# Signatures of Reputation: Towards Trust Without Identity

John Bethencourt* UC Berkeley

Elaine Shi PARC

Dawn Song* UC Berkeley

**Abstract**

Reputation systems have become an increasingly important tool for highlighting quality information and filtering spam within online forums. However, the dependence of a user's reputation on their history of activities seems to preclude any possibility of anonymity. We show that useful reputation information can, in fact, coexist with strong privacy guarantees. We introduce and formalize a novel cryptographic primitive we call *signatures of reputation* which supports monotonic measures of reputation in a completely anonymous setting. In our system, a user can express trust in others by voting for them, collect votes to build up her own reputation, and attach a proof of her reputation to any data she publishes, all while maintaining the unlinkability of her actions.

## 1 Introduction

In various forms, reputation has become a ubiquitous tool for improving the quality of online discussions. For example, a user may mark a product review on Amazon or a business review on Yelp as "useful", and these ratings allow others to more easily identify the best reviews and reviewers. Most web message boards also include a means of providing feedback to help highlight quality content, such as Slashdot's "karma" system and the similar systems employed in many boards running phpBB (the most common web message board software).

Unfortunately, in all such systems, a user is linked by their pseudonym to a history of their messages or other activities. In many online communities (e.g., a support group for victims of abuse), users may hope that the use of a pseudonym allows them to remain anonymous. However, recent work has shown that very little prior information about an individual is necessary to match them to their pseudonym [NS08, BDK07, Arr06]. Building a truly private forum requires abandoning the notion of persistent identities.

We raise the question of whether it is possible to gain all the utility of existing reputation systems while maintaining the unlinkability and anonymity of individual user actions, thus avoiding the histories of activity which threaten privacy. Such a system would enable a number of intriguing applications. For example, we might imagine an anonymous message board in which every post stands alone – not even associated with a pseudonym. Users would rate posts based on whether they are helpful or accurate, collect

*Dawn Song and John Bethencourt are partially supported by the National Science Foundation under Grants No. 0716230, 0448452, and CCF-0424422 and by the Office of Naval Research under MURI Grant No. N000140911081. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Office of Naval Research.

reputation from other users' ratings, and annotate or sign new posts with the collected reputation. Other users could then judge new posts based on the author's reputation while remaining unable to identify the earlier posts from which it was derived. Such a forum would allow effective filtering of spam and highlighting of quality information while providing an unprecedented level of user privacy.

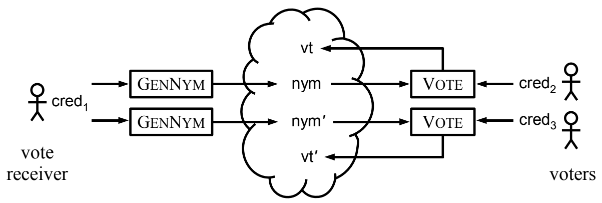
*Our approach.* To build toward this goal, we propose *signatures of reputation* as a new cryptographic framework enabling the counter-intuitive combination of reputation and anonymity. In a conventional signature scheme, a signature is associated with a public key and convinces the verifier that the signer knows the corresponding private key. Based on the public key, a verifier could then retrieve the reputation of the signer. Through signatures of reputation, we aim to eliminate the middle step of identifying the signer: instead, verification of the signature directly reveals the signer's reputation. With such a tool, a user may apply their reputation to *any* data that they wish to publish online, without risking their privacy. By formally defining this setting, we hope to spur further research into techniques for its realization.

As a first step, we provide a construction for signatures of reputation that supports *monotonic* aggregation of reputation. That is, we assume that additional feedback cannot decrease a user's reputation. While a user's misbehavior cannot damage reputation they have already accumulated, such a system is sufficient to prevent more casual attackers who, for example, wish to post spam without taking the time to obtain reputation first. Although some existing reputation systems are monotonic (e.g., Google's PageRank algorithm), one would ultimately hope to support non-monotonic reputation as well. We leave this as a primary open problem for future work.

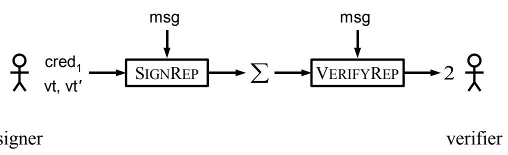
In our construction, the reputation feedback takes the form of cryptographic "votes" that users construct and send to one another, and a user's reputation is simply the number of votes they have collected from distinct users. Each user stores the votes they have collected, and to anonymously sign a message with their reputation, the user constructs a non-interactive zero-knowledge (NIZK) proof of knowledge which demonstrates possession of some number of votes. The ability of a reputation system to limit the influence of any single user is crucial in enabling applications to control abuse. To this end, our construction ensures that each user can cast at most one valid vote for another user (or up to $k$ for any fixed $k \geq 1$). Enforcing this property is a major technical problem due to the tension with the desired unlinkability properties; we solve it through a technique for proving the distinctness of a list of values within a NIZK.

One unique (to our knowledge) feature of our constructions is the use of *nested NIZKs*, that is, NIZKs which prove knowledge of other NIZKs and demonstrate that they satisfy the verification equations. This situation arises because a vote contains a NIZK proof that the voter has a valid credential. When a signer later uses the vote, they include this NIZK within a further NIZK to demonstrate the validity of the votes while maintaining their anonymity. Other interesting technical features are the use of key-private, homomorphic encryption to allow users to receive votes while remaining anonymous and a space-saving technique employing Merkle hash trees, which reduces the size of a signature demonstrating reputation $c$ from $O(c)$ to $O(\log c)$.

*Related work.* While we are not aware of any work directly comparable to our proposed signatures of reputation, others have explored the conflict between reputation and unlinkability [Ste08, PS08, Ste06]. E-cash schemes [ASM08, CLM07, AWSM07, CHL05] also attempt to maintain the unlinkability of individual user interactions, and in several cases [BCE$^+$07, CHL06, ACBM08] they have been applied for reputation or incentive purposes. The work of Androulaki et al. [ACBM08] is particularly close to ours in its aims. However, this and all other e-cash based approaches are incapable of supporting



(a) A user anonymously posts two one-time pseudonyms, each of which receives a vote.



(b) The user may sign a message while proving they have votes from two distinct users.

Figure 1: One-time pseudonyms, votes, and signatures of reputation.

the type of abuse resistance provided by our scheme because they allow a single user to give multiple coins to another, inflating their reputation. In our scheme, it is possible to prove that a collection of votes came from *distinct* users. This ability to prove distinctness while maintaining the mutual anonymity of both voters and vote receivers is the key technical achievement of our construction.

Schemes for anonymous credentials [CKS09, BCKL08, BCC$^+$09, CHK$^+$06] employ some similar techniques to those of our constructions and may also be considered an effort toward the goal of "trust without identity". There are two key distinctions, however. First, anonymous credentials are concerned with the setting of access control based on trust derived from explicit authorities, whereas this work aims to support trust derived from a very different source: the aggregate opinions of other users. Second, like e-cash based approaches, existing anonymous credential schemes lack a mechanism for proving that votes or credentials come from distinct users while simultaneously hiding the identities of those users.

Finally, our setting superficially resembles that of e-voting protocols [Gro05, Gro04, DGS03], in that it allows the casting of votes while maintaining certain privacy properties. However, schemes for e-voting are designed for an election scenario in which the candidates have no need to receive votes and prove possession of votes anonymously, among other differences, and cannot be used to achieve the properties we require.

*Organization.* The remainder of this paper is organized as follows. In Section 2, we elaborate on the context for the vote counting scenario and list the algorithms that would constitute a scheme for signatures of reputation. In Section 3, we define the privacy and security properties we desire from such a scheme. In Section 4, we discuss various cryptographic tools and modules we will need in the full constructions, which are given in Section 5. Finally, in Section 6, we describe modifications to the scheme to reduce the size of the signatures before concluding with a list of open problems in Section 7.

## 2 Defining Signatures of Reputation

Figure 1 illustrates the flow of information in our formulation of signatures of reputation. We refer to each user in the system as a *vote receiver*, *voter*, *signer*, or *verifier* depending on their role in the specific algorithm being discussed. To ensure *receiver anonymity*, a vote receiver invokes the GenNym algorithm to compute a "one-time pseudonym" called a nym, which they attach to some content that they publish and wish to receive credit for. A voter can then use the Vote algorithm on a nym to produce a vote which hides their identity, even from the recipient (referred to as *voter anonymity*). The voter posts the vote online where the recipient can later retrieve it. After collecting some votes, a signer runs the SignRep algorithm to construct a signature of reputation on a message. The signature contains a proof of knowledge of distinct votes, that is, votes received by the signer from distinct voters. The zero-knowledge nature of the proofs protects the identity of the signer (*signer anonymity*). We also ensure that a malicious signer cannot inflate its reputation (*reputation soundness*).

To participate in the system, each user must register with a *registration authority* (RA) which generates the user's private credentials, just as the key generating server does within IBE schemes. Although we trust the RA for both privacy and reputation soundness, it need only be trusted when registering users and may thereafter go offline. As with typical IBE schemes [BF01], it is also possible to reduce the trust necessary in the RA by distributing it amongst multiple parties [GJKR99]. Devising a scheme which maintains privacy in the presence of a malicious RA is an interesting problem for future work. On the other hand, relying on the honesty of the RA for reputation soundness seems inevitable, since a malicious RA could always register additional phony users (i.e., Sybil identities) to arbitrarily create votes and inflate reputations.

*Additional considerations.* Astute readers have likely noticed that the reputation value is itself identifying if each user has received a unique number of votes. Clearly, there is an inherent tradeoff between the precision of a measure of reputation and the anonymity of a user with any specific value, as pointed out by Steinbrecher [PS08]. The solution is to use a sufficiently coarse-grained reputation; fortunately, this is easily accomplished in the case of our construction. When producing a signature, a user may prove any desired *lower bound* on their reputation instead of revealing the actual value. In this way, our construction allow users to implement their own policies for the precision of their reputations. For example, one policy would be to always round down to a power of two.

Another issue to consider is the connection between a piece of content a user has posted and the attached $\mathsf{nym}$. Two abuses are possible: reposting the $\mathsf{nym}$ of another user with a piece of undesirable content in order to malign the user's reputation and reposting the desirable content of another user with one's own $\mathsf{nym}$ in order to steal the credit. The former problem can be easily prevented by including a signature within the $\mathsf{nym}$ linking it to a specific message. However, this is only useful in a reputation system supporting negative feedback. Since our constructions only support monotonic reputation, we do not include this feature. On the other hand, there is, in general, no simple way of preventing the latter problem. Note that the problem of assigning credit does not stem from the anonymity that we provide; it equally affects non-privacy-preserving reputation systems. In the case of audio or video content, one way to address this would be to use digital watermarking techniques [CMB02] to embed the $\mathsf{nym}$ throughout the content.

*Algorithms.* We now specify the parameters to and output of each of the algorithms in a scheme for signatures of reputation. In our notation, we use the equals sign to denote evaluation of VerifyRep, which must be deterministic. We use arrows to denote evaluation of each of the other algorithms to emphasize that they may be randomized.

$\textsc{Setup}(1^\lambda) \to (\mathsf{params}, \mathsf{authkey})$: The Setup algorithm is run once on security parameter $1^\lambda$ to establish the public parameters of the system $\mathsf{params}$ and a key $\mathsf{authkey}$ for the registration authority.

$\textsc{GenCred}(\mathsf{params}, \mathsf{authkey}) \to \mathsf{cred}$: To register a user, the registration authority runs GenCred and returns the credential $\mathsf{cred}$.

$\textsc{GenNym}(\mathsf{params}, \mathsf{cred}) \to \mathsf{nym}$: The GenNym algorithm produces a one-time pseudonym $\mathsf{nym}$ from a user's credential.

$\textsc{Vote}(\mathsf{params}, \mathsf{cred}, \mathsf{nym}) \to \mathsf{vt}$ or $\perp$: Given the credentials $\mathsf{cred}$ of some user and a one-time pseudonym $\mathsf{nym}$, Vote outputs a vote from that user for the owner of $\mathsf{nym}$, or $\perp$ in case of failure (e.g., if $\mathsf{nym}$ is invalid).

$\textsc{SignRep}(\mathsf{params}, \mathsf{cred}, V, \mathsf{msg}) \to \Sigma$ or $\perp$: Given the credentials $\mathsf{cred}$ of some user, the $\textsc{SignRep}$ algorithm constructs a signature of reputation $\Sigma$ on a message $\mathsf{msg}$ using a collection of $c$ votes $V = \{\mathsf{vt}_1, \mathsf{vt}_2, \ldots, \mathsf{vt}_c\}$ for that user. The signature corresponds to a reputation $c' \leq c$, where $c'$ is the number of distinct users who generated votes in $V$. The $\textsc{SignRep}$ algorithm outputs $\perp$ on failure, specifically, when $V$ contains an invalid vote or one whose recipient is not the owner of $\mathsf{cred}$.

$\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) = c$ or $\perp$: The $\textsc{VerifyRep}$ algorithm checks a purported signature of reputation on $\mathsf{msg}$ and outputs the corresponding reputation $c$, or $\perp$ if the signature is invalid.

The most basic property required of an implementation of these algorithms is correctness: they should produce the expected results when executed normally. We define this property as follows.

**Definition 1** (Correctness for signatures of reputation)**.** *Let $n$ be a positive integer and $S \subseteq \{1, \ldots, n\}$. Set* $(\mathsf{params}, \mathsf{authkey}) \leftarrow \textsc{Setup}(1^\lambda)$ *and* $\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})$, *for $i \in \{1, \ldots, n\}$. Set* $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_1)$, $V = \{ \mathsf{vt} \mid \mathsf{vt} \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_i, \mathsf{nym}), i \in S \}$, *and* $\Sigma \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_1, V, \mathsf{msg})$, *for some message* $\mathsf{msg}$. *If the preceding implies, with probability one, that* $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) = |S|$, *then we say that the scheme is* correct.

We now go on to define the other properties we desire of a scheme for signatures of reputation.

## 3 Privacy and Security Properties

In the preceding section, we gave intuitive descriptions of the four intended privacy and security properties: receiver anonymity, voter anonymity, signer anonymity, and reputation soundness. Defining these properties rigorously requires considerably more subtlety.

In particular, to ensure our definitions are well-formed, we require the existence of several additional algorithms. Given a special "opening key" produced by an alternate version of $\textsc{Setup}$, the two opening algorithms (which must be deterministic) reveal the users associated with pseudonyms and votes, thereby establishing a ground truth to which we can refer when defining the privacy and security properties. These are directly analogous to the opening algorithm of group signature schemes. However, while opening is considered an intended feature of group signatures, in our case the opening algorithms exist solely for the definitions and would not be used in practice. For brevity, each of the opening algorithms is given $\mathsf{params}$ and a list of user credentials $\mathsf{cred}_1, \ldots, \mathsf{cred}_n$ as implicit arguments.

$\textsc{Setup}'(1^\lambda) \to (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey})$: $\textsc{Setup}'$ produces values $\mathsf{params}, \mathsf{authkey}$ according to the same distribution as $\textsc{Setup}$, but also outputs an opening key $\mathsf{openkey}$.

$\textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym}) = i$: Output the index of the credential that produced $\mathsf{nym}$.

$\textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}) = (i, \mathsf{nym})$: Output the index of the credential and the $\mathsf{nym}$ from which the vote $\mathsf{vt}$ was constructed.

Any scheme for signatures of reputation must provide an implementation of the above algorithms that is *correct* according to the following definition.

**Definition 2** (Correctness of opening algorithms)**.** *Let $n$ be a positive integer. Set* $(\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda)$ *and* $\mathsf{cred}_i = \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})$ *for* $i \in \{1, \dots, n\}$*. Then given any* $i \in \{1, \dots, n\}$ *and* $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_i)$*, we require that* $\textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym}) = i$*. In addition, given any* $i, j \in \{1, \dots, n\}$*,* $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_i)$ *and* $\mathsf{vt} \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_j, \mathsf{nym})$*, we require that* $\textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}) = (j, \mathsf{nym})$*. If both of these properties hold, we say that the scheme has* correct opening algorithms*.*

For later notational convenience, given a set of votes $V$, we also define the functions $\textsc{OpenVoters}(\mathsf{openkey}, V) = \{\ i \mid (i, \mathsf{nym}) = \textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}), \mathsf{vt} \in V\ \}$ and $\textsc{OpenReceivers}(\mathsf{openkey}, V) = \{\ \mathsf{nym} \mid (i, \mathsf{nym}) = \textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}), \mathsf{vt} \in V\ \}$.

Before describing the games defining each of the privacy and security properties, one more preliminary matter must be discussed. In the games we define, the adversary may make queries to an oracle $\mathcal{O}$. The oracle is given access to a list of user credentials $\mathsf{cred}_1, \dots, \mathsf{cred}_n$ and responds to the following four types of queries. On input ("corrupt", $i$), $\mathcal{O}$ returns $\mathsf{cred}_i$. On input ("nym", $i$), $\mathcal{O}$ returns $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, cred_i)$. On input ("vote", $i, \mathsf{nym}$), $\mathcal{O}$ returns $\mathsf{vt} \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_i, \mathsf{nym})$. On input ("signrep", $i, V, \mathsf{msg}$), $\mathcal{O}$ returns $\Sigma \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_i, V, \mathsf{msg})$. In each case, $\mathcal{O}$ also logs the tuple on which it was queried and its response by adding them to a set $L$. We will refer to the logged queries and responses in order to state the winning conditions for each game.

