOCRYPT. Lecture Notes in Computer Science, vol. 3797, pp. 347–360. Springer (2005)
35. Wallach, D.S., Mebane, W. (eds.): 2015 USENIX Journal of Election Technology and Systems (JETS '15), Washington, D.C., USA, August 11, 2015, vol. 3. USENIX Association (2015), https://www.usenix.org/conference/jets15
36. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: FOCS. pp. 160–164. IEEE Computer Society (1982)

A Ciphertext Lifting (and the Millionaire Problem)

We now describe an alternative to Damgård and Jurik's ciphertext lifting procedure [17]. One key ingredient is that the authorities decide on an access structure Λ (a set of t indices indentifying the authorities that will perform the computation) in advance. When such an access structure has been decided, each of the authorities uses their private exponent s_i to compute a share c_i starting from a ciphertext c as follows:

$$c_i \leftarrow c^{4\Delta^2 s_i \prod_{j \in \Lambda \setminus \{i\}} \frac{-j}{i-j}} \mod n^2 \ .$$

These shares constitute a multiplicative sharing modulo n^2 as the following equation is guaranteed to hold thanks to Lagrange:

$$(1+n)^m = \prod_{i \in \Lambda} c_i \mod n^2 \ .$$

In the next step of the protocol, each authority divides their share into two digits base n: $c_i = a_i + b_i n$ where $a_i, b_i < n$. The a_i are published; the b_i are kept secret. It is easy to see that the a_i do not leak information on the plaintext m. Moreover, the proof of correct exponentiation which is used to prove the correctness of a decryption share, can be performed modulo n rather than modulo n^2 to yield a zero-knowledge proof that a_i is correctly formed.

The authorities proceed to compute an additive sharing modulo n of the plaintext as follows. Each authority in the access structure computes their share u_i:

$$u_i \leftarrow b_i \prod_{j \in \Lambda \setminus \{i\}} a_j \mod n \ .$$

Anyone can compute $1 + nr = \prod_{i \in \Lambda} a_i \mod n^2$. At this point, the following equality is guaranteed to hold:

$$m = r + \sum_{i \in \Lambda} u_i \mod n \ .$$

In addition to computing their shares u_i, the authorities commit to their shares by publishing $\mathsf{E}(u_i)$. Moreover, they choose some randomness to encrypt r as well. At this point the authorities use the regular threshold decryption procedure to prove that the value

$$c^{-1}\mathsf{E}(r) \prod_{i \in \Lambda} \mathsf{E}(u_i) \mod n^2$$

does indeed decrypt to 0 — implying that the commitments to their shares are correct.

The next step is to obtain an additive sharing of the message *without modular reduction*. This step only works by leaking the most significant few bits of the plaintext. While this property is not desirable (as it precludes UC-security), we argue that this is a minor compromise considering the size of the modulus and the fact that this lifting procedure's most pertinent use case is comparing plaintexts — *i.e.* discovering certain information about the most significant bits.

We use a safety parameter $\varsigma \geq 2$ to divide the range $\mathbb{Z}_n$ into a *secret zone* $S = [0; \frac{n}{\varsigma}) \cap \mathbb{Z}_n$ and a *danger zone* $D = [n - \frac{n}{\varsigma}; n) \cap \mathbb{Z}_n$. These sets have the property that any secret element added to a non-dangerous element is guaranteed not to overflow. Symbolically:

$$\forall\, s \in S, d \in \mathbb{Z}_n \backslash D \ .\ s + d < n \ .$$

We use this principle to find another sharing of the plaintext whose equation holds with and without modular reduction. In particular, the sum of all the secret shares belongs to the safe zone, and the new public remainder r' does not belong to the danger zone. In particular, each authority in the access structure chooses a new share $u'_i \xleftarrow{\$} \left\{0, \ldots, \left\lfloor \frac{n}{\varsigma\, t} \right\rfloor\right\}$ at random. (Bear in mind that $t = |\Lambda|$.) The difference $\delta_i = u_i - u'_i$ is made public along with commitments such that anyone can verify that $\mathsf{E}(u_i) = \mathsf{E}(\delta_i)\mathsf{E}(u'_i) \mod n^2$. The new remainder r' is computed as $r' = r + \sum \delta_i \mod n$. Thus we have an additive sharing which is guaranteed with high probability not to overflow:

$$r' + \sum_{i \in \Lambda} u'_i < n \ .$$

Of course, there is a chance that the procedure as described above might overflow after all and produce $\mathsf{E}(m+n)$ rather than $\mathsf{E}(m)$. This occurs whenever $m < \sum_{i \in \Lambda} u'_i$; the probability of this event is $\frac{1}{2\varsigma}$. It is possible to make this probability small by increasing the safety parameter ς, but this comes at the cost of leaking more information on the plaintext.

Whenever this event occurs, it will be signalled by $r' \in D$ (although $r' \in D$ does not automatically imply $m < \sum_{i \in \Lambda} u'_i$). If the authorities discover that $r' \in D$ and if it is known that $m \notin D$, then they can circumvent the failure by adding $\lfloor \frac{n}{\varsigma} \rfloor$ to r', performing the reduction modulo n, lifting, and homomorphically subtracting $\lfloor \frac{n}{\varsigma} \rfloor$ from the lifted ciphertext.

Of course, this does leave open the question what to do when it is not known whether $m \notin D$ and r' is found to lie in D. We would argue that despite the lack of robustness against errors with respect to all plaintexts, this lifting procedure does offer a practical solution to most use cases, especially considering that ς can be balanced against the known distribution of m. More importantly, however, this drawback does not seem to impact the solution to the millionaire problem. We now demonstrate this.

Without loss of generality, we can assume that $m_1 \notin S$ and $m_2 \notin S$, because we can homomorphically add $\lfloor \frac{n}{\varsigma} \rfloor$ to both plaintexts at no cost. However, if m_1 or m_2 happens to lie in D, then the addition of $\lfloor \frac{n}{\varsigma} \rfloor$ will cause overflow, breaking the protocol. If this is the case then the solution is to use a larger modulus n.

The protocol performs lifting on three numbers: $C_1 = \mathsf{Lift}(c_1), C_2 = \mathsf{Lift}(c_2)$ and $B = \mathsf{Lift}(c_1 \ominus c_2)$. The first two are guaranteed to succeed with this lifting procedure. However, the ciphertext B may encode a plaintext that is larger than n, depending on the particular relation between m_1 and m_2. In particular, with the application of the addition trick from the paragraph before last, we know that B satisfies:

$$\mathsf{D}(B) = \begin{cases} m_1 - m_2 & \text{if} \quad m_1 \geq m_2 \\ 2n + m_1 - m_2 & \text{if} \quad m_2 - m_1 \in S \backslash \{0\} \\ n + m_1 - m_2 & \text{else} \end{cases} .$$

However, $b = C_1 \ominus C_2$ will satisfy:

$$\mathsf{D}(b) = \begin{cases} m_1 - m_2 & \text{if} \quad m_1 \geq m_2 \\ n^2 + m_1 - m_2 & \text{if} \quad m_2 - m_1 \in S \backslash \{0\} \\ n^2 + m_1 - m_2 & \text{else} \end{cases} .$$

And hence the difference $A = B \ominus b$ will satisfy:

$$\mathsf{D}(A) = \begin{cases} 0 & \text{if} \quad m_1 \geq m_2 \\ 2n & \text{if} \quad m_2 - m_1 \in S \backslash \{0\} \\ n & \text{else} \end{cases} .$$

Which implies in particular that the inequality test at the conclusion of the millionaire problem solution still holds: $\mathsf{D}(A) \neq 0 \Rightarrow m_1 < m_2$ and conversely $\mathsf{D}(A) = 0 \Rightarrow m_1 \geq m_2$. In order to further reduce the leakage of information, the ciphertext A must be raised to an unknown and random (but provably nonzero) exponent prior to decryption.

B Universally Composable and Verifiable Voting Systems

B.1 Security and Verifiability of the Tally-Hiding Voting System

Proof of Theorem 2.
Correctness. Correctness follows from construction.