*Receiver anonymity.* The receiver anonymity property captures the notion that a one-time pseudonym generated by the GenNym algorithm should reveal nothing about its owner, unless the adversary has seen that user's credential or made a SignRep query which trivially reveals the owner. This property may be defined by the following game, where $\mathsf{st}$ denotes the internal state of the adversary.

$$\mathbf{Pr}_{\mathcal{A}}^{\mathsf{RANON}}(\lambda) = \Pr\left[\begin{array}{c|c} b = b' \ \wedge & (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ \textsc{Legal}(i_0^*, i_1^*, \mathsf{openkey}, L) & [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}; \\ & (i_0^*, i_1^*, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{params});\ b \xleftarrow{R} \{0, 1\}; \\ & \mathsf{nym}^* \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_{i_b^*});\ b' \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{st}, \mathsf{nym}^*) \end{array}\right]$$

To prevent $\mathcal{A}$ from winning this game through normal usage of the scheme, we make the following requirements on its queries and challenge $(i_0^*, i_1^*)$, which we abbreviate as "Legal". First, ("corrupt", $i_0^*$) $\notin L$ and ("corrupt", $i_1^*$) $\notin L$. In other words, if an adversary must compromise a user's private credentials to detect their pseudonyms, we will not consider that a failure of receiver anonymity.[1] Second, for all ("signrep", $i, V, \mathsf{msg}$) $\in L$, if $i \in \{i_0^*, i_1^*\}$, we require that $\mathsf{nym}^* \notin \textsc{OpenReceivers}(\mathsf{openkey}, V)$. An adversary that violates this property is one that simply returns the challenge $\mathsf{nym}^*$ (after voting for it) in a SignRep query. The reply to such a query immediately reveals $b$, as expected by the semantics of the scheme ($i = i_b^*$ iff the reply is not $\perp$). Given these rules, we define receiver anonymity as follows.

**Definition 3.** *A scheme for signatures of reputation is* receiver anonymous *if, for all PPT adversaries $\mathcal{A}$, $|\mathbf{Pr}_{\mathcal{A}}^{\mathsf{RANON}}(\lambda) - \frac{1}{2}|$ is a negligible function of $\lambda$.*

[1] One might try to extend this definition to incorporate a forward security property ensuring pseudonyms generated before a user's credentials are compromised remain unlinkable (that is, by updating the credentials after generating each pseudonym). However, this is futile: if the updated credential can still use votes cast for old pseudonyms, then Vote and SignRep can be used to detect the old pseudonyms.

*Voter anonymity.* We wish to define the voter anonymity property to encompass the strongest form of unlinkability compatible with the general semantics of the scheme, as we did in the case of receiver anonymity. Doing so is more subtle in this case, however, due to the necessity of detecting duplicate votes. Because we require a SignRep algorithm to demonstrate the number of votes from *distinct* users, such an algorithm can be used by a vote receiver to determine whether two votes cast for any of their pseudonyms were produced by the same voter (duplicates). That is, the receiver can try to use the two votes to produce a signature and then check the reputation of the result with VerifyRep.

In defining voter anonymity, we allow precisely this type of duplicate detection, but nothing more. While initially this may seem like an "exception" to the unlinkability of votes, in actuality, it is not only inevitable,[2] but also unlikely to be a practical concern. Although a vote receiver must be able to detect duplicate votes, we can still avoid the voting histories we originally set out to eliminate. In particular, our definition ensures that in the following cases it is not possible to determine whether two votes were cast by the same user (i.e., to link the votes):

1. A user cannot link a vote for one of their pseudonyms with a vote for a pseudonym of another user, nor can they link two votes for distinct pseudonyms of another user (or two different users).

2. A *colluding group* of users cannot link votes between their pseudonyms, provided the pseudonyms correspond to different credentials. Furthermore, they are not able to link the *numbers* of duplicates they have observed. For example, if a user determines that they have received two votes from one user and three votes from another, they will have no way of matching these totals up with those of another colluding user.

The game below captures these properties.

$$\mathbf{Pr}_{\mathcal{A}}^{\mathsf{VANON}}(\lambda) = \Pr\left[\begin{array}{c|c} b = b' \ \wedge & (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ \textsc{Legal}(j_0^*, j_1^*, \mathsf{nym}^*, & [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}; \\ \mathsf{openkey}, L) & (j_0^*, j_1^*, \mathsf{nym}^*, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{params});\ b \xleftarrow{R} \{0,1\}; \\ & \mathsf{vt}^* \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_{j_b^*}, \mathsf{nym}^*);\ b' \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{st}, \mathsf{vt}^*) \end{array}\right]$$

In this case, we define Legal to check the following. Let $i^* = \textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym}^*)$. First, if the adversary has made a query ("signrep", $i^*, V, \mathsf{msg}) \in L$ where $\mathsf{vt}^* \in V$, we require that $\{j_0^*, j_1^*\} \subseteq \textsc{OpenVoters}(\mathsf{openkey}, V)$. In other words, the coin $b$ should not determine the number of distinct voters in a "signrep" query involving $\mathsf{vt}^*$. Second, if ("corrupt", $i^*) \in L$, we require that there not exist a ("vote", $j, \mathsf{nym}) \in L$ such that $j \in \{j_0^*, j_1^*\}$ and $i^* = \textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym})$. That is, if the adversary controls the receiver $i^*$ of the challenge vote, then they may not request another vote from $j_0^*$ or $j_1^*$, since its status as a duplicate or lack thereof would reveal $b$.

**Definition 4.** *A scheme for signatures of reputation is* voter anonymous *if, for all PPT adversaries* $\mathcal{A}$, $|\mathbf{Pr}_{\mathcal{A}}^{\mathsf{VANON}}(\lambda) - \frac{1}{2}|$ *is a negligible function of* $\lambda$.

*Signer anonymity.* The signer anonymity property requires that a signature of reputation reveal nothing about the signer beyond their reputation. In this case, we allow the

[2]Allowing proofs of vote distinctness while eliminating the ability to identify duplicates could only be possible if the notion of discrete votes is abandoned. This approach would require *all* votes in the system to be aggregated into a indivisible block before they can be used to produce signatures, a vastly impractical solution.

adversary access to all user's credentials. As a result, they have no need for the oracle $\mathcal{O}$, as the adversary could answer the queries itself.

$$\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SANON}}(\lambda) = \Pr\left[\begin{array}{c|c} b = b' \ \wedge & (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ \textsc{Legal}(\mathsf{msg}, \Sigma_0^*, \Sigma_1^*) & [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}; \\ & (i_0^*, i_1^*, V_0^*, V_1^*, \mathsf{msg}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{params}, [\mathsf{cred}_i]);\ b \xleftarrow{R} \{0,1\}; \\ & \Sigma_b^* \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_{i_b^*}, V_b^*, \mathsf{msg});\ b' \leftarrow \mathcal{A}(\mathsf{st}, \Sigma_b^*) \end{array}\right]$$

Here, $\textsc{Legal}$ requires only that $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma_0^*) = \textsc{VerifyRep}(\ \mathsf{params}, \mathsf{msg}, \Sigma_1^*)$. That is, the value of $b$ should affect neither the reputation values of the resulting signatures nor their validity.

**Definition 5.** *A scheme for signatures of reputation is* signer anonymous *if, for all PPT adversaries $\mathcal{A}$, $|\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SANON}}(\lambda) - \frac{1}{2}|$ is a negligible function of $\lambda$.*

*Reputation soundness.* To define the soundness of a scheme for signatures of reputation, we use a computational game in which the adversary must forge a valid signature of some reputation strictly greater than that of any signature they could have produced through legitimate use of the scheme.

$$\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SOUND}}(\lambda) = \Pr\left[\begin{array}{c|c} \textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) \neq \bot & (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ \wedge\ \textsc{Legal}(\mathsf{openkey}, L, \mathsf{msg}, \Sigma) & [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}; \\ & (\mathsf{msg}, \Sigma) \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{params}) \end{array}\right]$$

In this case, $\textsc{Legal}$ makes the requirement that $\Sigma \notin L$. More subtly, it must also ensure that the forged signature has reputation greater than what it could be if the adversary had used the scheme normally. The value of the best such legitimately obtainable reputation will depend on several things: the number of users the adversary has corrupted (since the adversary may use their credentials to produce votes), the number of votes received from honest users via oracle queries, and how those votes were distributed amongst the corrupted users. More precisely, the adversary may legitimately construct a signature of reputation equal to, at most, the sum of the *number of corrupted users* and the *greatest number of distinct honest users that voted for a single corrupt user.*

The corresponding requirement checked by $\textsc{Legal}$ may be formalized as follows. Let $C = \{\ i \mid (\text{"corrupt"}, i) \in L\ \}$ be the set of corrupted users. For each $i \in C$, define $S_i = \{\ j \mid (\text{"vote"}, j, \mathsf{nym}) \in L \wedge j \notin C \wedge i = \textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym})\ \}$. Let $\ell_1 = |C|$, and let $\ell_2 = \max_{i \in C} |S_i|$. That is, $\ell_2$ is the greatest number of distinct honest users that voted for a single corrupt user. Then we require that $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) > \ell_1 + \ell_2$ for the adversary to succeed.

**Definition 6.** *A scheme for signatures of reputation is* sound *if, for all PPT adversaries $\mathcal{A}$, $\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SOUND}}(\lambda)$ is a negligible function of $\lambda$.*

In some applications, a weaker version of soundness may suffice and may be desirable for greater efficiency. One natural way to relax the definition is to specify an additional security parameter $\varepsilon \in [0, 1)$ as a multiplicative bound on the severity of cheating we wish to prevent. That is, we require a signature of reputation $c$ to ensure that at least $(1-\varepsilon) \cdot c$ distinct votes for the signer exist. To this end, we define $\mathbf{Pr}_{\mathcal{A}}^{\varepsilon-\mathsf{SOUND}}(\lambda)$ the same way as $\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SOUND}}(\lambda)$, but using the requirement that $(1-\varepsilon) \cdot \textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) > \ell_1 + \ell_2$. This yields the following definition of $\varepsilon$-soundness.

**Definition 7.** *A scheme for signatures of reputation is* $\varepsilon$-sound *if, for all PPT adversaries $\mathcal{A}$, $\mathbf{Pr}_{\mathcal{A}}^{\varepsilon-\mathsf{SOUND}}(\lambda)$ is a negligible function of $\lambda$.*

Note that Definition 6 is the special case of the above where $\varepsilon = 0$.

## 4 Building Blocks

We now describe the technical tools from which our scheme is constructed, including several standard cryptographic primitives, two specialized modules, and our complexity assumptions. First of all, our constructions rely on a bilinear map between groups of prime order $p$, which we denote $e : \mathbb{G} \times \widehat{\mathbb{G}} \rightarrow \mathbb{G}_T$. We also assume the availability of a collision-resistant hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$.

*Non-interactive proof systems for bilinear groups.* Our scheme will also make extensive use of the recent Groth-Sahai non-interactive proof system [GS08]. Their techniques allow the construction of non-interactive witness-indistinguishable (NIWI) and non-interactive zero-knowledge (NIZK) proofs for pairing product equations, multi-scalar equations in $\mathbb{G}$ or $\widehat{\mathbb{G}}$, and quadratic equations in $\mathbb{Z}_p$. We now define the notation we will use to refer to this scheme. We write $\text{GS.S}\textsc{etup}(1^\lambda) \rightarrow (\mathsf{crs}, \mathsf{xk})$ to denote the setup algorithm, which outputs a common reference string $\mathsf{crs}$ and an extractor key $\mathsf{xk}$. We use the notation introduced by Camenisch and Stadler [CS97] of the form $\Pi = \text{NIZK}\{\ x_1, \ldots, x_k\ :\ E_1 \wedge \ldots \wedge E_\ell\ \}$ to denote the construction of a zero-knowledge proof that a set of equations $E_1, \ldots, E_\ell$ is satisfiable. Here, $x_1, \ldots, x_k$ denote the secret witness variables. The NIZK consists of a commitment to each of the $k$ witness variables, along with a constant size value for each of the $\ell$ equations. When variables other than the witnesses appear in the listed equations, those are public values which are not included in the proof. These values must be available for verification of the resulting proof, which we denote by $\text{GS.V}\textsc{erify}(\mathsf{crs}, \Pi, \langle a_1, \ldots, a_m\rangle)$. The arguments $a_1, \ldots, a_m$ are the public values; the relevant equations will be clear from context. Note that in the GS proof system, it is possible to produce a NIZK only when the equations being proved are "tractable" [Gro07]. This condition holds for all the equations throughout this paper, since none involve any elements of $\mathbb{G}_T$ except the identity.

*Selective-tag weakly CCA-secure encryption.* Next, we use a tag based encryption scheme [MRY04], which we require to be selective-tag, weakly CCA-secure. For this we may employ the scheme due to Kiltz [Kil06] based on the DLinear assumption. We denote its algorithms as follows.

$\text{CCAE}\textsc{nc}.\text{S}\textsc{etup}(1^\lambda) \rightarrow (\mathsf{pk}_{\text{cca}}, \mathsf{sk}_{\text{cca}})$: Generate a public, private key pair.

$\text{CCAE}\textsc{nc}.\text{E}\textsc{nc}(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, \mathsf{msg}, (r, s)) \rightarrow C$: Encrypt a message under the given public key and $\mathsf{tag}$ using randomness $(r, s) \in \mathbb{Z}_p^2$.

$\text{CCAE}\textsc{nc}.\text{D}\textsc{ec}(\mathsf{sk}_{\text{cca}}, \mathsf{tag}, C) \rightarrow \mathsf{msg}$: Use the private key to decrypt a ciphertext encrypted under $\mathsf{tag}$.

When we need to encrypt multiple elements $\vec{x} = (x_1, \ldots, x_k) \in \mathbb{G}^k$, we use the following shorthand: $\text{CCAE}\textsc{nc}.\text{E}\textsc{nc}(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, \vec{x}, \vec{r})$, where $\vec{r} \in \mathbb{Z}_p^{2k}$.

*Weakly EF-CMA secure signatures and strong one-time signatures.* We will also use the SDH-based signature scheme due to Boneh and Boyen [BB08], which we denote BBS$\textsc{ig}$. Let $g \xleftarrow{R} \mathbb{G}, s \xleftarrow{R} \mathbb{Z}_p$. In BBS$\textsc{ig}$, the signing key is $\mathsf{sk}_{\text{bb}} = (g, s)$, the verification key is $\mathsf{vk}_{\text{bb}} = (g, g^s)$, and a message $\mathsf{msg} \in \mathbb{Z}_p$ is signed by computing $\text{BBS}\textsc{ig}.\text{S}\textsc{ign}(\mathsf{sk}_{\text{bb}}, \mathsf{msg}) = g^{\frac{1}{s+\mathsf{msg}}}$. This scheme is existentially unforgeable under weak chosen-message attack (weak EF-CMA security). In a weak chosen message attack, the adversary commits to the query messages at the beginning of the security game. The scheme is also a strong one-time signature scheme; in our construction, we use subscripts such as $\sigma_{\text{bb}}$ and $\sigma_{\text{ots}}$ to distinguish the cases where we use the scheme for its weak EF-CMA security from the cases where we use it to produce a strong one-time signature.

*Signature scheme for certificates.* To produce users' secret credentials in our construction, the registration authority will need to sign tuples of $\ell$ elements from $\mathbb{G}$. For this purpose we define the following signature scheme, denoted CERT.

CERT.SETUP$(1^\lambda) \to (\mathsf{vk}_{\text{cert}}, \mathsf{sk}_{\text{cert}})$: Randomly select $\gamma \xleftarrow{R} \mathbb{Z}_p$, $\widehat{g}, \widehat{g}_0 \xleftarrow{R} \widehat{\mathbb{G}}$, $g, h, f_1, f_2 \xleftarrow{R} \mathbb{G}$, and, for $1 \le i \le \ell$, $\widehat{u}_i, \widehat{v}_i \xleftarrow{R} \widehat{\mathbb{G}}$. Output $\mathsf{sk}_{\text{cert}} = \gamma$ and $\mathsf{vk}_{\text{cert}} = (g, h, f_1, f_2, \widehat{g}, \widehat{g}_0, g^\gamma, \widehat{u}_1, \ldots, \widehat{u}_\ell, \widehat{v}_1, \ldots, \widehat{v}_\ell)$.

CERT.SIGN$(\mathsf{vk}_{\text{cert}}, \mathsf{sk}_{\text{cert}}, \mathsf{msg}) \to \sigma$: Given an $\ell$ element message $\mathsf{msg} = (x_1, \ldots, x_\ell)$ in $\mathbb{G}^\ell$, select $\rho, r_1, \ldots, r_\ell, s_1, \ldots, s_\ell \xleftarrow{R} \mathbb{Z}_p$ and compute the signature as $\sigma = \Big(\sigma_\rho, g^\rho, h^\rho, \widehat{g}_0^\rho, \{\sigma_{r_i}, \sigma_{s_i}, g^{r_i}, h^{s_i}, \widehat{u}_i^{r_i}, \widehat{v}_i^{s_i}, (x_i f_1)^{r_i}, (x_i f_2)^{s_i}\}_{1 \le i \le \ell}\Big)$, where $\sigma_\rho = \widehat{g}^{\frac{1}{\gamma+\rho}}$, $\sigma_{r_i} = \widehat{g}^{\frac{1}{\rho + r_i}}$, and $\sigma_{s_i} = \widehat{g}^{\frac{1}{\rho + s_i}}$.