UC-security. We show that no environment $\mathcal{E}$ can tell the difference between an execution of the protocol under attack by a real protocol adversary $\mathcal{A}$ or an invocation of the ideal functionality $\mathcal{F}_{\mathsf{VS}}$ under attack by an adversary-simulator $\mathcal{S}$.

The adversary-simulator $\mathcal{S}$ runs with dummy voters $\hat{V}_i$ and dummy authorities $\hat{A}_i$. The honest dummy voters relay their inputs from $\mathcal{E}$ directly to the ideal functionality $\mathcal{F}_{\mathsf{VS}}$. The corrupted dummy voters and authorities are controlled by the adversary-simulator $\mathcal{S}$. At the end of the experiment, $\mathcal{E}$ reads the outputs of the voters.

The adversary-simulator simulates an execution of the protocol $\mathcal{P}_{\mathsf{THVS}}$ and simulates the real protocol adversary $\mathcal{A}$. The protocol runs with simulated voters V_i and authorities A_j, some of which are controlled by the real process adversary $\mathcal{A}$ and the others of which are controlled by the adversary-simulator $\mathcal{S}$. The set of corrupt dummy participants (i.e.: participants $\hat{V}_i$ or $\hat{A}_i$ that are controlled by $\mathcal{S}$) matches the set of corrupt simulated participants (i.e.: controlled by $\mathcal{A}$). The adversary-simulator $\mathcal{S}$ proceeds as follows.

– $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and $\mathcal{E}$.
– $\mathcal{S}$ starts by simulating the invocation of $\mathcal{F}_{\mathsf{PKG}}$ and $\mathcal{F}_{\mathsf{SKG}}$. $\mathcal{S}$ uses $\mathcal{F}_{\mathsf{BB}}$ throughout. As a result, $\mathcal{S}$ knows *all* private keys and shares in the system private key, in addition to all public keys.
– If one of the simulated voters V_i that is controlled by the real process adversary $\mathcal{A}$ casts a ballot b_i and a proof p_i, the adversary-simulator $\mathcal{S}$ checks the proof and if it is valid, decrypts b_i to obtain V_i's vote, v_i. Next, $\mathcal{S}$ has $\hat{V}_i$ send $(\mathbf{vote}, sid, v_i, \hat{V}_i)$ to $\mathcal{F}_{\mathbf{VS}}$. $\mathcal{S}$ follows up with a $(\mathbf{no\text{-}block}, sid, \hat{V}_i)$ message.
– If the adversary-simulator $\mathcal{S}$ receives $(\mathbf{vote}, sid, \hat{V}_i)$ from $\mathcal{F}_{\mathsf{VS}}$ pertaining to an uncorrupted voter $\hat{V}_i$, $\mathcal{S}$ simulates V_i's vote in the simulated protocol by casting a zero vote $v_i = (0, 0)$ to the bulletin board.
– If the authorities in the simulated protocol invoke $\mathcal{F}_{\mathsf{MP}}$, $\mathcal{S}$ lets all $\hat{A}_i$'s under his control send $(\mathbf{tally}, sid)$ to $\mathcal{F}_{\mathsf{VS}}$, which will produce a tally t. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{MP}}$ for the simulated protocol by presenting the authorities with an answer conformant with t. The dummy voters $\hat{V}_i$ receive the tally from $\mathcal{F}_{\mathsf{VS}}$.

The real protocol adversary $\mathcal{A}$ has no way to determine whether or not he is being simulated by $\mathcal{S}$. The only messages that might indicate that he is being simulated, are the encryptions of votes that are not conformant to the winner coming from $\mathcal{F}_{\mathsf{MP}}$. However, if $\mathcal{A}$ can tell that the vote encryptions do not conform to the tally, then he either breaks the semantic security of the cryptosystem or has access to more than $k - t$ shares in the private system key.