CERT.VERIFY$(\mathsf{vk}_{\text{cert}}, \mathsf{msg}, \sigma) \to 1$ or $0$: To check a signature $\sigma$, we verify that $e(g^\gamma g^\rho,\ \sigma_\rho) = e(g, \widehat{g})$, $e(g^\rho, \widehat{g}_0) = e(g, \widehat{g}_0^\rho)$, $e(h^\rho, \widehat{g}_0) = e(h, \widehat{g}_0^\rho)$ and that, for $1 \le i \le \ell$, $e(g^\rho g^{r_i}, \sigma_{r_i}) = e(g, \widehat{g})$, $e(h^\rho h^{r_i}, \sigma_{s_i}) = e(h, \widehat{g})$, $e(g^{r_i}, \widehat{u}_i) = e(g, \widehat{u}_i^{r_i})$, $e(h^{s_i}, \widehat{v}_i) = e(h, \widehat{v}_i^{s_i})$, $e(x_i f_1, \widehat{u}_i^{r_i}) = e((x_i f_1)^{r_i}, \widehat{u}_i)$, $e(x_i f_2, \widehat{v}_i^{s_i}) = e((x_i f_2)^{s_i}, \widehat{v}_i)$.

The basic idea of CERT.SIGN is to first use the long-term signing key $\gamma$ to sign a one-time signing key $\rho$, then use $\rho$ to sign random numbers $r_i$ and $s_i$, which are in turn used to sign the components of the message. In Appendix B, we prove that this scheme (like BBSIG) satisfies weak EF-CMA security.

*Key-private encryption.* Our construction also makes use of an IK-CPA secure (a.k.a. key-private) encryption scheme which offers a multiplicative homomorphism. Informally, the key privacy property ensures it is infeasible to match a ciphertext with the public key used to produce it; this property is used to achieve receiver anonymity. Below, we give an IK-CPA secure scheme which may be regarded as a variant of linear encryption [BBS04].

IKENC.SETUP$(1^\lambda) \to \mathsf{params}_{\text{ike}}$: Select $\mathsf{params}_{\text{ike}} = (f, h) \xleftarrow{R} \mathbb{G}^2$.

IKENC.GENKEY$(\mathsf{params}_{\text{ike}}) \to (\mathsf{upk}_{\text{ike}}, \mathsf{usk}_{\text{ike}})$: To generate a key pair, select $\mathsf{usk}_{\text{ike}} = (a, b) \xleftarrow{R} \mathbb{Z}_p^2$ and compute $\mathsf{upk}_{\text{ike}} = (f^a, h^b) \in \mathbb{G}^2$.

IKENC.ENC$(\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike}}, \mathsf{msg}, (r, s)) \to C$: To encrypt a $\mathsf{msg} \in \mathbb{G}$ under public key $\mathsf{upk}_{\text{ike}} = (A, B)$ using random exponents $r, s \in \mathbb{Z}_p$, compute $C = (\mathsf{msg}{\cdot}A^r B^s, f^r, h^s)$.

IKENC.DEC$(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike}}, C) \to \mathsf{msg}$: To decrypt a ciphertext $C = (C_1, C_2, C_3)$ with private key $\mathsf{usk}_{\text{ike}} = (a, b)$, compute $\mathsf{msg} = C_1 \cdot C_2^{-a} \cdot C_3^{-b}$.

To denote encryption of a $k$-block message $\vec{x} \in \mathbb{G}^k$, we will use the shorthand IKENC.ENC$(\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike}}, \vec{x}, \vec{r})$, where $\vec{r} \in \mathbb{Z}_p^{2k}$. In Appendix C, we provide a formal definition of IK-CPA security and a proof the above scheme meets it.

The multiplicative homomorphism of this encryption scheme may be evaluated through component-wise multiplication, denoted $\otimes$. Specifically, if $(C_1, C_2, C_3)$ and $(C_1', C_2', C_3')$ are encryptions of $x$ and $x'$ using the same $\mathsf{upk}_{\text{ike}}$ and exponents $r, s$ and $r', s'$ respectively, then $(C_1, C_2, C_3) \otimes (C_1', C_2', C_3')$ is the encryption of $x \cdot x'$ under $r + r'$ and $s + s'$. Also, we will write $(C_1, C_2, C_3) \odot x'$ to denote $(C_1 \cdot x', C_2, C_3)$, which is an encryption of $x \cdot x'$ under the original randomness $r, s$.

When using the above homomorphism to compute an encryption of $x \cdot x'$, the distribution of the resulting ciphertext is dependent on that of the input ciphertexts. In our scheme, we will need to rerandomize the ciphertexts to remove this dependency. Furthermore, we will need to do so without knowledge of the $\mathsf{upk}_{\mathrm{ike}}$ used for encryption. Observe that this is possible if we have available two encryptions of $1 \in \mathbb{G}$ under independent random exponents. Specifically, suppose $C_x$ is an encryption of $x$ using $r_x, s_x$ and $C_1$, $C_2$ are encryptions of 1 using $r_1, s_1$ and $r_2, s_2$, respectively. Select $t_1, t_2 \overset{R}{\longleftarrow} \mathbb{Z}_p$ and compute $C'_x = C_x \otimes C_1^{t_1} \otimes C_2^{t_2}$, where $C_1^{t_1}$ and $C_2^{t_2}$ denote componentwise exponentiation. Then $C'_x$ is an encryption of $x$ using exponents $r_x + r_1 t_1 + r_2 t_2$ and $s_x + s_1 t_1 + s_2 t_2$, and the distribution of $C'_x$ is independent of the distribution of $C_x$.

*Assumptions.* Here we detail the complexity assumptions necessary to prove the privacy and security properties of our constructions. In addition to the well-known decisional linear (DLinear) and strong Diffie-Hellman (SDH) assumptions, we employ the following three assumptions, the first two of which are parameterized by a positive integer $q$.

**BB-HSDH** Select $\gamma \overset{R}{\longleftarrow} \mathbb{Z}_p^*$, $g \overset{R}{\longleftarrow} \mathbb{G}$, $\widehat{g}, \widehat{g}_0 \overset{R}{\longleftarrow} \widehat{\mathbb{G}}$, and $\rho_i \overset{R}{\longleftarrow} \mathbb{Z}_p$ for $i \in \{1, \ldots, q\}$. Then given $(g, g^\gamma, \widehat{g}, \widehat{g}^\gamma, \widehat{g}_0, (\rho_i, \widehat{g}^{\frac{1}{\gamma+\rho_i}})_{1 \le i \le q})$, it is computationally infeasible to output a tuple $(g^\rho, \widehat{g}_0^\rho, \widehat{g}^{\frac{1}{\gamma+\rho}})$ where $\rho \notin \{\rho_1, \ldots, \rho_q\}$.

**BB-CDH** Select $\gamma \overset{R}{\longleftarrow} \mathbb{Z}_p^*$, $g \overset{R}{\longleftarrow} \mathbb{G}$, $\widehat{g}, \widehat{u} \overset{R}{\longleftarrow} \widehat{\mathbb{G}}$, and $\rho_i \overset{R}{\longleftarrow} \mathbb{Z}_p$ for $i \in \{1, \ldots, q\}$. Then given $(g, g^\gamma, \widehat{g}, \widehat{g}^\gamma, \widehat{u}, (\rho_i, \widehat{g}^{\frac{1}{\gamma+\rho_i}})_{1 \le i \le q})$, it is infeasible to output $\widehat{u}^\gamma$.

**SCDH** Select $\rho, r, s \overset{R}{\longleftarrow} \mathbb{Z}_p$, $g, h \overset{R}{\longleftarrow} \mathbb{G}$, and $\widehat{g}, \widehat{u}, \widehat{v} \overset{R}{\longleftarrow} \widehat{\mathbb{G}}$. Then given $(\rho, g, h, \widehat{g}, \widehat{u}, \widehat{v}, \widehat{u}^r, \widehat{v}^s, g^r, h^s, \widehat{g}^{\frac{1}{r+\rho}}, \widehat{g}^{\frac{1}{s+\rho}})$ it is infeasible to output a tuple $(z, z^r, z^s)$ where $z \in \mathbb{G}$ and $z \neq 1$.

The first two assumptions above were introduced in the delegatable anonymous credential work of Belenkiy et. al. [BCC+09]. The SCDH ("stronger than CDH") assumption is new; we provide a proof of its hardness in generic groups in Appendix A. Note that if we remove the terms $\widehat{g}^{\frac{1}{r+\rho}}, \widehat{g}^{\frac{1}{s+\rho}}$ from the SCDH assumption, the resulting assumption would be implied by DLinear. Therefore, we are assuming that these two terms "will not help" the adversary in outputting $(z, z^r, z^s)$.

## 5 Main Constructions

To better motivate our full construction, we first present a simpler, "unblinded" version which neglects the receiver anonymity property and assumes users correctly follow the protocol. These algorithms will form part of the full version.

*Unblinded scheme.* In unblinded scheme, each user $i$ generates a voting key $\mathsf{votekey}_i$ and a receiver key $\mathsf{rcvkey}_i$. Given $\mathsf{rcvkey}_i$, a user $j$ can use its $\mathsf{votekey}_j$ to compute an unblinded vote $U_{j,i}$ for user $i$. User $i$ can then demonstrate its reputation by showing a "weak encryption" of the unblinded votes it has received.

SetupUnblinded$(1^\lambda) \to \mathsf{params}_{\mathrm{ub}}$: Select $\mathsf{params}_{\mathrm{ub}} = (g, h) \overset{R}{\longleftarrow} \mathbb{G}^2$.

GenKey$(\mathsf{params}_{\mathrm{ub}}) \to \mathsf{rcvkey}_i, \mathsf{votekey}_i$: To make a key pair for user $i$, select $\alpha_i, \beta_i \overset{R}{\longleftarrow} \mathbb{Z}_p$ and $x_{i,k}, y_{i,k}, z_{i,k} \overset{R}{\longleftarrow} \mathbb{G}$ for $k \in \{1, 2\}$. Define $\mathsf{vsk}_i = (\alpha_i, \beta_i)$ and $\mathsf{vpk}_i = (g^{\alpha_i}, h^{\beta_i}, z_{i,1}, z_{i,2})$ and output $\mathsf{rcvkey}_i = (x_{i,1}, y_{i,1}, x_{i,2}, y_{i,2})$ and $\mathsf{votekey}_i = (\mathsf{vsk}_i, \mathsf{vpk}_i)$.

VoteUnblinded$(\mathsf{rcvkey}_i, \mathsf{votekey}_j) \to U_{j,i}$: To compute a vote from $j$ to $i$, we parse the keys as above, using subscripts $i, j$ to distinguish the components of user $i$'s key and user $j$'s key, then output $U_{j,i} = (x_{i,1}^{\alpha_j} \cdot y_{i,1}^{\beta_j} \cdot z_{j,1},\ x_{i,2}^{\alpha_j} \cdot y_{i,2}^{\beta_j} \cdot z_{j,2})$.

$\textsc{ShowRep}(U_1, \ldots, U_c, (r, s)) \to \mathsf{rep}$: To compute the weakly encrypted version of $c$ unblinded votes using random exponents $r, s \in \mathbb{Z}_p$, we output $\mathsf{rep} = [\ u_{i,1}^r \cdot u_{i,2}^s\ ]_{1 \le i \le c}$, where $u_{i,1}, u_{i,2}$ denote the two components of $U_i$.

These algorithms are designed to ensure several properties we will need when they are used within the full construction. First, an unblinded vote $U_{j,i}$ is a deterministic function of $\mathsf{votekey}_j$ and $\mathsf{rcvkey}_i$, so votes $U_{j_1,i}$, $U_{j_2,i}$ from distinct voters $j_1 \neq j_2$ will have distinct values. Furthermore, $\textsc{ShowRep}$ preserves this distinctness, so if $U_{j_1,i} \neq U_{j_2,i}$ and $(V_{j_1,i}, V_{j_2,i}) = \textsc{ShowRep}(U_{j_1,i}, U_{j_2,i}, (r, s))$, then $V_{j_1,i} \neq V_{j_2,i}$. Second, without $\mathsf{votekey}_j$, an adversary cannot forge a vote from user $j$ (based on the CDH assumption). These two properties will be needed for the *soundness* of the full construction. Third, given $U_{j_1,i_1}, U_{j_2,i_2}$, two colluding receivers $i_1$ and $i_2$ cannot determine whether $j_1 = j_2$.[3] This relies on the DLinear assumption and will help ensure *voter anonymity.* Finally, if $\textsc{ShowRep}$ is invoked twice on the same unblinded votes but with independent randomness, the resulting values $\mathsf{rep}_1$, $\mathsf{rep}_2$ cannot be linked to one another. This also relies on the DLinear assumption and will be used to help ensure *signer anonymity.*

*Full construction.* The full construction is given in Figures 2 and 3. As for the opening algorithms, $\textsc{Setup}'$ is obtained from $\textsc{Setup}$ by simply returning the extractor key $\mathsf{xk}$ of the Groth-Sahai proof system as $\mathsf{openkey} = \mathsf{xk}$ rather than discarding it. The $\textsc{OpenNym}$ algorithm then uses the extractor key on the NIZK in a $\mathsf{nym}$ to obtain the committed $\mathsf{rcvkey}$, which may then be matched against a list of credentials $\mathsf{cred}_1, \ldots, \mathsf{cred}_n$ to determine the owner of $\mathsf{nym}$. Similarly, $\textsc{OpenVote}$ works by using the extractor key to obtain the $\mathsf{rcvkey}$ of the voter from the commitment in the vote's NIZK.

The algorithms of Figures 2 and 3 (along with opening algorithms described above) satisfy Definitions 1–6. The correctness properties may be verified by inspection; for each of the other properties, we provide detailed proofs in Appendices B–E. Intuitively, the full scheme is obtained through three modifications to the unblinded scheme. First, to limit voting to valid members of the system, a user's $\mathsf{rcvkey}$ and $\mathsf{votekey}$ are signed and issued by the registration authority. Second, we use a "blinded" voting protocol based on key-private, homomorphic encryption to achieve receiver anonymity. Third, users construct NIZKs to prove they have correctly followed the protocol. We now elaborate on the later two ideas and the operation of the $\textsc{SignRep}$ algorithm.

*Blinded voting.* From a high level, a user computes a one-time pseudonym $\mathsf{nym}$ by encrypting their $\mathsf{rcvkey}$ under their $\mathsf{upk}_{\mathrm{ike}}$. Instead of voting on the $\mathsf{rcvkey}$, a voter then votes on the encrypted version in the $\mathsf{nym}$. This is made possible by the homomorphism of the encryption scheme: the voter homomorphically computes an encryption of the unblinded vote, which the recipient can later decrypt. Only the recipient has the secret key $\mathsf{usk}_{\mathrm{ike}}$ necessary to do so.