Similarly, the environment $\mathcal{E}$ has no way of detecting whether there is a adversary-simulator $\mathcal{S}$ between himself and the real protocol adversary $\mathcal{A}$. The tallies on the tapes of the voters V_i and authorities A_j are the same in both cases. All $\mathcal{E}$'s queries are answered honestly by a real protocol adversary $\mathcal{A}$ who has no way of knowing whether or not he is being simulated. Moreover, $\mathcal{E}$ is subject to the same computational constraints, which in particular imply that $\mathcal{E}$ cannot break the semantic security of the cryptosystem either.

Universal verifiability. We show this by constructing a verifier $\mathcal{V}$ which verifies an execution of the protocol.

$\mathcal{V}$ does not need to verify the correct execution of $\mathcal{F}_{\mathsf{PKG}}$ or $\mathcal{F}_{\mathsf{SKG}}$ as these are trusted functionalities. The ballots that are cast are accompanied by zero-knowledge proofs which prove their correct composition. Moreover, they are signed, which authenticates the eligibility of their originator. $\mathcal{V}$ can calculate the aggregates of the vote encryptions himself and verify that the input to $\mathcal{F}_{\mathsf{MP}}$ is the correct input. He does not need to verify the execution of $\mathcal{F}_{\mathsf{MP}}$ as it is a trusted functionality. Lastly, $\mathcal{V}$ checks whether the tally sent to the bulletin board conforms with the answer from $\mathcal{F}_{\mathsf{MP}}$. □

B.2 Security and Verifiability of the Millionaire Problem Solution based on Lifting

Proof of Theorem 3.
Correctness. Correctness follows from construction. Let $d = m_1 - m_2$, then we have:

$$A = B \ominus b = \mathsf{Lift}(c_1 \ominus c_2) \ominus (\mathsf{Lift}(c_1) \ominus \mathsf{Lift}(c_2))$$

$$= \mathsf{Lift}(\mathsf{E}(m_1) \ominus \mathsf{E}(m_2)) \ominus (\mathsf{Lift}(\mathsf{E}(m_1)) \ominus \mathsf{Lift}(\mathsf{E}(m_2)))$$

$$= \mathsf{E}(m_1 - m_2 \mod n) \ominus \mathsf{E}(m_1 - m_2 \mod n^2)$$

$$\mathsf{E}(n+d \mod n) \ominus \mathsf{E}(n^2+d \mod n^2) = \mathsf{E}(rn) \ .$$

If $m_1 < m_2$, then $r \neq 0$ (and in fact $r = 1$). On the other hand, if $m_1 \geq m_2$, then $r = 0$.

UC-security. We demonstrate that for every adversary $\mathcal{A}$ that attacks the protocol, there exists an adversary-simulator $\mathcal{S}$ that attacks the ideal functionality with equal success such that any computationally bounded external environment $\mathcal{E}$ cannot determine whether it is interacting with $\mathcal{A}$ and the protocol, or with $\mathcal{S}$ and the ideal functionality.

In the beginning of the experiment, $\mathcal{E}$ reads the system public key. He then chooses a pair of plaintexts m_1 and m_2 and encrypts them using the public key to obtain c_1 and c_2. These values are sent to all participants. At the end of the experiment, the participants communicate the result to $\mathcal{E}$. This implicit reliance on a pre-distributed private key is made explicit if the protocol invokes $\mathcal{F}_{\mathsf{SKG}}$. The implicit reliance that this public key is accessible by $\mathcal{E}$ is made explicit if the protocol uses $\mathcal{F}_{\mathsf{BB}}$. Since the protocol is only a subprotocol within $\mathcal{P}_{\mathsf{THVS}}$ which does use these ideal functionalities, this is not a problem. However, UC-security only holds in the hybrid model.

The adversary-simulator $\mathcal{S}$ controls the corrupt participants. Moreover, $\mathcal{S}$ runs a simulation of the protocol $\mathcal{P}_{\mathsf{CLMP}}$ with participants A_i, mapping corrupt $\hat{A}_i$ to corrupt A_i and simulating honest A_i as well as the adversary $\mathcal{A}$ that attacks the protocol. Lastly, $\mathcal{S}$ controls the random oracle, which means that the simulated parties A_i can create valid proofs for false claims.