More precisely, if $(C_{x,k}, C_{y,k})_{k \in \{1,2\}}$ is the encryption of $\mathsf{rcvkey} = (x_1, y_1, x_2, y_2)$, the voter computes the encrypted vote as $(C_{x,k}^{\alpha} \otimes C_{y,k}^{\beta} \odot z_k)$, where $\alpha, \beta, z_k$ come from the voter's key. To allow the voter to rerandomize the resulting ciphertext using the technique described in Section 4, the recipient also includes two independent encryptions of $1 \in \mathbb{G}$ in the $\mathsf{nym}$, which we denote $C_{1,1}$ and $C_{1,2}$. To understand the requirement that $\textsc{IKEnc}$ be selective-tag weakly CCA-secure, recall that in the security definition, when an adversary makes a "signrep" oracle query, it can indirectly learn whether one or more votes correspond to the user $i$. This allows the oracle to be used as something similar to

[3] Note that, if the term $z_{j,k}$ is omitted, an attack is possible. Two colluding users vote for a recipient $i$, resulting in two votes $U_1, U_2$. Later, when $i$ constructs $\mathsf{rep} = \textsc{ShowRep}(U_1, U_2, (r, s))$, the adversary would be able to detect the correlation in $\mathsf{rep}$ and confirm that it came from $i$. The term $z_{j,k}$ prevents this because the adversary does not know its exponent.

```
$\underline{\textsc{Setup}}(1^\lambda)$.
  $(\mathsf{crs}, \mathsf{xk}) \leftarrow \mathrm{GS}.\textsc{Setup}(1^\lambda)$
  $\mathsf{params}_{\mathrm{ub}} \leftarrow \textsc{SetupUnblinded}(1^\lambda)$
  $\mathsf{params}_{\mathrm{ike}} \leftarrow \mathrm{IKEnc}.\textsc{Setup}(1^\lambda)$
  $(\mathsf{pk}_{\mathrm{cca}}, \mathsf{sk}_{\mathrm{cca}}) \leftarrow \mathrm{CCAEnc}.\textsc{Setup}(1^\lambda)$
  $(\mathsf{vk}_{\mathrm{cert}}, \mathsf{sk}_{\mathrm{cert}}) \leftarrow \mathrm{Cert}.\textsc{Setup}(1^\lambda)$ ,
  Return
  $\mathsf{params} = (\mathsf{crs}, \mathsf{params}_{\mathrm{ub}}, \mathsf{params}_{\mathrm{ike}}, \mathsf{pk}_{\mathrm{cca}}, \mathsf{vk}_{\mathrm{cert}})$,
  $\mathsf{authkey} = \mathsf{sk}_{\mathrm{cert}}$.
```

```
$\underline{\textsc{GenCred}}(\mathsf{params}, \mathsf{authkey})$.
  $(\mathsf{rcvkey}, \mathsf{vsk}, \mathsf{vpk}) \leftarrow \textsc{GenKey}(\mathsf{params}_{\mathrm{ub}})$
  $(\mathsf{vk}_{\mathrm{bb}}, \mathsf{sk}_{\mathrm{bb}}) \leftarrow \mathrm{BBSig}.\textsc{Setup}(1^\lambda)$
  $(\mathsf{upk}_{\mathrm{ike}}, \mathsf{usk}_{\mathrm{ike}}) \leftarrow \mathrm{IKEnc}.\textsc{GenKey}(\mathsf{params}_{\mathrm{ike}})$
  Denote $\mathit{pub\_cred} := (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}})$
  $\mathsf{cert} \leftarrow \mathrm{Cert}.\textsc{Sign}(\mathsf{vk}_{\mathrm{cert}}, \mathsf{sk}_{\mathrm{cert}}, \mathit{pub\_cred})$
  Return $\mathsf{cred} = (\mathit{pub\_cred}, \mathsf{cert}, \mathsf{sk}_{\mathrm{bb}}, \mathsf{vsk}, \mathsf{usk}_{\mathrm{ike}})$
```

```
$\underline{\textsc{GenNym}}(\mathsf{params}, \mathsf{cred})$.
  Parse $\mathsf{cred} = (\mathit{pub\_cred}, \mathsf{cert}, \mathsf{sk}_{\mathrm{bb}}, \ldots)$
  Parse $\mathit{pub\_cred} = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}})$
  Denote $\mathsf{msg} := (\mathsf{rcvkey}, 1, 1) \in \mathbb{G}^6$
  $(\mathsf{vk}_{\mathrm{ots}}, \mathsf{sk}_{\mathrm{ots}}) \leftarrow \mathrm{BBSig}.\textsc{Setup}(1^\lambda)$,   $\vec{r} \xleftarrow{R} \mathbb{Z}_p^{12}$
  $C \leftarrow \mathrm{IKEnc}.\textsc{Enc}(\mathsf{params}_{\mathrm{ike}}, \mathsf{upk}_{\mathrm{ike}}, \mathsf{msg}, \vec{r})$
  $\sigma_{\mathrm{bb}} \leftarrow \mathrm{BBSig}.\textsc{Sign}(\mathsf{sk}_{\mathrm{bb}}, H(\mathsf{vk}_{\mathrm{ots}}))$
  $\Pi = \mathrm{NIZK}\{\ \mathit{pub\_cred}, \mathsf{cert}, \sigma_{\mathrm{bb}}, \vec{r}\ :$
      $\mathrm{Cert}.\textsc{Verify}(\mathsf{vk}_{\mathrm{cert}}, \mathit{pub\_cred}, \mathsf{cert})$
    $\wedge\ \mathrm{BBSig}.\textsc{Verify}(\mathsf{vk}_{\mathrm{bb}}, H(\mathsf{vk}_{\mathrm{ots}}), \sigma_{\mathrm{bb}})$
    $\wedge\ C = \mathrm{IKEnc}.\textsc{Enc}(\mathsf{params}_{\mathrm{ike}}, \mathsf{upk}_{\mathrm{ike}}, \mathsf{msg}, \vec{r})\}$
  $\sigma_{\mathrm{ots}} \leftarrow \mathrm{BBSig}.\textsc{Sign}(\mathsf{sk}_{\mathrm{ots}}, H(C \| \Pi \| \mathsf{vk}_{\mathrm{ots}}))$
  Return $\mathsf{nym} = (C, \Pi, \mathsf{vk}_{\mathrm{ots}}, \sigma_{\mathrm{ots}})$
```

```
Subroutine: $\underline{\textsc{VerifyNym}}(\mathsf{params}, \mathsf{nym})$.
  Parse $\mathsf{nym} = (C, \Pi, \mathsf{vk}_{\mathrm{ots}}, \sigma_{\mathrm{ots}})$
  Denote $\mathit{pub\_vals} := \langle \mathsf{params}, C, \mathsf{vk}_{\mathrm{ots}} \rangle$
  If $\mathrm{BBSig}.\textsc{Verify}(\mathsf{vk}_{\mathrm{ots}}, H(C \| \Pi \| \mathsf{vk}_{\mathrm{ots}}), \sigma_{\mathrm{ots}})$
    $\wedge\ \mathrm{GS}.\textsc{Verify}(\mathsf{crs}, \Pi, \mathit{pub\_vals})$
  Return 1;   Else return 0
```

```
$\underline{\textsc{Vote}}(\mathsf{params}, \mathsf{cred}, \mathsf{nym})$.
  Parse $\mathsf{cred} = (\mathit{pub\_cred}, \mathsf{cert}, \mathsf{sk}_{\mathrm{bb}}, \mathsf{vsk}, \mathsf{usk}_{\mathrm{ike}})$
  Parse $\mathit{pub\_cred} = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}})$
  Parse $\mathsf{nym} = (C, \ldots)$
        where $C = (\{C_{x,k}, C_{y,k}\}_{k \in \{1,2\}}, C_{1,1}, C_{1,2})$
  Parse $\mathsf{vsk} = (\alpha, \beta), \mathsf{vpk} = (A, B, z_1, z_2)$
  Parse $\mathsf{rcvkey} = (x_1, \ldots)$
  If $\neg\textsc{VerifyNym}(\mathsf{params}, \mathsf{nym})$ Return $\perp$
  $(\mathsf{vk}_{\mathrm{ots}}, \mathsf{sk}_{\mathrm{ots}}) \leftarrow \mathrm{BBSig}.\textsc{Setup}(1^\lambda)$,
  $\mathsf{tag} = H(\mathsf{vk}_{\mathrm{ots}})$,   $\sigma_{\mathrm{bb}} \leftarrow \mathrm{BBSig}.\textsc{Sign}(\mathsf{sk}_{\mathrm{bb}}, \mathsf{tag})$
  $\vec{r} = (r_{1,1}, r_{1,2}, r_{2,1}, r_{2,2}) \xleftarrow{R} \mathbb{Z}_p^4$,   $\vec{s} \xleftarrow{R} \mathbb{Z}_p^2$
  $C_1 = \left[(C_{x,k}^{\alpha} \otimes C_{y,k}^{\beta} \odot z_k) \otimes C_{1,1}^{r_{k,1}} \otimes C_{1,2}^{r_{k,2}}\right]_{k \in \{1,2\}}$
  $C_2 \leftarrow \mathrm{CCAEnc}.\textsc{Enc}(\mathsf{pk}_{\mathrm{cca}}, \mathsf{tag}, x_1, \vec{s})$
  $\Pi = \mathrm{NIZK}\{\ \mathit{pub\_cred}, \mathsf{cert}, \mathsf{vsk}, \sigma_{\mathrm{bb}}, \vec{r}, \vec{s}\ :$
      $\mathrm{Cert}.\textsc{Verify}(\mathsf{vk}_{\mathrm{cert}}, \mathit{pub\_cred}, \mathsf{cert})$
    $\wedge\ \mathrm{BBSig}.\textsc{Verify}(\mathsf{vk}_{\mathrm{bb}}, \mathsf{tag}, \sigma_{\mathrm{bb}})$
    $\wedge\ A = g^{\alpha}\ \wedge\ B = h^{\beta}$
    $\wedge\ C_1 = \left[(C_{x,k}^{\alpha} \otimes C_{y,k}^{\beta} \odot z_k) \otimes C_{1,1}^{r_{k,1}} \otimes C_{1,2}^{r_{k,2}}\right]_{k \in \{1,2\}}$
    $\wedge\ C_2 = \mathrm{CCAEnc}.\textsc{Enc}(\mathsf{pk}_{\mathrm{cca}}, \mathsf{tag}, x_1, \vec{s})\ \}$
  $\sigma_{\mathrm{ots}} \leftarrow \mathrm{BBSig}.\textsc{Sign}(\mathsf{sk}_{\mathrm{ots}}, H(\mathsf{nym} \| C_1 \| C_2 \| \Pi \| \mathsf{vk}_{\mathrm{ots}}))$
  Return $\mathsf{vt} = (\mathsf{nym}, C_1, C_2, \Pi, \mathsf{vk}_{\mathrm{ots}}, \sigma_{\mathrm{ots}})$
```

```
$\underline{\textsc{VerifyRep}}(\mathsf{params}, \mathsf{msg}, \Sigma)$.
  Parse $\Sigma = (c, \mathsf{msg}, C, \mathsf{rep}, \Pi, \mathsf{vk}_{\mathrm{ots}}, \sigma_{\mathrm{ots}})$
  If $|\mathsf{rep}| = c$ and
    There are no duplicate values in $\mathsf{rep}$ and
    $\mathrm{GS}.\textsc{Verify}(\mathsf{crs}, \Pi, \langle \mathsf{params}, C, \mathsf{rep}, \mathsf{vk}_{\mathrm{ots}} \rangle)$ and
    $\mathrm{BBSig}.\textsc{Verify}(\mathsf{vk}_{\mathrm{ots}}, H(c \| \mathsf{msg} \| C \| \mathsf{rep} \| \Pi \| \mathsf{vk}_{\mathrm{ots}}), \sigma_{\mathrm{ots}})$
  Return $c$;   Else return $\perp$
```

```
Subroutine: $\underline{\textsc{VerifyVote}}(\mathsf{params}, \mathsf{vt})$.
  Parse $\mathsf{vt} = (\mathsf{nym}, C_1, C_2, \Pi, \mathsf{vk}_{\mathrm{ots}}, \sigma_{\mathrm{ots}})$
  Denote $\mathit{pub\_vals} := \langle \mathsf{params}, C_1, C_2, \mathsf{vk}_{\mathrm{ots}} \rangle$
  If $\mathrm{BBSig}.\textsc{Verify}(\mathsf{vk}_{\mathrm{ots}}, H(\mathsf{nym} \| C_1 \| C_2 \| \Pi \| \mathsf{vk}_{\mathrm{ots}}), \sigma_{\mathrm{ots}})$
    $\wedge\ \mathrm{GS}.\textsc{Verify}(\mathsf{crs}, \Pi, \mathit{pub\_vals})$
    $\wedge\ \textsc{VerifyNym}(\mathsf{params}, \mathsf{nym})$
  Return 1;   Else return 0
```

Figure 2: The Setup, GenCred, GenNym, Vote, and VerifyRep algorithms.

a decryption oracle, ultimately requiring a CCA security property. To ensure an adversary cannot frame a honest user $i$ by forging a $\mathsf{nym}$ that opens to user $i$, the GenNym algorithm also picks a one-time signature key pair $(\mathsf{sk}_{\mathrm{ots}}, \mathsf{vk}_{\mathrm{ots}})$ and proves knowledge of a signature $\sigma_{\mathrm{bb}} \leftarrow \mathrm{BBSig}.\textsc{Sign}(\mathsf{sk}_{\mathrm{bb}}, H(\mathsf{vk}_{\mathrm{ots}}))$, then uses $\mathsf{sk}_{\mathrm{ots}}$ to sign the entire $\mathsf{nym}$. One-time signatures are similarly employed in Groth's group signature scheme [Gro07]; we also use this technique in the votes and signatures of reputation.

```
$\underline{\textsc{SignRep}}(\mathsf{params}, \mathsf{cred}, V, \mathsf{msg})$
  Parse $\mathsf{cred} = (pub\_cred, \mathsf{cert}, \mathsf{sk_{bb}}, \mathsf{vsk}, \mathsf{usk_{ike}})$ where $pub\_cred = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk_{bb}}, \mathsf{upk_{ike}})$
  Parse $\mathsf{params_{ike}} = (f, h)$,  $\mathsf{upk_{ike}} = (A, B)$,  $\mathsf{usk_{ike}} = (a, b)$
  Parse $V = \{\mathsf{vt}_1, \mathsf{vt}_2, \dots, \mathsf{vt}_c\}$ where $\mathsf{vt}_i = (\mathsf{nym}_i, C_{1,i}, C_{2,i}, \Pi_i, \mathsf{vk}_{\mathrm{ots},i}, \sigma_{\mathrm{ots},i})$
  Denote $\mathsf{tag}_i = H(\mathsf{vk}_{\mathrm{ots},i})$
  $\forall 1 \le i \le c'$ : Parse $\mathsf{nym}_i = (C'_i, \Pi'_i, \mathsf{vk_{ots}}'_{,i}, \sigma_{\mathrm{ots}}{}'_{,i})$
  If $\exists 1 \le i \le c'$ : $1 \ne \textsc{VerifyVote}(\mathsf{params}, \mathsf{vt}_i)$, return $\perp$
  If $\exists 1 \le i \le c'$ : $\mathsf{rcvkey} \ne \text{IKE}\textsc{nc}.\textsc{Dec}(\mathsf{params_{ike}}, \mathsf{usk_{ike}}, C'_i)$, return $\perp$
  $(\mathsf{vk_{ots}}, \mathsf{sk_{ots}}) \leftarrow \text{BBS}\textsc{ig}.\textsc{Setup}(1^\lambda)$,  $\mathsf{tag} = H(\mathsf{vk_{ots}})$,  $\sigma_{\mathrm{bb}} \leftarrow \text{BBS}\textsc{ig}.\textsc{Sign}(\mathsf{sk_{bb}}, H(\mathsf{vk_{ots}}))$
  $\forall 1 \le i \le c'$ : $U'_i \leftarrow \text{IKE}\textsc{nc}.\textsc{Dec}(\mathsf{params_{ike}}, \mathsf{usk_{ike}}, C_{1,i})$
  Remove duplicates: $\{U_1, U_2, \dots, U_c\} := \{U'_1, \dots, U'_{c'}\}$, where $U_1, \dots, U_c$ are all distinct
  $\vec{r} \xleftarrow{R} \mathbb{Z}_p^2$,  $\mathsf{rep} \leftarrow \textsc{ShowRep}((U_1, \dots, U_c), \vec{r})$
  $\vec{s} \xleftarrow{R} \mathbb{Z}_p^{2c}$,  $C \leftarrow \text{CCAE}\textsc{nc}.\textsc{Enc}(\mathsf{pk_{cca}}, \mathsf{tag}, (U_1, \dots, U_c), \vec{s})$
  $\Pi = \text{NIZK}\{\ pub\_cred, \mathsf{cert}, \mathsf{usk_{ike}}, \sigma_{\mathrm{bb}}, (\mathsf{vt}_i, \mathsf{tag}_i, U_i)_{1 \le i \le c}, \vec{r}, \vec{s}$ :
            $\wedge\ \textsc{Cert}.\textsc{Verify}(\mathsf{vk_{cert}}, pub\_cred, \mathsf{cert})$
            $\wedge\ \text{BBS}\textsc{ig}.\textsc{Verify}(\mathsf{vk_{bb}}, \mathsf{tag}, \sigma_{\mathrm{bb}})\ \wedge\ A = f^a\ \wedge B = h^b$
            $\wedge\ \forall 1 \le i \le c$ : GS.$\textsc{Verify}(\mathsf{crs}, \Pi_i, \langle \mathsf{params}, C_{1,i}, C_{2,i}, \mathsf{tag}_i \rangle)$
            $\wedge\ \forall 1 \le i \le c$ : GS.$\textsc{Verify}(\mathsf{crs}, \Pi'_i, \langle \mathsf{params}, C'_i \rangle)$          $(*)$
            $\wedge\ \forall 1 \le i \le c$ : $\mathsf{rcvkey} = \text{IKE}\textsc{nc}.\textsc{Dec}(\mathsf{params_{ike}}, \mathsf{usk_{ike}}, C'_i)$
            $\wedge\ \forall 1 \le i \le c$ : $U_i = \text{IKE}\textsc{nc}.\textsc{Dec}(\mathsf{params_{ike}}, \mathsf{usk_{ike}}, C_{1,i})$
            $\wedge\ \mathsf{rep} = \textsc{ShowRep}((U_1, \dots, U_c), \vec{r})\ \wedge\ C = \text{CCAE}\textsc{nc}.\textsc{Enc}(\mathsf{pk_{cca}}, \mathsf{tag}, (U_1, \dots, U_c), \vec{s})\ \}$
  $\sigma_{\mathrm{ots}} \leftarrow \text{BBS}\textsc{ig}.\textsc{Sign}(\mathsf{sk_{ots}}, H(c \| \mathsf{msg} \| C \| \mathsf{rep} \| \Pi \| \mathsf{vk_{ots}}))$
  Return $\Sigma = (c, \mathsf{msg}, C, \mathsf{rep}, \Pi, \mathsf{vk_{ots}}, \sigma_{\mathrm{ots}})$
  (*): For this line, verify all equations in the NIZK $\Pi'_i$, except the BBSig.Verify equation.
```
Figure 3: The SignRep algorithm.