$\mathcal{S}$ behaves as follows:

- $\mathcal{S}$ forwards all messages between $\mathcal{E}$ and $\mathcal{A}$.
- The simulated participants A_i hold the same ciphertexts c_1 and c_2.
- If $\mathcal{S}$ controls t or more participants $\hat{A}_i$, then $\mathcal{S}$ is able to decrypt the ciphertexts and obtain plaintexts m_1, m_2. Otherwise, he chooses plaintexts m_1 and m_2 at random.
- When the participants A_i run the Lift subprocedure, they raise ℓ_{i-1} to a random secret power and generate a non-interactive proof that this exponentiation conforms to their share verification key (v, v_i) using the corrupt random oracle.
- When the participants A_i decrypt A, they produce random decryption shares consistent with 0 (i.e.: produces 0 upon combination) if $m_1 \geq m_2$ and rn otherwise, where r is a random value from $\mathbb{Z}_n$. Since $\mathcal{S}$ controls the random oracle, the proofs of "correct" decryption are easy to falsify.

The real protocol adversary $\mathcal{A}$ has no way of discovering that he is being simulated by $\mathcal{S}$. The only messages that might indicate that he is being simulated, are the ciphertexts c_1, c_2 and intermediate encryptions having to do with lifting that do not conform to the result coming back from $\mathcal{F}_{\mathsf{MP}}$. However, if $\mathcal{A}$ can infer this from the ciphertexts, then he breaks the cryptosystem's semantic security. Similarly, if he can glean from the transcripts of the zero-knowledge proofs that the intermediate values do not conform to the result returned from $\mathcal{F}_{\mathsf{MP}}$, then he breaks the computational zero-knowledge property. Since $\mathcal{A}$ trusts the random oracle, he has no reason to assume the zero-knowledge proofs are incorrect.

Similarly, the environment $\mathcal{E}$ has no way of discovering that there is an adversary-simulator $\mathcal{S}$ between himself and the protocol-adversary $\mathcal{A}$, who also has no idea that such an $\mathcal{S}$ exists. Moreover, since $\mathcal{E}$ does not have access to the participants' decryption shares, he cannot discover that the ciphertexts and intermediate ciphertexts do not conform to the result of the protocol without breaking the cryptosystem's semantic security. Moreover, this result is guaranteed to match the parties' inputs as $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{MP}}$.

Universal verifiability. Every step in the protocol has an associated message on the bulletin board. Except for lifting and decryption, the verifier can check the new values by calculating

them himself. Lifting is a trusted ideal functionality. As for decryption, every step in this subprotocol is accompanied by efficiently verifiable zero-knowledge proofs which ensures the validity of that one step. □

B.3 Security and Verifiability of the Self-Tallying Vote

Proof of Theorem 4.
Correctness. Correctness follows from construction. Let $\mathsf{E}(m, r)$ denote the encryption using the system public key of the message m using randomizer r. Let $\oplus$ denote the operation on ciphertexts that homomorphically corresponds to addition. The aggregation of the encrypted votes is given by:

$$\mathsf{E}(v_1, r_1) \oplus \ldots \oplus \mathsf{E}(v_n, r_n) = \mathsf{E}\left(\sum_{i=1}^{n} v_i, \prod_{i=1}^{n} r^{x_i}\right) = \mathsf{E}\left(t, r^{\sum_{i=1}^{n} x_i}\right) = \mathsf{E}\left(t, r^0\right) = \mathsf{E}(t, 1)$$

UC-security. We demonstrate that for every adversary $\mathcal{A}$ who attacks the protocol, there exists an adversary-simulator $\mathcal{S}$ who attacks the ideal functionality with equal success, such that no computationally bounded external environment $\mathcal{E}$ can tell whether he is interacting with an execution of the protocol and a real adversary $\mathcal{A}$ attacking it, or with an invocation of the ideal functionality and an adversary-simulator $\mathcal{S}$ that attacks it.