*Nested NIZKs.* Users must prove through a series of NIZKs that they have correctly followed the algorithms using credentials certified by the registration authority. It is worth mentioning that we use "nested" NIZKs. Specifically, a signature of reputation includes a commitment to the votes and a NIZK proving they are valid. Because the votes themselves contain NIZKs, proving that the votes are valid involves proving that the NIZKs they contain satisfy the Groth-Sahai NIZK verification equations, all within the NIZK for the resulting signature.

*Signatures of reputation.* To construct a signature of reputation, the signer uses $\mathsf{usk_{ike}}$ to decrypt the ciphertexts in the votes they have received, obtaining unblinded votes $U_1, \dots, U_c$. It calls $\textsc{ShowRep}(U_1, \dots, U_c)$ to compute a weak encryption of these unblinded votes. Recall that this encryption preserves "distinctness". It also encrypts these unblinded votes using CCAEnc; this allows a simulator to open the signature of reputation under a simulated $\mathsf{crs}$ without $\mathsf{xk}$.

## 6 Short Signatures of Reputation

What we have described thus far produces signatures of reputation $c$ that are of size $\Theta(c)$. If perfect soundness is not necessary, this cost can be dramatically reduced. Specifically, in this section we describe a way to obtain signatures of size $O(\frac{1}{\varepsilon} \log c)$ while maintaining $\varepsilon$-soundness (Definition 7) in the random oracle model.

From a high level, we take the following approach in improving space efficiency. Rather than including all votes in the signature, we only include a randomly selected, constant size subset. Specifically, we require that the signer first commit to a list of all the votes with a hash function $H$ and then interpret the output of $H$ as a challenge specifying the indices of the votes to include. The signer must also demonstrate that the votes included were in fact the votes at the indices in the challenge set when the commitment was formed. To do so efficiently, we compute the commitment using a Merkle hash tree [Mer89]. We implement this technique with the following changes to the SignRep algorithm.

After determining the number of distinct unblinded votes $c$, set $\ell = \lceil \frac{\lambda}{\varepsilon} \rceil$; this will be the size of our challenge set. Recall that $\lambda$ is the security parameter. Now if $\ell \geq c$, we include all votes, computing $\Sigma$ normally. Otherwise, we proceed as follows. Let $\mathsf{rep} = (R_1, \ldots R_c)$. Sort these values to obtain a list $R_{\rho_1}, R_{\rho_2}, \ldots, R_{\rho_c}$, where $R_{\rho_i} < R_{\rho_{i+1}}$ for $1 \leq i \leq c-1$. The NIZK $\Pi$ computed by SignRep will include the following values for the $\rho_i$th vote: a tuple $\theta_i$ of commitments to $\mathsf{vt}_i, \mathsf{tag}_i, U_i$ and a tuple of values $\zeta_i$ used to verify the GS.Verify and IKEnc.Dec equations. We collect these together to form $\omega_i = (R_{\rho_i}, R_{\rho_{i+1}}, \theta_i, \zeta_i)$, with $R_{\rho_{c+1}}$ defined as a special symbol $\infty$ for consistency.

Now we can compute the set of challenge indices. Let $m = \lceil \log_2 c \rceil$ be the height of the smallest binary tree with at least $c$ leaf nodes. We construct a complete binary tree of height $m$, associating the first $c$ leaf nodes with the values $\omega_1, \omega_2, \ldots, \omega_c$ and any remaining leaf nodes with dummy values $\omega_{c+1}, \ldots, \omega_{2^m} = 0$. Next, we compute a hash value $h_n$ for each node $n$ in the tree as follows. If $n$ is a leaf with index $i$, we set $h_n = H(\omega_i)$; otherwise, $n$ has a left child $n_l$ and a right child $n_r$ and we set $h_n = H(h_{n_l} \| h_{n_r})$. We thus obtain a hash value $h_{\text{root}}$ for the root of the tree to be used to construct a set of $\ell$ distinct challenge indices $I \subset \{1, 2, \ldots, c\}$. This is done by starting with an empty set $I$, and adding the indices $1 + (H(0 \| h_{\text{root}}) \bmod c)$, $1 + (H(1 \| h_{\text{root}}) \bmod c)$, etc., one by one, skipping duplicates and stopping when $|I| = \ell$.

Now that we have specified the challenge set $I$, we may list the values included in the final signature of reputation. We start with the proof $\Pi$ computed as before and remove all per-vote values $\theta_{\rho_i}, \zeta_{\rho_i}$ for $i \notin I$. Note that the result is a valid Groth-Sahai NIZK that only verifies the votes at indices in $I$; furthermore, it is distributed identically to a proof computed directly using only those votes. In addition to the reduced proof $\Pi$, we include in the final signature of reputation the pairs $(R_{\rho_i}, R_{\rho_{i+1}})$ for each $i \in I$ and the off-path hashes needed to verify that the challenge set was constructed correctly. There are precisely $\lceil \log_2 c \rceil$ off-path hash values for each vote (although some will be shared by multiple votes), so we obtain an overall signature size of $O(\ell \log c) = O(\frac{1}{\varepsilon} \log c)$.

The necessary modifications to the VerifyRep algorithm are straightforward. We verify the proof $\Pi$ normally, then collect each of the per-vote terms present and hash them with $H$ to obtain the values of the corresponding leaves in the hash tree. Using the provided off-path hash values, we recompute the root value $h_{\text{root}}$. From $h_{\text{root}}$, we compute the challenge set $I$, and then we check that it corresponds to the votes that were included. Also, for each pair $(R_{\rho_i}, R_{\rho_{i+1}})$, we check that $R_{\rho_i} < R_{\rho_{i+1}}$.

Let SignRep$'$ and VerifyRep$'$ denote the modified versions of the SignRep and VerifyRep algorithms as described above. Then the algorithms Setup, GenCred, GenNym, Vote, SignRep$'$, and VerifyRep$'$ constitute an $\varepsilon$-sound scheme for signatures of reputation. In Appendix F, we prove this in the random oracle model.

## 7 Conclusion and Open Problems

In summary, we have provided a formalization of the concept of signatures of reputation and a construction that meets our definitions. Our construction supports monotonic measures of reputation in a completely anonymous setting. In our system, a user can express trust in others by voting for them, collect votes to build up her own reputation, and attach a proof of her reputation to any data she publishes, all while maintaining the unlinkability of her actions.

Because this work concerns a novel concept, many interesting open questions remain. Most importantly, how can we support non-monotonic reputation systems, which can express and enforce "bad" reputation as well as good? Answering this question will require innovative definitions as well as cryptographic constructions. Other challenges include devising a scheme which maintains privacy despite a malicious registration authority and handling measures of reputation more expressive than the vote counting of in this paper.

## References

[ACBM08] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In *Privacy Enhancing Technologies*, 2008.

[Arr06] Michael Arrington. AOL proudly releases massive amounts of user search data. *TechCrunch News*, August 2006.

[ASM08] Man Ho Au, Willy Susilo, and Yi Mu. Practical anonymous divisible e-cash from bounded accumulators. In *Financial Cryptography*, 2008.

[AWSM07] Man Ho Au, Q. Wu, Willy Susilo, and Yi Mu. Compact e-cash from bounded accumulator. In *CT-RSA*, 2007.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptol.*, 21(2), 2008.

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *Asiacrypt*, 2001.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Crypto*, 2004.

[BCC$^+$09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Crypto*, 2009.

[BCE$^+$07] Mira Belenkiy, Melissa Chase, Chris Erway, John Jannotti, Alptekin Kupcu, Anna Lysyanskaya, and Eric Rachlin. Making p2p accountable without losing privacy. In *WPES*, 2007.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Non-interactive anonymous credentials. In *TCC*, 2008.

[BDK07] Lars Backstrom, Cynthia Dwork, and Jon M. Kleinberg. Wherefore art thou r3579x? In *International World Wide Web Conference*, 2007.

[BF01] Dan Boneh and Matthew Franklin. Identity based encryption from the weil pairing. In *Crypto*, 2001.

[CHK$^+$06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clone wars: Efficient periodic n-times anonymous authentication. In *CCS*, 2006.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Eurocrypt*, 2005.

[CHL06] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks (SCN)*, 2006.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *PKC*, 2009.

[CLM07] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security and Privacy*, 2007.

[CMB02] Ingemar Cox, Matthew Miller, and Jeffery Bloom. *Digital Watermarking.* Morgan Kaufmann, 2002.

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Inst. for TCS, ETH Zurich, 1997.

[DGS03] Ivan Damgard, Jens Groth, and Gorm Salomonsen. The theory and implementation a of an electronic voting system. In *Secure Electronic Voting*, 2003.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt*, 1999.

[Gro04] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *ACNS*, 2004.

[Gro05] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS*, 2005.

[Gro07] Jens Groth. Fully anonymous group signatures without random oracles. In *Asiacrypt*, 2007.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Eurocrypt*, 2008.

[Kil06] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In *TCC*, 2006.

[Mer89] Ralph C. Merkle. A certified digital signature. In *Crypto*, 1989.

[MRY04] Philip MacKenzie, Michael Reiter, and Ke Yand. Alternatives to non-malleability: definitions, constructions, and applications. In *TCC*, 2004.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, 2008.

[PS08] Franziska Pingel and Sandra Steinbrecher. Multilateral secure cross-community reputation systems for internet communities. In *TrustBus*, 2008.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Eurocrypt*, 1997.

[Ste06] Sandra Steinbrecher. Design options for privacy-respecting reputation systems within centralised internet communities. In *Intl. Information Sec. Conf. (SEC)*, 2006.

[Ste08] Sandra Steinbrecher. Enhancing multilateral security in and by reputation systems. In *FIDIS/IFIP Internet Security and Privacy Summer School*, September 2008.

## A Hardness of SCDH in Generic Groups

The SCDH ("stronger than CDH") assumption may be stated as follows.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map, where the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are of prime order $p$. Let $\varphi : \mathbb{G}_2 \to \mathbb{G}_1$ be an efficiently computable isomorphism. Assume $g_2$ is a generator of $\mathbb{G}_2$ and let $g_1 = \varphi(g_2)$. Let $\rho \in \mathbb{Z}_p^*$. Select $r, s \overset{R}{\longleftarrow} \mathbb{Z}_p \setminus \{-\rho\}$, $h \overset{R}{\longleftarrow} \mathbb{G}_1$, $u, v \overset{R}{\longleftarrow} \mathbb{G}_2$. Then given

$$\rho, g_1, h, g_2, u, v, u^r, v^s, g_1^r, h^s, g_2^{\frac{1}{\rho+r}}, g_2^{\frac{1}{\rho+s}} \quad ,$$

it is computationally infeasible to output a triple $(z, z^r, z^s) \in \mathbb{G}_1^3$ where $z \neq 1$.

We prove this in the generic group model [Sho97] by providing an upper bound on the probability that an adversary is able to output such a triple.

*The generic group formulation of SCDH.* In this model, we assume elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are identified only by random string identifiers. Specifically, we identify the elements of $\mathbb{G}_1$ using an injective map $\xi_1 : \mathbb{Z}_p \to \{0,1\}^k$ selected uniformly at random, where $k$ is sufficiently large that we will assume the adversary cannot guess any valid element identifiers. For any $x \in \mathbb{Z}_p$, the identifier $\xi_1(x)$ represents the element $g_1^x \in \mathbb{G}_1$. The elements of $\mathbb{G}_2$ and $\mathbb{G}_T$ are similarly identified via maps $\xi_2$ and $\xi_3$. To select the random elements $h, u, v$, we select random exponents $x, y_1, y_2$ and let $h = g_1^x$, $u = g_2^{y_1}$, and $v = g_2^{y_2}$; in this way their identifiers may be computed using $\xi_1$ and $\xi_2$.

The adversary will start with the identifiers of the elements in the challenge and must query an oracle $\mathcal{O}$ to compute the group operation in any of the three groups, to evaluate the bilinear map, or to evaluate $\varphi$. Upon termination, it must output three strings $\pi, \pi', \pi'' \in \{0,1\}^k$. If there exists a $z \in \mathbb{Z}_p$ such that $\pi = \xi_1(z)$, $\pi' = \xi_1(zr)$, and $\pi'' = \xi_1(zs)$, then the adversary has won the game.

**Theorem A.1.** *For any adversary $\mathcal{A}$ making at most $q$ queries to the oracle $\mathcal{O}$,*

$$\Pr\left[\begin{array}{c} \mathcal{A}^{\mathcal{O}}\begin{pmatrix} p, \rho, \xi_1(1), \xi_1(x), \xi_1(r), \xi_1(xs), \xi_2(1), \xi_2(y_1), \\ \xi_2(y_2), \xi_2(y_1 r), \xi_2(y_2 s), \xi_2\left(\frac{1}{r+\rho}\right), \xi_2\left(\frac{1}{s+\rho}\right) \end{pmatrix} = \pi, \pi', \pi'' \ \wedge \\ \exists z \in \mathbb{Z}_p \textit{ such that } \pi = \xi_1(z), \pi' = \xi_1(zr), \textit{ and } \pi'' = \xi_1(zs) \end{array} \;\middle|\; \begin{array}{c} x, y_1, y_2 \overset{R}{\longleftarrow} \mathbb{Z}_p \\ r, s \overset{R}{\longleftarrow} \mathbb{Z}_p \setminus \{-\rho\} \end{array}\right] \leq \frac{24(q+11)^2}{p} \quad .$$

*Proof.* Our proof employs the following strategy. First, we define an alternative (the "formal game") to the real game described above. Next, we show that it is impossible for the adversary to win the formal game. Finally, we show that with probability at least $1 - \frac{24(q+11)^2}{p}$, the view of an adversary that played the formal game is identical to what their view would have been if they were playing the real game.

*The formal game.* In this version of the game, the oracle $\mathcal{O}$ ignores the actual values of $x$, $y_1$, $y_2$, $r$, and $s$ and instead treats them as formal variables $X$, $Y_1$, $Y_2$, $R$, and $S$ (the exponent $\rho$ is treated differently, as will be explained).

Specifically, $\mathcal{O}$ maintains three lists $L_1, L_2, L_3$ of pairs. In each pair $(\pi, F)$, $\pi$ is an identifier string and $F$ is a rational function with indeterminates $X, Y_1, Y_2, R, S$. That is, $F$ is a member of the field of rational functions $\mathbb{Z}_p(X, Y_1, Y_2, R, S)$. The elements of the field $\mathbb{Z}_p(X, Y_1, Y_2, R, S)$ may be considered to be (multivariate) polynomial fractions $\frac{P}{Q}$, where $P$ and $Q$ are in the polynomial ring $\mathbb{Z}_p[X, Y_1, Y_2, R, S]$. More precisely, $F$ is an equivalence class of such fractions, where $\frac{P_1}{Q_1} = \frac{P_2}{Q_2}$ if $P_1 Q_2 = P_2 Q_1$. In the following, we identify a rational function $F$ with the representative element of its equivalence class $\frac{P}{Q}$ that is written in lowest terms (i.e., $\gcd(P, Q) = 1$).

Each list is initialized with the identifiers of and the polynomial fractions corresponding to the challenge values in each of the three groups. So initially

$$L_1 = ((\pi_{1,1}, 1), (\pi_{1,2}, X), (\pi_{1,3}, R), (\pi_{1,4}, XS))$$
$$L_2 = \left((\pi_{2,1}, 1), (\pi_{2,2}, Y_1), (\pi_{2,3}, Y_2), (\pi_{2,4}, Y_1R), (\pi_{2,5}, Y_2S), \left(\pi_{2,6}, \frac{1}{R+\rho}\right), \left(\pi_{2,7}, \frac{1}{S+\rho}\right)\right)$$
$$L_3 = () \enspace .$$

Note that only $X$, $Y_1$, $Y_2$, $R$, and $S$ are indeterminates in the above polynomial fractions; $\rho$ is simply a constant in $\mathbb{Z}_p$. Now whenever the adversary makes a query to perform the group operation in $\mathbb{G}_1$ on $\pi_{1,i}$ and $\pi_{1,j}$ (including a selection bit specifying whether they wish to multiply or divide), we look up in list $L_1$ the corresponding polynomial fractions $F_{1,i}, F_{1,j}$ and compute $F = F_{1,i} \pm F_{1,j}$. We check whether $F$ (in a canonical form) already exists in $L_1$ and, if so, return the corresponding identifier. Otherwise, we randomly select a new identifier $\pi$ and append $(\pi, F)$ to $L_1$. Queries for the group operations in $\mathbb{G}_2$ and $\mathbb{G}_T$ are answered analogously. To answer queries for the bilinear map, we compute $F = F_{1,i} \cdot F_{2,j}$ and check $L_3$ for $F$, again, adding it if it was not already present. To answer queries for the isomorphism $\varphi$ applied to some $F_{2,i}$, we simply check $L_1$ for $F_{2,i}$.

We now argue that it is impossible for the adversary to win when $\mathcal{O}$ responds to queries using the rules of the formal game. In order to win, the adversary must construct polynomial fractions $F_{1,i_1}$, $F_{1,i_2}$, and $F_{1,i_3}$ in $L_1$, where $F_{1,i_2} = F_{1,i_1} \cdot R$, $F_{1,i_3} = F_{1,i_1} \cdot S$, and $F_{1,i_1} \neq 0$. However, any $F_{1,*}$ which the adversary can construct in $L_1$ is of the form

$$F_{1,*} = a_1 + a_2X + a_3R + a_4XS + a_5Y_1 + a_6Y_2 + a_7Y_1R + a_8Y_2S + \frac{a_9}{R+\rho} + \frac{a_{10}}{S+\rho} \enspace ,$$

where $a_1, \ldots, a_{10} \in \mathbb{Z}_p$.[4] So if $F_{1,i_2} = F_{1,i_1} \cdot R$, then $F_{1,i_1}$ must be of the form $F_{1,i_1} = a_3 + a_7Y_1$. Similarly, if $F_{1,i_3} = F_{1,i_1} \cdot S$, then $F_{1,i_1}$ must be of the form $F_{1,i_1} = a_4X + a_8Y_2$.

So if the adversary is to output a triple $F_{1,i_1}$, $F_{1,i_2}$, $F_{1,i_3}$ satisfying $F_{1,i_2} = F_{1,i_1} \cdot R$ and $F_{1,i_3} = F_{1,i_1} \cdot S$, then the only possibilities values for $F_{1,i_1}$ are

$$(\{a_3 + a_7Y_1 \mid a_3, a_7 \in \mathbb{Z}_p\} \cap \{a_4X + a_8Y_2 \mid a_4, a_8 \in \mathbb{Z}_p\}) \setminus \{0\} = \varnothing \enspace .$$

Thus, the adversary cannot win when the oracle follows the rules of the formal game.

*The real game.* Next, we argue that, with probability at least $1 - \frac{24(q+11)^2}{p}$, an adversary playing the formal game receives oracle query answers distributed identically to those it would have received in the real game.

We might imagine two ways the formal game could differ from the real game. The first would be for the oracle to give out a previously returned identifier when it should have selected a new one. This would happen if the adversary made a query on two rational functions and the result was formally identical to a previous rational function, but when evaluated on the specific values $x, y_1, y_2, r, s$, the two differed. Of course, this cannot happen. If two rational functions are identical, they will have the same value when evaluated.

The other way the formal game could differ from the real game would be for the oracle to give out a new identifier when it should have given an existing one. That is, if the oracle returned the identifier of a new rational function $F_1 = \frac{P_1}{Q_1}$ which was not formally equal to an existing one $F_2 = \frac{P_2}{Q_2}$ (that is, $P_1Q_2 \neq P_2Q_1$), but $F_1(x, y_1, y_2, r, s) = F_2(x, y_1, y_2, r, s)$.

[4]Note that the adversary is capable of incorporating $\rho$ into the values $a_1, \ldots, a_{10}$, for example, setting $a_1 = 4\rho$ or $a_3 = \rho^{-5}$.

This case is indeed possible, but we argue that it occurs with probability (over the selection of $x, y_1, y_2, r, s$) at most $\frac{24(q+11)^2}{p}$. Specifically, $F_1(x, y_1, y_2, r, s) = F_2(x, y_1, y_2, r, s)$ iff

$$P_1(x, y_1, y_2, r, s)Q_2(x, y_1, y_2, r, s) = P_2(x, y_1, y_2, r, s)Q_1(x, y_1, y_2, r, s) \ ,$$

that is, iff

$$P_1(x, y_1, y_2, r, s)Q_2(x, y_1, y_2, r, s) - P_2(x, y_1, y_2, r, s)Q_1(x, y_1, y_2, r, s) = 0 \ .$$

So we see that the oracle only gives an incorrect reply to a query on $F_1 = \frac{P_1}{Q_1}$ and $F_2 = \frac{P_2}{Q_2}$ if $x, y_1, y_2, r, s$ is a root of the polynomial $P_1Q_2 - P_2Q_1$. We bound the probability of $x, y_1, y_2, r, s$ being a root based on the degree of the polynomial.

Specifically, for any $\frac{P_1}{Q_1}$ and $\frac{P_2}{Q_2}$ in $L_1$,

$$\begin{aligned}\deg(P_1Q_2 - P_2Q_1) &\leq \max(\deg(P_1Q_2), \deg(P_2Q_1)) \\ &= \max(\deg(P_1) + \deg(Q_2), \deg(P_2) + \deg(Q_1)) \\ &\leq \max(4 + 2, 4 + 2) \\ &= 6 \ ,\end{aligned}$$

so $P_1Q_2 - P_2Q_1$ will have at most 6 roots. The probability that a query for the group operation in $\mathbb{G}_1$ will return the wrong result is thus at most $\frac{6}{p}$. The case of queries for the group operation in $\mathbb{G}_2$ is similar and results in the same bound. If $\frac{P_1}{Q_1}$ and $\frac{P_2}{Q_2}$ are in list $L_3$ (i.e., they are the result of pairing queries), we obtain the following bounds.

$$\begin{aligned}\deg(P_1Q_2 - P_2Q_1) &\leq \max(8 + 4, 8 + 4) \\ &= 12 \ ,\end{aligned}$$

So the probability of that type of query being answered incorrectly is at most $\frac{12}{p}$.

Now since the adversary is initially given 11 identifiers and makes at most $q$ queries, the number of distinct queries it can make for either the group operation in $\mathbb{G}_1$, the group operation in $\mathbb{G}_2$, or the pairing is at most $(q + 11)^2$. So the total probability of at least one query being answered incorrectly is at most

$$(q+11)^2\frac{6}{p} + (q+11)^2\frac{6}{p} + (q+11)^2\frac{12}{p} = \frac{24(q+11)^2}{p} \ .$$

□

## B Unforgeability of the Signature Scheme

We distinguish the following four types of forgeries which the adversary may attempt.

- **Type 1 forgery.** In the forgery, the adversary uses a $\rho^*$ value that never appeared. This will break the BB-HSDH assumption by a trivial reduction.
- **Type 2 forgery.** In the forgery, the adversary uses $\rho^* = \rho_j$, but there exists $k$, such that $r_k^* \neq r_{j,k}$.
  - Case 1: $r_k^* = \gamma$.
    This breaks the BB-CDH assumption.
    Suppose the simulator obtains a BB-CDH instance (see Section 4).

The adversary commits to $q$ messages to be signed. The simulator chooses the parameters of the signature scheme, such that the variables $\gamma, g, \widehat{g}$ inherit the corresponding variables from the BB-CDH instance. For all $1 \leq i \leq \ell$, the simulator also chooses $\widehat{u}_i = \widehat{u}^{\tau_i}$, such that it knows their discrete logs $\tau_i$ (base $\widehat{u}$). The remaining parameters are picked directly.

In the $q$ signatures returned to the adversary, the simulator uses $\rho = \rho_1, \ldots, \rho_q$ respectively. Although the simulator does not know the secret signing key $\gamma$, it clearly has enough information to compute the signatures shown to the signature adversary.

When the adversary outputs a Type 2-Case 1 forgery, it contains the term $\widehat{u}_k^{\gamma}$. As the simulator knows the discrete log of $\widehat{u}_k$ base $\widehat{u}$, it can compute $\widehat{u}^{\gamma}$, thereby breaking the BB-CDH assumption.

– Case 2: $r_k^* \in \{r_{j,1}, \ldots, r_{j,\ell}, s_{j,1}, \ldots, s_{j,\ell}\} \backslash \{r_{j,k}\}$.

Similar to Case 1, this also breaks the BB-CDH assumption. In particular, by renaming variables and letting $q = 1$, the BB-CDH assumption immediately implies the following assumption henceforth referred to as BB-CDH-1. Given the tuple

$$g, \widehat{g}, g^r, \widehat{g}^r, \widehat{u}, \rho, \widehat{g}^{\frac{1}{r+\rho}} \quad ,$$

it is computationally infeasible to output $\widehat{u}^r$.

We now show that if an adversary can succeed in a Type 2-Case 2 forgery, we can build a simulator that breaks the above BB-CDH-1 assumption.

The adversary first commits to $q$ messages to be signed. The simulator guesses the $j$ and $k' \neq k$ such that $\rho^* = \rho_j$ and $r_k^* = r_{j,k'}$. The case when $r_k^* = s_{j,k''}$ ($1 \leq k'' \leq \ell$) is similar, so here, without loss of generality, we prove the case for $r_k^* = r_{j,k'}$.

When choosing parameters, the simulator inherits the $g, \widehat{g}$ values from the BB-CDH-1 instance. The simulator lets $\widehat{u}_k = \widehat{u}$. For $i \neq k$, the simulator picks $\widehat{u}_i = \widehat{g}^{\tau_i}$ . The simulator picks $f_1 = x_{j,k'}^{-1} g^{\tau}$ where $\tau \xleftarrow{R} \mathbb{Z}_p$. The simulator picks the remaining parameters directly.

Now the simulator computes the signatures for the $q$ messages specified by the adversary at the beginning of the game. For all except the $j$-th (message, signature) pair, the simulator computes all other signatures directly.

For the $j$-th signature, the simulator builds the $\rho$ value from the BB-CDH-Derived instance into the signature: $\rho_j = \rho$. In addition, it builds the $r$ value into the $k'$-th coordinate, that is, the simulator implicitly lets $r_{j,k'} = r$. Although the simulator does not know $r$, it can compute the term $\widehat{u}_{k'}^r$ as it knows the discrete log of $\widehat{u}_{k'}$ base $\widehat{g}$. In addition, the term $(x_{j,k'} f_1)^r = (g^r)^{\tau}$ can also be computed partly due to the way $f_1$ was chosen earlier. It is clear that the rest of the signature can be computed directly.

If the adversary outputs a forgery of this type, the forged signature contains $\widehat{u}_k^r = \widehat{u}^r$, thereby breaking the BB-CDH-1 assumption.

– Case 3: $r_k^* \notin \{r_{j,1}, \ldots, r_{j,\ell}, s_{j,1}, \ldots, s_{j,\ell}\}$, and and $r^* \neq \gamma$. This breaks the BB-HSDH assumption. In particular, by renaming variables, and letting $q = 2\ell+1$, the BB-HSDH assumption states that given

$$\widehat{g}, \widehat{g}^{\rho}, \widehat{u}, g, g^{\rho}, \gamma, g^{\frac{1}{\rho+\gamma}}, (r_i, \widehat{g}^{\frac{1}{\rho+r_i}})_{1 \leq i \leq \ell}, (s_i, \widehat{g}^{\frac{1}{\rho+s_i}})_{1 \leq i \leq \ell}$$

it is hard to output $(g^{r^*}, \widehat{u}^{r^*}, g^{\frac{1}{\rho+r^*}})$, where $r^* \notin \{r_1, \ldots, r_\ell, s_1, \ldots, s_\ell, \gamma\}$. The simulator obtains this instance, and performs the following interactions with the adversary. The adversary first commits to $q$ messages to be signed. Now the simulator picks the parameters of the signature scheme to inherit the $g, \widehat{g}, \gamma$ variables from the above BB-HSDH instance. It picks $h = g^{\tau_1}$,$\widehat{g}_0 = \widehat{g}^{\tau_2}$, so that the simulator knows the exponents $\tau_1, \tau_2$, and can compute $h^\rho$ and $\widehat{g}_0^\rho$ from $g^\rho$ and $\widehat{g}^\rho$ respectively. For all $1 \leq i \leq \ell$, the simulator picks $\widehat{u}_i = \widehat{u}^{\mu_i}$. The simulator picks the remaining parameters directly.

The simulator guesses the $j$ in which $\rho^* = \rho_j$. In the $j$-th signature returned to the adversary, the simulator uses the $\rho$ and $\{r_i, s_i\}_{1 \leq i \leq \ell}$ values from the above BB-HSDH instance. It is not hard to see that the simulator has sufficient information to compute a signature for the $j$-th message. For all other $q-1$ (message, signature) pairs, the simulator computes their signatures directly.

If the adversary can succeed in a forgery of this case, the simulator can obtain the tuple $(g^{r^*_k}, \widehat{u}_i^{r^*_k} = (\widehat{u}^{r^*_k})^{\mu_k}, \widehat{g}^{\frac{1}{\rho + r^*_k}})$, thereby solving the above BB-HSDH instance.

- **Type 3 forgery**. In the forgery, the adversary uses $\rho^* = \rho_j$, $r^*_j = r_{j,i}$ for all $1 \leq i \leq \ell$, but there exists $k$, such that $s^*_k \neq s_{j,k}$. The proof is similar to Type 2 forgery.

- **Type 4 forgery**. In the forgery, the adversary uses $\rho^* = \rho_j$, $r^*_j = r_{j,i}$ and $s^*_j = s_{j,i}$ for all $1 \leq i \leq \ell$, but there exists a $k$ such that $x^*_k \neq x_{j,k}$. This breaks the SCDH assumption through the following reduction. The simulator is given an SCDH instance (see Section 4), and performs the following interactions with the adversary.

  The adversary first commits to $q$ messages to be signed. The simulator guesses the $j$ in which $\rho^* = \rho_j, \forall i : r^*_i = r_{j,i}$ and $s^*_i = s_{j,i}$. When setting up parameters of the signature scheme, the simulator inherits the $\widehat{g}, g, h$ value from the above SCDH instance. For all $1 \leq i \leq \ell$, it lets $\widehat{u}_i = \widehat{u}^{\mu_i}, \widehat{v}_i = \widehat{v}^{\nu_i}$. The simulator also lets $f_1 = x_{j,k}^{-1} g^\tau$, and $f_2 = x_{j,k}^{-1} h^\omega$where $\tau, \omega \xleftarrow{R} \mathbb{Z}_p$.

  Now the simulator constructs signatures on the $q$ specified messages and return them to the adversary. Except for the $j$-th (message, signature) pair, the simulator constructs all other (message, signature) pairs directly.

  For the $j$-th signature, the simulator uses the following strategy. It builds the $\rho$ value from the SCDH instance into the $j$-th signature, that is $\rho_j = \rho$. In addition, it builds the $r, s$ values from the SCDH instance into the $k$-th coordinate, that is, $r_{j,k} = r, s_{j,k} = s$. Although the simulator does not know the values of $r, s$, it knows or can compute all of the following terms in the signature. In particular, $(x_{j,k} f_1)^r = (g^r)^\tau$, $(x_{j,k} f_2)^s = (h^s)^\omega$. $\widehat{u}_k^r$ and $\widehat{v}_k^s$ can be computed as the simulator knows their discrete logs base $\widehat{u}$ and $\widehat{v}$ respectively. All the remaining terms are trivially computable.

  If the adversary success in a Type 3 forgery, the resulting signature contains $(x^*, (x^* f_1)^r, (x^* f_2)^s)$ where $x^* \neq x_{j,k}$. The simulator can thereby compute $z, z^r, z^s$, where $z = x^* x_{j,k}^{-1} \neq 1$, by dividing $(x^*, (x^* f_1)^r, (x^* f_2)^s)$ and $(x_{j,k}, (x_{j,k} f_1)^r, (x_{j,k} f_2)^s)$ coordinate-wise. This clearly breaks the SCDH assumption.

## C IK-CPA Security: Definition and Proof

IK-CPA security was first defined by Bellare et. al. [BBDP01]. The IKEnc described in Section 4 has IK-CPA security, that is, no polynomial-time adversary has more than negligible advantage in the following game:

*Setup.* The challenger returns to the adversary the public parameters of the encryption system $\mathsf{params}_{\text{ike}}$, and two user public keys $\mathsf{upk}_{\text{ike},0}$ and $\mathsf{upk}_{\text{ike},1}$.

*Challenge.* The adversary submits two messages $\mathsf{msg}_0$ and $\mathsf{msg}_1$. The challenger flips a random coin $b$, and returns $\text{IKEnc.Enc}(\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike},b}, \mathsf{msg}_b)$ to the adversary.

*Guess.* The adversary outputs a guess $b'$ of $b$. The adversary wins the game if $b' = b$.

**Remark 1.** *The above security definition should still hold when* $\mathsf{upk}_{ike,0} = \mathsf{upk}_{ike,1} = \mathsf{upk}_{ike}$. *In this case, the security definition is equivalent to the standard IND-CPA security (under a specific user public key* $\mathsf{upk}_{ike}$*).*

*Proof.* Consider the following hybrid sequence. In Game 0, the challenger encrypts $\mathsf{msg}_0$ under $\mathsf{upk}_{\text{ike},0}$ in the challenge stage. In Game R, the challenger returns to the adversary a random ciphertext in the challenge stage, that is, $(R_1, R_2, R_3) \xleftarrow{R} \mathbb{G}^3$. In Game 1, the challenger encrypts $\mathsf{msg}_1$ under $\mathsf{upk}_{\text{ike},1}$ in the challenge stage.

Below we prove that Game 0 is computationally indistinguishable from Game M. (The indistinguishability between Game 1 and Game M is similar, and hence omitted.)

Suppose a simulator obtains the following DLinear instance:

$$f, h, A, B \xleftarrow{R} \mathbb{G}, \quad f^r, h^s, T$$

It tries to distinguish whether $T \xleftarrow{R} \mathbb{G}$ or $T = A^r B^s$. See Definition 8 for more details on this DLinear variant.