The adversary-simulator runs with dummy voters $\hat{V}_i$ and dummy authorities $\hat{A}_j$. The dummy voters $\hat{V}_1, \ldots, \hat{V}_{n-1}$ receive their input from $\mathcal{E}$. The corrupt voters and authorities are controlled by $\mathcal{S}$ and the honest voters and authorities relay their inputs to $\mathcal{F}_{\mathsf{STVS}}$. At the end of the experiment, $\mathcal{E}$ reads the outputs of the dummy voters. Throughout the experiment, $\mathcal{E}$ can query the adversary. After the experiment, $\mathcal{E}$ is allowed a polynomial-time computation before he must decide whether he was interacting with $\mathcal{A}$ or $\mathcal{S}$.

The adversary-simulator $\mathcal{S}$ simulates an execution of the protocol between simulated authorities A_j and simulated voters V_i, along with the (simulated) real protocol adversary $\mathcal{A}$. The set of corrupt simulated voters and authorities matches the set of corrupt dummy voters and authorities. The adversary-simulator $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and $\mathcal{E}$.
- $\mathcal{S}$ starts by simulating the invocation of $\mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{SKG}}$ and simulates $\mathcal{F}_{\mathsf{BB}}$ throughout. As a result, $\mathcal{S}$ knows all participants' private keys as well as the system private key.
- $\mathcal{S}$ allows the authorities A_j to generate secret exponents x_i for all voters V_i. $\mathcal{S}$ is able to decrypt these messages and obtain the x_i.
- If one of the corrupt voters V_i casts their vote, $\mathcal{S}$ checks the proofs and decrypts the vote v_i and instructs V_i to send $(\textbf{vote}, sid, v_i, \hat{V}_i)$ to $\mathcal{F}_{\mathsf{VS}}$. $\mathcal{S}$ follows up with a $(\textbf{no-block}, sid, \hat{V}_i)$ message.
- On receiving $(\textbf{vote}, sid, \hat{V}_i)$ from $\mathcal{F}_{\mathsf{VS}}$ pertaining to an honest voter $\hat{V}_i$, $\mathcal{S}$ instructs V_i to cast a random vote. In order to generate sound proofs, $\mathcal{S}$ simulates the random oracle $\mathcal{F}_{\mathsf{RO}}$ to generate suitable outputs.
- When all the corrupt voters V_i have cast their votes, $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{VS}}$ and receives the tally t. $\mathcal{S}$ determines the current tally t' in the simulated protocol and instructs the control voter V_n to cast a vote of the form $\mathsf{E}(t - t')$. Again, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{RO}}$ so as to generate sound proofs.

The only ways in which $\mathcal{A}$ can discover that he is being simulated by $\mathcal{S}$ is by discovering that the encrypted votes (from the honest voters as well as from the control voter) are not valid votes. In order to come to this conclusion, $\mathcal{A}$ would have to break the semantic security of the cryptosystem or else discover that the challenge values in the zero-knowledge proofs are not random. But as $\mathcal{A}$ has bounded computational power and trusts $\mathcal{F}_{\mathsf{RO}}$, this cannot happen.

The environment $\mathcal{E}$ gets to communicate with a real process adversary $\mathcal{A}$ who has no idea he is being simulated. The tally on the tapes of the voters $\hat{V}_i$ corresponds to the inputs given by $\mathcal{E}$ to the voters at the start of the experiment or by $\mathcal{A}$ who has corrupted them. $\mathcal{E}$, like $\mathcal{A}$, cannot break the semantic security of the cryptosystem nor does he have any cause to mistrust the random oracle's output.

Universal verifiability. Except for $\mathcal{F}_{\mathsf{PGK}}$, $\mathcal{F}_{\mathsf{SKG}}$ and $\mathcal{F}_{\mathsf{RO}}$, which are trusted functionalities, all steps are either reproducible by the verifier himself or else accompanied by a zero-knowledge proof. □

B.4 Security and Verifiability of the Authenticated Voting Credential System

Proof of Theorem 5.
Correctness. Correctness follows trivially from construction. All voters V_i cast their vote v_i. The tally is given by $t = \sum_i v_i$.