Now the simulator sets up the public parameters of the encryption scheme to be $f, h$. It chooses $\mathsf{upk}_{\text{ike},0} = (A, B)$, and it picks $(\mathsf{upk}_{\text{ike},1}, \mathsf{usk}_{\text{ike},1})$ by directly calling the IKEnc.GenKey algorithm. In the challenge stage, the adversary submits two messages $\mathsf{msg}_0$ and $\mathsf{msg}_1$. The simulator returns the following ciphertext to the adversary:

$$\mathsf{msg}_0 \cdot T, f^r, h^s$$

It is not hard to see that if $T = A^r B^s$, then the above simulation is identical to Game 0. Otherwise, it is identical to Game R. □

## D Security Definitions and Proofs for the Unblinded Scheme

### D.1 Definitions

*Voter anonymity.* No polynomial-time adversary has more than negligible advantage in the following game.

**Setup**. The challenger gives the adversary all users' $\mathsf{rcvkey}$ and $\mathsf{vpk}$. At this stage, the challenger retains all $\mathsf{vsk}$ to itself.

**Corrupt**. The adversary adaptively corrupts a user by learning its secret voting key $\mathsf{vsk}$.

**Vote**. The adversary requests an unblinded vote from an uncorrupted voter to a recipient.

**Challenge**. The adversary submits two uncorrupted voters $j_0^*$ and $j_1^*$, and a recipient $i$. The adversary must not have previously queried a vote from either $j_0^*$ or $j_1^*$ to $i$. The challenger flips a random coin $b$, and returns an unblinded vote from $j_b^*$ to $i$.

**ShowRep**. The adversary specifies a signer $i$, and a list of voters $j_1, \ldots, j_c$, and signer $i$ constructs rep based on votes from these voters. Notice that this may involve votes from $j_0^*$ or $j_1^*$ to $i$. rep is returned to the adversary.

**Guess**. The adversary outputs a guess $b'$ of $b$, and wins the game if $b' = b$.

*Vote unforgeability.* No polynomial-time adversary has more than negligible advantage in the following game.

**Setup**. The challenger gives the adversary all users' rcvkey and vpk.

**Corrupt**. The adversary adaptively corrupts a user by learning its vsk.

**Vote**. The adversary requests a vote from an uncorrupted voter to a recipient.

**ShowRep**. The adversary specifies a signer $i$, and a list of voters $j_1, \ldots, j_c$. User $i$ constructs rep based on votes from these voters, and returns it to the adversary.

**Forge**. The adversary outputs a vote from an uncorrupted user $j^*$ to a recipient $i^*$. The adversary wins if the vote is correct, and it has not previously queried a vote from $j^*$ to $i^*$.

*Reputation anonymity.* No polynomial-time adversary has more than negligible advantage in the following game.

**Setup**. The challenger generates $n$ users, and reveal all users' keys including rcvkey, votekey to the adversary.

**Challenge**. The adversary chooses a user $i^*$, and a list of $c$ voters $j_1, \ldots, j_c$. The challenger flips a random coin $b$, and depending on the value of $b$, it returns to the adversary either faithfully constructed rep, or a list of random numbers.

**Guess**. The adversary outputs a guess $b'$ of $b$, and wins the game if $b' = b$.

## D.2 Reputation Anonymity Proof

We prove the DLinear based instantiation.

**Definition 8** ($n$-DLinear)**.** *Given* $(g, h, z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, \ldots, z_{n,1}, z_{n,2}) \xleftarrow{R} \mathbb{G}^{2n+2}$*, and* $g^r, h^s$*, where* $r, s \xleftarrow{R} \mathbb{Z}_p$*, it is computationally infeasible to distinguish the following tuple from a completely random tuple:*

$$T = (z_{1,1}^r z_{1,2}^s,\ z_{2,1}^r z_{2,2}^s,\ \ldots,\ z_{n,1}^r z_{n,2}^s)$$

*Proof.* We now prove that the $n$-DLinear assumption is implied by the DLinear assumption. Let $0 \le d \le n$, let $\Gamma_i = z_{i,1}^r z_{i,2}^s$. Define a hybrid sequence: in the Game $d$ ($0 \le d \le n$), the challenger gives the adversary $g, h, g^r, h^s, z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, \ldots, z_{n,1}, z_{n,2}$, and the following tuple:

$$*, *, \ldots, *, \Gamma_{d+1}, \ldots, \Gamma_n$$

where each $*$ denotes an independent random element from $\mathbb{G}$.

Due to the hybrid argument, it suffices to show that no PPT adversary can distinguish any two adjacent games.

We now show that no PPT adversary can distinguish between Game $d$ and Game $d-1$, where $1 \le d \le n$. Suppose a simulator gets a DLinear instance $g, h, f, g^r, h^s, X$, it tries to tell whether $X = f^{r+s}$ or $X \xleftarrow{R} \mathbb{G}$. It picks $z_{d,1} = f, z_{d,2} = fh^{\tau}$. For all $i \neq d$, the simulator picks $z_{i,1} = g^{\omega_i}, z_{i,2} = h^{\mu_i}$. Now the simulator gives the adversary $g, h, g^r, h^s, z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, \ldots, z_{n,1}, z_{n,2}$, and the following tuple:

$$*, *, \ldots, *, X \cdot (h^s)^{\tau}, \quad (g^r)^{\omega_{d+1}}(h^s)^{\mu_{d+1}}, \ldots, (g^r)^{\omega_n}(h^s)^{\mu_n}$$

Clearly, the above game is equivalent to Game $d-1$ if $X = f^{r+s}$. Otherwise, it is equivalent to Game $d$. □

Now we build a simulator that leverages an adversary against reputation anonymity to break the $n$-DLinear assumption. When choosing all users' upk and usk values, the simulator inherits the $z_{i,k}$ values from the $n$-DLinear assumption, where $1 \leq i \leq n, k \in \{1,2\}$. It picks $x_{i,1} = g^{\tau_{i,1}}, y_{i,1} = g^{\omega_{i,1}}$, and $x_{i,2} = h^{\tau_{i,2}}, y_{i,2} = h^{\omega_{i,2}}$ for all $1 \leq i \leq n$. It picks all other parameters in upk and usk directly.

In the challenge phase, the simulator computes rep as below:

$$\forall 1 \leq j \leq m : u_{j,1}^r u_{j,2}^s = (x_{i,1}^{\alpha_j} y_{i,1}^{\beta_j})^r (x_{i,2}^{\alpha_j} y_{i,2}^{\beta_j})^s \cdot T_j$$

where $T_j$ is inherited from the $n$-DLinear instance. As the simulator knows the discrete-log of $x_{i,1}$ and $y_{i,1}$ base $g$, and the discrete-log of $x_{i,2}$ and $y_{i,2}$ base $h$, the simulator can compute the term $(x_{i,1}^{\alpha_j} y_{i,1}^{\beta_j})^r (x_{i,2}^{\alpha_j} y_{i,2}^{\beta_j})^s$.

It is not hard to see that if $T$ is a true $n$-DLinear tuple, the above constructed rep is faithful. Otherwise, it is a random tuple.

## D.3 Voter Anonymity Proof

*Game 1: answering ShowRep queries at random.* First, modify the voter anonymity game such that when the adversary makes ShowRep queries, the challenger simply returns a list of random numbers. Answering ShowRep queries randomly does not affect the adversary's advantage in winning the voter anonymity game.

To see why, observe that it is computationally infeasible to distinguish between Game 1 and the real voter anonymity game. This can concluded from reputation anonymity and a simple hybrid argument.

*Reduction to the DLinear assumption.* We can now reduce voter anonymity to the DLinear assumption.

Below, we prove the real-or-random version of voter anonymity.

Notice that DLinear implies that the following problem is hard: Given

$$g, h, g^{\alpha}, h^{\beta}, u_1, v_1, u_2, v_2, T$$

it is computationally infeasible to distinguish whether $T = (u_1^{\alpha} v_1^{\beta}, u_2^{\alpha} v_2^{\beta})$ or $T \xleftarrow{R} \mathbb{G}^2$. In fact, this is a special case of the $n$-DLinear assumption mentioned above, with $n = 2$.

- **Setup**. The simulator obtains the above 2-DLinear instance. It inherits the parameters $g, h$ from the 2-DLinear instance. It guesses the challenge voter $j^*$ and recipient $i^*$. It picks $x_{i^*,k} = u_k, y_{i^*,k} = v_k$ for $k \in \{1,2\}$. For all $i \neq i^*$, pick $x_{i,k} = g^{\tau_{i,k}}, y_{i,k} = h^{\omega_{i,k}}$ for $k \in \{1,2\}$. In addition, the simulator lets $\alpha_{j^*} = \alpha$, $\beta_{j^*} = \beta$. The remaining parameters are picked directly. It is not hard to see that the simulator can compute all users vpk and rcvkey, which the simulator releases to the adversary at the beginning of the game.

- **Corrupt**. If the targeted user is $j^*$, abort. Otherwise, return to the adversary the user's $\mathsf{vsk}$.
- **Vote**. If the vote queried is from $j^*$ to $i^*$, abort. Otherwise, if the voter is $j^*$ and the recipient is $i \neq i^*$, compute the vote as follows:

$$[(g^{\alpha})^{\tau_{i,k}}(h^{\beta})^{\omega_{i,k}} z_{j^*,k}]_{k\in\{1,2\}}$$

  Else if the voter is not $j^*$, compute the vote directly.
- **ShowRep**. Return a random tuple.
- **Challenge**. If the challenger voter and recipient are not $j^*$ and $i^*$, abort. Otherwise, return the following vote:

$$(T_1 z_{j^*,1}, T_2 z_{j^*,2})$$

  Clearly, if $T = (T_1, T_2)$ is a true 2-DLinear instance, the above vote is correctly constructed. Otherwise, it is a random pair.

## D.4 Vote Unforgeability Proof

- **Setup**. The simulator obtains a CDH instance $g, g^{\alpha}, h$. It tries to output $h^{\alpha}$.

  The simulator guesses the voter $j^*$ and recipient $i^*$ in the forged vote output by the adversary. It implicitly lets $\alpha_{j^*} = \alpha$. It picks $x_{i^*,k} = h^{\tau_{i^*,k}}$ where $k \in \{1,2\}$. For any user $i \neq i^*$, the simulator picks $x_{i,k} = g^{\tau_{i,k}}$ where $k \in \{1,2\}$. The simulator picks the remaining parameters directly. It is not hard to see that the simulator can compute all users $\mathsf{vpk}$ and $\mathsf{rcvkey}$, which the simulator releases to the adversary at the beginning of the game.
- **Corrupt**. If the targeted user is $j^*$, abort. Otherwise, give away the user's $\mathsf{vsk}$ to the adversary.
- **Vote**. If the requested vote is from $j^*$ to $i^*$, abort. If the requested vote is from $j^*$ to $i \neq i^*$, the simulator computes the vote as follows:

$$[(g^{\alpha})^{\tau_{i,k}} y_{i,k}^{\beta_{j^*}} z_{j^*}]_{k\in\{1,2\}}$$

  If the requested vote is from $j \neq j^*$ to any user $i$, the simulator computes the vote directly.
- **ShowRep**. Return a random tuple. Like in the voter anonymity game, this change should not affect the adversary's probability in winning the vote unforgeability game.
- **Forge**. When the adversary outputs a vote from $j^*$ to $i^*$, the simulator can compute $h^{\alpha}$ as below. First, denote the vote as $(u_1, u_2)$. Now, compute $h^{\alpha}$ as below:

$$(u_1 z_{j^*,1}^{-1} y_{i^*,1}^{-\beta_{j^*}})^{\frac{1}{\tau_{i^*,1}}}$$

## E Security Proofs for the Full Scheme

**Theorem E.1.** *The algorithms* $\textsc{Setup}$*,* $\textsc{GenCred}$*,* $\textsc{GenNym}$*,* $\textsc{Vote}$*,* $\textsc{SignRep}$*, and* $\textsc{VerifyRep}$ *defined in Section 5 constitute a* correct*,* receiver anonymous*,* voter anonymous*,* signer anonymous*, and* sound *scheme for signatures of reputation.*

In this section, we provide proofs for each of the four security properties defined in Section 3. We begin by proving a series of lemmas we will need.

In the three privacy games (receiver, voter, and signer anonymity), the adversary outputs a challenge, which varies from game to game, and challenger responds based on a coin flip $b$. In the proofs in this section, it will be convenient to refer to each of these as an oracle query the adversary can make, so we define the following additional oracle queries which correspond to the challenge stage of each game:

*Ch_RecvAnon.* On input $(i_0^*, i_1^*)$, select $b \xleftarrow{R} \{0,1\}$ and respond with $\mathsf{nym}^* \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_{i_b^*})$.

*Ch_SignerAnon.* On input $(j_0^*, j_1^*, \mathsf{nym}^*)$, select $b \xleftarrow{R} \{0,1\}$ and respond with $\mathsf{vt}^* \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_{j_b^*}, \mathsf{nym}^*)$.

*Ch_VoterAnon.* On input $(i_0^*, i_1^*, V_0^*, V_1^*, \mathsf{msg})$, select $b \xleftarrow{R} \{0,1\}$ and respond with $\Sigma_b^* \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_{i_b^*}, V_b^*, \mathsf{msg})$.

We now go to prove the first lemma we will need.

### E.1 Traceability

Intuitively, traceability means that all nyms, votes and signatures of reputation must be traceable to registered recipients and voters.

In the following, we refer to an additional opening algorithm $\textsc{OpenSigRep}$ which works exactly like $\textsc{OpenNym}$ and $\textsc{OpenVote}$. That is, it uses the extractor key $\mathsf{xk}$ to obtain the $\mathsf{rcvkey}$ of the signer from the commitment in the NIZK within the signature.

**Lemma E.2.** *No polynomial-time adversary has more than negligible advantage in the following game.*

*Setup*: The challenger runs the $\textsc{Setup}$ algorithm, registers $n$ users, and returns $\mathsf{params}$ and all users' credentials to the adversary.

*Forge*: The adversary wins the game if one of the following occurs:

- The adversary outputs a valid $\mathsf{nym}^*$ and $\textsc{OpenNym}(\mathsf{params}, \mathsf{openkey}, \mathsf{nym}^*) = \perp$.
- The adversary outputs a valid vote $\mathsf{vt}^*$ and $\textsc{OpenVote}(\mathsf{params}, \mathsf{openkey}, \mathsf{vt}^*) = \perp$.
- The adversary outputs a valid signature of reputation $\Sigma^*$ and $\textsc{OpenSigRep}(\mathsf{params}, \mathsf{openkey}, \Sigma^*) = \perp$.

In the above, "valid" means that the $\mathsf{nym}^*$, $\mathsf{vt}^*$ or $\Sigma^*$ passes the corresponding verification algorithm.

We also refer to the above as nym traceability, vote traceability, signature of reputation traceability respectively.

*Proof.* We prove the case for signature of reputation traceability. The other cases are similar if not easier. There are 2 possible cases if $\textsc{OpenSigRep}(\mathsf{params}, \mathsf{openkey}, \Sigma^*) = \perp$:

- Case 1: The OpenSigRep algorithm uses the extractor key $\mathsf{xk}$ of the GS proof system to extract the signer's $pub\_cred = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}})$, and a certificate on the above tuple. The $pub\_cred$ extracted is not among the registered users.

- Case 2: Use $\mathsf{xk}$ to extract a list of votes, and now further use OpenVote on these votes to extract a set $S$ of $c$ distinct voters, more specifically, for each $j \in S$, extract $pub\_cred_j = (\mathsf{rcvkey}_j, \mathsf{vpk}_j, \mathsf{vk}_{\mathrm{bb}j}, \mathsf{upk}_{\mathrm{ike}j})$ and a certificate for each $pub\_cred_j$. There exists a $j \in S$ such that $pub\_cred_j$ is not among the registered users.

If either of the above cases is true, we can build a simulator that breaks the security of the certification scheme, or more specifically, the existential unforgeability under the weak chosen message attack (henceforth referred to as weak EF-CMA security).

At the beginning of the game, the simulator picks $pub\_cred_i = (\mathsf{rcvkey}_i, \mathsf{vpk}_i, \mathsf{vk}_{\mathrm{bb},i}, \mathsf{upk}_{\mathrm{ike},i})$ for all $1 \leq i \leq n$, and the corresponding secret keys $\mathsf{vsk}_i, \mathsf{sk}_{\mathrm{bb},i}, \mathsf{usk}_{\mathrm{ike},i}$. The simulator submits all $pub\_cred_i$ $(1 \leq i \leq n)$ to the weak EF-CMA challenger $\mathcal{C}$. The EF-CMA challenger now returns to the simulator the public verification key $\mathsf{vk}_{\mathrm{cert}}$ of the certification scheme, as well as $n$ certificates on the submitted messages. With this, the simulator has chosen the $\mathsf{vk}_{\mathrm{cert}}$ for our reputation system, as well as the user credentials for $n$ users. The simulator picks the other required system parameters directly.

The simulator now releases all users' secret credentials to a traceability adversary, which outputs a forgery consisting of a signature of reputation $\Sigma^*$. No matter which of the above case is true, the simulator is able to extract a $pub\_cred$ that does not match any registered user, and a certificate for $pub\_cred$. In this way, the simulator has forged a certificate on a new message, thereby breaking the weak EF-CMA security of the certification scheme. □

## E.2 Non-frameability

**Lemma E.3.** *No polynomial-time adversary has more than negligible advantage in the following game.*

*Setup*: The challenger runs the Setup algorithm, registers $n$ users, and returns $\mathsf{params}$ to the adversary.