UC-security. We show that the protocol is UC-secure by showing that for every adversary $\mathcal{A}$ that attacks the protocol $\mathcal{P}_{\mathsf{AVS}}$, there is an adversary-simulator $\mathcal{S}$ that attacks the ideal functionality $\mathcal{F}_{\mathsf{VS}}$ with equal success and such that no computationally bounded external environment $\mathcal{E}$ can tell whether he is interacting with $\mathcal{A}$ and $\mathcal{P}_{\mathsf{AVS}}$ or with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{VS}}$.

The adversary-simulator $\mathcal{S}$ runs with dummy voters $\hat{V}_i$, who receive their input from $\mathcal{E}$ and with dummy authorities $\hat{A}_j$. After the experiment is finished, $\mathcal{E}$ looks at the tapes of the voters V_i and authorities A_j, computes for a polynomial duration and eventually outputs a guess as to whether he is interacting with $\mathcal{A}$ or $\mathcal{S}$. Throughout the experiment, $\mathcal{E}$ can communicate with the adversary.

The adversary-simulator simulates an execution of the protocol between simulated authorities A_j, simulated voters V_i and a simulated real-protocol adversary $\mathcal{A}$. The parties A_j or V_i that were corrupted by $\mathcal{A}$ correspond to the parties $\hat{A}_j$ or $\hat{V}_i$ that were corrupted by $\mathcal{S}$. The adversary-simulator proceeds as follows:

- $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and $\mathcal{E}$.
- $\mathcal{S}$ starts by simulating the invocation of $\mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{SKG}}$ and simulates $\mathcal{F}_{\mathsf{BB}}$ throughout. As a result, $\mathcal{S}$ knows all private keys and private key shares.
- $\mathcal{S}$ instructs all honest voters V_i to withdraw a credential.
- $\mathcal{S}$ instructs all honest authorities A_i to approve all credential withdrawal requests from eligible voters, including from the corrupted voters (provided that they made a valid credential withdrawal request).
- $\mathcal{S}$ instructs the authorities to make the $\mathcal{F}_{\mathsf{BB}}$ unreadable by sending (**close**, *sid*) to the bulletin board.
- If any corrupt V_i casts a vote (i.e.: posts a $(v_{i'}, a, Z)$-message to $\mathcal{F}_{\mathsf{BB}}$), then $\mathcal{S}$ determines who cast the vote and he instructs the matching $\hat{V}_i$ to cast the same vote in $\mathcal{F}_{\mathsf{VS}}$. $\mathcal{S}$ allows this vote to pass.
- If $\mathcal{S}$ receives (**vote**, $\hat{V}_i$, *sid*) from $\mathcal{F}_{\mathsf{VS}}$ pertaining to an honest voter $\hat{V}_i$, $\mathcal{S}$ allows it to pass.

– When all $\hat{V}_i$ have cast their votes, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{VS}}$ to compute the tally t. From this tally, $\mathcal{S}$ is able to infer the votes that were cast by all the honest voters, without mapping any one vote to the voter that cast it. $\mathcal{S}$ chooses a random mapping of honest votes $v_{i'}$ to honest voters V_i. For each of these honest voters V_i, the bulletin board $\mathcal{F}_{\mathsf{BB}}$ stores a post of the form $(v_{i'}, a, Z)$, where $v_{i'}$ is the vote, a is the preimage of V_i's credential and Z is a non-interactive proof transcript attesting to knowledge of a certificate and involving a and $v_{i'}$. The timestamps of these posts are chosen at random. (The timestamps of the posts from the corrupt parties are not modified.)
– At this point, $\mathcal{S}$ instructs the honest authorities A_i to open $\mathcal{F}_{\mathsf{BB}}$ for reading by sending a (**read**, *sid*) message. All participants including $\mathcal{A}$ can read the bulletin board and calculate the tally.

In order to discover that he is being simulated by the adversary-simulator $\mathcal{S}$, the real protocol adversary $\mathcal{A}$ must somehow discover a discrepancy. However, since the adversary cannot observe the behavior of the honest parties except by what they post to the bulletin board, he has nothing to compare the timings of their posts to and hence no indication that suggests they are not authentic. Moreover, the timings of the posts of the corrupted parties conforms to what the adversary instructed them to do.

Since access to the bulletin board is anonymous and since the credentials are unlinkable to the transcript of the protocol that created them, the adversary has no way to determine which of the honest votes was cast by which honest voter.

The external environment $\mathcal{E}$ communicates with $\mathcal{A}$ throughout the protocol. However, $\mathcal{A}$ has no idea he is being simulated. Moreover, there is no indication that there is a $\mathcal{S}$ relaying messages between $\mathcal{E}$ and a simulated $\mathcal{A}$. Since the credentials are unlinkable to the voters, $\mathcal{E}$ cannot compare the mapping between cast votes and the voters of the protocol with the mapping implicit in his input to the voters at the start of the experiment.

Universal verifiability. Vote casting is verifiable as it is accompanied by a zero-knowledge proof which authenticates the credential as well as the vote. It is trivial to check that credentials are not reused as the value a must remain fixed. The verifier can check that only eligible voters are assigned credentials via the bulletin board. This cannot happen as long as at least t authorities are honest. In this sense, verifiability of correctness is still universal (i.e.: *anyone* can verify), but correctness itself is reduced to an assumption of *trust*, down from no assumption at all. □

C Zero-Knowledge Proofs

C.1 Guillou and Quisquater Zero-Knowledge Proof of RSA Signature

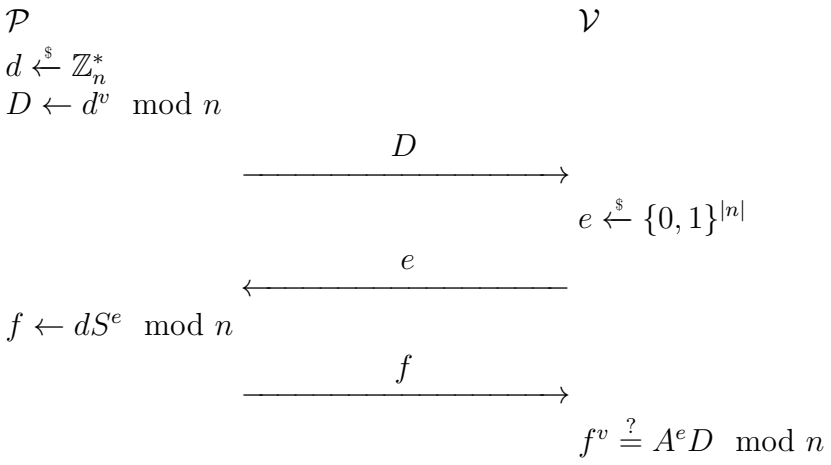
Theorem 6. *Protocol 11 is an interactive proof of knowledge of S satisfying $S^v = A \mod n$ which satisfies the completeness, special soundness and special honest verifier zero-knowledge properties.*

Proof of Theorem 6.
Completeness. Completeness follows from construction.

$$f^v = (dS^e)^v = d^v(S^v)^e = DA^e \mod n \ .$$

Interactive Proof $\mathcal{P}_{\mathsf{RSA}}$

Public knowledge: n, v, A.
Private knowledge for $\mathcal{P}$: S s.t. $S^v = A \mod n$.



Protocol 11: interactive proof of knowledge of RSA signature.

Special soundness. Given two transcripts T_1 and T_2 proving the same claim and having identical first two messages but different third message, it is easy to compute the signature S. We have two points $(T_1.e, T_1.f)$ and $(T_2.e, T_2.f)$ which define a linear function f. Using Lagrange interpolation, we evaluate f in $e = 0$ first to determine d and then in $e = 1$ to determine dS and subsequently remove the factor d.

Special honest-verifier zero-knowledge. There exists an efficient simulator $\mathcal{S}$ who, given a claim A and challenge e, can generate a sound proof proving knowledge of S such that $S^v = A \mod n$. This simulated proof is indistinguishable by any adversary.

$\mathcal{S}$ proceeds as follows. Set $f \xleftarrow{\$} \mathbb{Z}_n^*$ and $D \leftarrow f^v A^{-e} \mod n$. The resulting proof transcript is $T = (A, D, e, f)$. No adversary can distinguish simulated transcripts from authentic ones as their distributions are identical. □