*Query*: The adversary adaptively makes *Corrupt*, *Nym*, *Vote* and *SignRep* queries to the challenger. The adversary can also make any of the challenge queries, including *Ch_RecvAnon*, *Ch_SignerAnon*, *Ch_VoterAnon* queries.

*Forge*: The adversary wins the game if it succeeds in one of the following forgeries:

- *Nym forgery.* The adversary outputs a forged $\mathsf{nym}^*$. $\mathsf{nym}^*$ has not been returned to the adversary by a previous *Nym* (or *Ch_RecvAnon*) query. In addition, OpenNym$(\mathsf{params}, \mathsf{openkey}, \mathsf{nym}^*) = i^*$, and $i^*$ is not among those corrupted by the adversary through a *Corrupt* query.
- *Vote forgery.* The adversary outputs a forged vote $\mathsf{vt}^*$. $\mathsf{vt}^*$ has not been returned to the adversary by a previous *Vote* (or *Ch_VoterAnon*) query, and $\mathsf{vt}^*$ opens to a voter $j^*$ who has not been compromised by the adversary through a *Corrupt* query.
- *Signature of reputation forgery.* The adversary outputs a forged signature of reputation $\Sigma^*$. $\Sigma^*$ has not been returned to the adversary by a previous *SignRep* (or *Ch_SignerAnon*) query, and $\Sigma^*$ opens to a signer $i^*$ who has not been compromised by the adversary through a *Corrupt* query.

*Proof.* We prove the case for nym non-frameability. The proofs for vote non-frameability and signature of reputation non-frameability are similar.

First, notice that the $(\mathsf{sk_{ots}}^*, \mathsf{vk_{ots}}^*)$ used in $\mathsf{nym}^*$ must agree one of those previously seen by the adversary in a *Nym*, *Vote* or *SignRep* query on the same user $i^*$. Otherwise, we can build a simulator that breaks the security of the BB-signature scheme, specifically, the existential unforgeability under the weak chosen message attack (henceforth referred to as weak EF-CMA security.) Groth used a similar argument in his group signature scheme [Gro07]. Below we describe this reduction in detail.

The simulator guesses $i^*$ at the beginning of the game. If the guess turns out to be wrong later in the game, the simulator aborts. The simulator guesses correctly with probability at least $1/n$, where $n$ denotes the total number of registered users.

The simulator obtains a verification key $V = g^s \in \mathbb{G}$ from the BB signature challenger. The simulator picks user $i^*$'s verification key $\mathsf{vk}_{\mathrm{bb},i^*}$ to be $V$. Notice that the simulator does not know the corresponding signing key $\mathsf{vk}_{\mathrm{bb},i^*} = s$. The simulator picks the other elements of user $i^*$'s credential directly, and signs a certificate for it. The simulator need not know $\mathsf{vk}_{\mathrm{bb},i^*} = s$ to produce the certificate, as the certificate signs $\mathsf{vk}_{\mathrm{bb},i^*} = V$ rather than the secret signing key $s$.

The simulator chooses $q$ random $(\mathsf{sk}_{\mathrm{ots},1}, \mathsf{vk}_{\mathrm{ots},1}), \ldots, (\mathsf{sk}_{\mathrm{ots},q}, \mathsf{vk}_{\mathrm{ots},q})$ pairs, and queries the BB signature challenger for signatures on $H(\mathsf{vk}_{\mathrm{ots},1}), \ldots, H(\mathsf{vk}_{\mathrm{ots},q})$. Whenever the adversary makes a *Nym*, *Vote*, or *SignRep* query, the simulator consumes one of these $(\mathsf{sk}_{\mathrm{ots},i}, \mathsf{vk}_{\mathrm{ots},i})$ where $1 \le i \le q$.

When the adversary outputs a forged $\mathsf{nym}^*$ with a $\mathsf{vk}^*_{\mathrm{ots}}$ never seen before, the simulator uses, the extractor key to open the NIZK, and obtains a new pair $(H(\mathsf{vk}^*_{\mathrm{ots}}), \mathrm{BBSig.Sign}(H(\mathsf{vk}^*_{\mathrm{ots}})))$. Due to the collision resistance of the hash function, $H(\mathsf{vk}^*_{\mathrm{ots}}) \notin \{H(\mathsf{vk}_{\mathrm{ots},1}), \ldots, H(\mathsf{vk}_{\mathrm{ots},q})\}$ (except with negligible probability). This breaks the weak EF-CMA security of the BB signature scheme.

As the $\mathsf{vk}^*_{\mathrm{ots}}$ used in $\mathsf{nym}^*$ agrees with one seen before (in a *Nym*, *Vote*, or *SignRep* query from $i^*$), $\mathsf{nym}^*$ must agree with a previously seen $\mathsf{nym}^*$ from user $i^*$. ( $\mathsf{nym}^*$ cannot agree with a previously seen vote or signature of reputation from $i^*$.) Otherwise, the $\mathsf{nym}^*$ would contain a one-time signature signed with $\mathsf{sk}^*_{\mathrm{ots}}$ on a new message, where the message contains all of $\mathsf{nym}^*$ except the one-time signature part. This breaks the security of the one-time signature scheme through a simple reduction. □

## E.3 Unforgeability

**Lemma E.4.** *No polynomial-time adversary has more than negligible advantage in the following game.*

*Setup.* The challenger sets up system parameters, registers $n$ users, and returns $\mathsf{params}$ to the adversary.

*Query.* The adversary adaptively makes *Corrupt*, *Nym*, *Vote*, *SignRep* queries.

*Forge.* The adversary wins the game if it succeeds in either of the following types of forgeries:

- *Vote forgery.* The adversary outputs a vote $\mathsf{vt}^*$ such that $\mathrm{OpenVote}(\mathsf{params}, \mathsf{openkey}, \mathsf{vt}^*) = (j, i)$, where $j$ has not been corrupted through a *Corrupt* query, and the adversary has not previously submitted a *Vote* query from user $j$ to any $\mathsf{nym}$ that opens to $i$.
- *Signature of reputation forgery.* The adversary outputs a signature of reputation $\Sigma^*$. Suppose $\mathrm{OpenSigRep}$ opens $\Sigma^*$ to the recipient $i$ and $c$ voters

$j_1, \ldots, j_c$. There exists $j \in \{j_1, \ldots, j_c\}$ such that $j$ has not been corrupted through a *Corrupt* query, and the adversary has not previously submitted a *Vote* query from user $j$ to a $\mathsf{nym}$ which opens to $i$.

*Proof.* By reduction to vote unforgeability of the unblinded scheme. We will perform the simulation under a simulated crs. This means that we can no longer rely on the extractor key $\mathsf{xk}$ to open the NIZK. However, notice that we can also implement the open algorithms by decrypting the ciphertexts in the nyms, votes, and signatures of reputation. Notice that under a real crs, opening using $\mathsf{xk}$ or through decryption yield the same result due to the perfect soundness of NIZK.

*Setup*: The simulator chooses a simulated crs instead of a real crs, and it knows the simulation secret $\mathsf{simkey}$. The simulator obtains all users' $\mathsf{rcvkey}$ and $\mathsf{vpk}$ from $\mathcal{C}$, the vote unforgeability challenger of the unblinded scheme. The simulator sets up the parameters of CCAEnc such that it knows the decryption key. The simulator picks all other system parameters directly. Notice that the simulator knows the $\mathsf{usk}_{\text{ike}}$ for all users.

*Corrupt*: The adversary specifies a user $i$ to corrupt. The simulator forwards the query to $\mathcal{C}$, and obtains the user's $\mathsf{vsk}$ in return. The simulator returns the credential of user $i$ to the adversary.

*Nym*, *SignRep*: It is not hard to see that the simulator can answer *Nym* and *SignRep* queries normally.

*Vote*: As the simulator has all users' $\mathsf{usk}_{\text{ike}}$, it is able to decrypt the ciphertext in the specified $\mathsf{nym}$, and identify the recipient $i$. See Appendix E.4 for more details on how this step can be achieved.

Now the simulator forwards the voter $j$ and the recipient $i$ to $\mathcal{C}$, and obtains an unblinded vote from $j$ to $i$. To compute the vote, the simulator encrypts the unblinded vote under IKEnc to obtain the term $C_1$. Then it computes $C_2$ normally. It uses the simulation secret $\mathsf{simkey}$ to compute the NIZK, and eventually, uses the one-time signature scheme to sign everything. It is not hard to see that a vote computed in this way is identically distributed as a real vote under a simulated crs.

*Forge*: Eventually, the adversary outputs a forgery. If the forgery is a vote $\mathsf{vt}^*$, the simulator decrypts the IKEnc ciphertext in $\mathsf{vt}^*$ to obtain an unblinded vote $U^*$. Otherwise, if the forgery is a signature of reputation $\Sigma^*$, the simulator decrypts the CCAEnc ciphertext in $\Sigma^*$ to obtain a list of unblinded votes, among which is $U^*$. If the adversary wins the vote unforgeability game, then $U^*$ is from an uncorrupted voter $j$ to a recipient $i$, and the adversary has never made a *Vote* query from $j$ to a nym corresponding to $i$. This means that our simulator has broken the vote unforgeability of the unblinded scheme.

□

## E.4 Alternative implementation of OpenNym

In all of the games, when the adversary makes a SignRep query (or a *Ch_SignerAnon* query), the challenger needs to check if the set of votes supplied by the adversary correspond to the recipient specified by the adversary. To do this, the challenger calls the OpenNym algorithm to trace the owners of the nyms that are included in the votes.

We now propose an alternative implementation of the OpenNym oracle.

First, we define a sub-routine called TestNym to test if a $\mathsf{nym}$ belongs to a specific user.

TestNym($\mathsf{params}$, $\mathsf{nym}$, $\mathsf{cred}_i$). The TestNym subroutine checks if a $\mathsf{nym}$ is owned by user $i$. Parse $\mathsf{nym}$ as $\mathsf{nym} = (\mathsf{vk}_{\text{ots}}, C, \Pi, \sigma_{\text{ots}})$, where $C = (C_0, C_1, C_1')$ consists of an encryption of user $i$'s receiver key $\mathsf{rcvkey}_i$ (denoted $C_0$) and two random encryptions of $1 \in \mathbb{G}$ (denoted $C_1$ and $C_1'$). Let $\mathsf{usk}_{\text{ike},i}$ denote the secret decryption key of user $i$.

The TestNym algorithm first tests if the following equations are true:

$$\text{IKEnc.Dec}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike},i}, C) = \mathsf{rcvkey}_i$$
$$\text{IKEnc.Dec}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike},i}, C_1) = 1$$
$$\text{IKEnc.Dec}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike},i}, C_1') = 1$$

Next, The TestNym algorithm checks that the two encryptions of 1 are uncorrelated in the following sense. Let $C_1 = (c_1, c_2, c_3)$, and let $C_1' = (c_1', c_2', c_3')$. The algorithm makes sure that

$$e(c_2, c_3') \neq e(c_2', c_3) \ . \tag{1}$$

If both of the above checks pass, the algorithm concludes that the $\mathsf{nym}$'s owner is user $i$.

**Lemma E.5.** *If* $\mathsf{nym}$ *is generated by user $i$, then* TestNym*(*$\mathsf{params}$*,* $\mathsf{nym}$*,* $\mathsf{cred}_i$*) returns 1 (except negligible probability). In addition, there exists at most one $i \in \{1..n\}$ such that* TestNym$(\mathsf{nym}, \mathsf{cred}_i) = 1$.

*Proof.* The first direction is obvious: if $\mathsf{nym}$ is generated by user $i$, clearly, TestNym($\mathsf{params}$, $\mathsf{nym}$, $\mathsf{cred}_i$) returns 1 (except negligible probability).

For the other direction, assume for the sake of contradiction that there exist 2 users $i \neq j \in \{1..n\}$ such that TestNym$(\mathsf{nym}, \mathsf{cred}_i) = 1$, and TestNym$(\mathsf{nym}, \mathsf{cred}_j) = 1$. This means that there exist $\mathsf{upk}_{\text{ike},i} = (A_i, B_i) \neq \mathsf{upk}_{\text{ike},j} = (A_j, B_j)$, and $r_1, s_1, r_2, s_2 \in \mathbb{Z}_p$, and

$$E_{\mathsf{upk}_{\text{ike}},i}(1, r_1, s_1) = E_{\mathsf{upk}_{\text{ike}},j}(1, r_2, s_2)$$
$$E_{\mathsf{upk}_{\text{ike}},i}(1, r_1', s_1') = E_{\mathsf{upk}_{\text{ike}},j}(1, r_2', s_2')$$

Clearly, for the latter two terms in the ciphertext to be equal, $r_1 = r_2 = r$, and $s_1 = s_2 = s$. Similarly, $r_1' = r_2' = r'$, and $s_1' = s_2' = s'$. For the first term in the ciphertext to be equal, we obtain through basic algebra:

$$A_i^r B_i^s = A_j^r B_j^s \quad \Rightarrow \quad (A_i/A_j)^r = (B_j/B_i)^s$$

Similarly,

$$(A_i/A_j)^{r'} = (B_j/B_i)^{s'}$$

But this would break the second check in the TestNym algorithm (see Equation (1) ). □

We now describe an alternative implementation of the OpenNym algorithm which the simulator will use in the simulations. Let $S_c$ denote the set of users that have been corrupted by the adversary thus far. Let $L = \{\mathsf{nym}_k, id_k\}_{1 \leq k \leq q}$ denote the list of $\mathsf{nym}$s that have been returned to the adversary through a *Nym* query (or a *Ch_RecvAnon* query). $\mathsf{nym}_k$ is the $\mathsf{nym}$ returned to the adversary, and $id_k$ denotes the id of the user specified in the query. (In the case of a *Ch_RecvAnon*, $id_k$ should be the one of the two users specified in the query, depending on the challenger's coin.)

```
OpenNym :
Step 1:  for $i \in S_c$ :   if $\textsc{TestNym}(\mathsf{nym}, \mathsf{cred}_i) = 1$ then return $i$;
Step 2:  if $\mathsf{nym} \in L$ :   return the corresponding id
Step 3:  return $\perp$
```

**Remark 2.** *This means that the adversary essentially has enough information to perform* OpenNym *itself, on any valid nym it is able to construct.*

**Lemma E.6.** *This alternative* OpenNym *procedure is correct in the real crs world, except with negligible probability.*

*Proof.* Due to $\mathsf{nym}$ traceability and unforgeability, except with negligible probability, either 1) the nym agrees with one previously seen by the adversary; or 2) the $\mathsf{nym}$ opens to a user within the adversary's coalition (where the open operation is performed using the extractor key.) Due to the perfect soundness of NIZK, for the second case, using the extractor key or TestNym to open the nym would produce the same result. ☐

Notice that the above alternative OpenNym implementation works in all games, even though each game may have a different type of challenge query.

## E.5 Signer Anonymity

We perform the simulation in the simulated crs world. This shouldn't affect the adversary's advantage by more than negligible amount. Under a simulated crs, the NIZK has perfect zero-knowledge. Therefore, the only terms in the signature of reputation that can possibly reveal the signer is the encryption of the unblinded vote $\text{CCAEnc.Enc}(U_1, \ldots, U_c)$, and $\mathsf{rep} = \textsc{ShowRep}(U_1, \ldots, U_c)$.

In the challenge stage, the adversary submits two signers $i_0$ and $i_1$, and a list of votes for each signer. Let $\vec{U_0} = (U_{0,1}, \ldots, U_{0,c})$, $\vec{U_1} = (U_{1,1}, \ldots, U_{1,c})$ denote the set of distinct unblinded votes for $i_0$ and $i_1$ respectively. These unblinded votes may be obtained by decrypting the ciphertexts in the votes.

Now consider the following hybrid sequence:

*Game 0*: In Game 0, the challenger chooses user $i_0$ to answer the challenge query. That is,

$$\Sigma^* = \Big(\ldots \quad E_0 = \text{CCAEnc.Enc}(\vec{U_0}), \quad \mathsf{rep}_0 = \textsc{ShowRep}(\vec{U_0}), \quad \ldots\Big)$$

*Game M*: In Game M, the challenger encrypts the unblinded votes from $i_0$, but uses the unblinded votes from $i_1$ in the ShowRep algorithm. In addition, it uses the simulation secret $\mathsf{simkey}$ to construct the NIZK.

$$\Sigma^* = \Big(\ldots \quad E_0 = \text{CCAEnc.Enc}(\vec{U_0}), \quad \mathsf{rep}_1 = \textsc{ShowRep}(\vec{U_1}), \quad \ldots\Big)$$

*Game 1*: In Game 1, the challenger chooses user $i_1$ to answer the challenge query. That is,

$$\Sigma^* = \Big(\ldots \quad E_1 = \text{CCAEnc.Enc}(\vec{U_1}), \quad \mathsf{rep}_1 = \textsc{ShowRep}(\vec{U_1}), \quad \ldots\Big)$$

*Game M and Game 1 are indistinguishable.* We can prove this through a reduction to the security of the encryption scheme. We only require CPA security in this case. We now show that given an adversary that can distinguish Game M and Game 1, we can build a simulator that breaks the CPA security of the encryption scheme. The simulator obtains the public key of the CCAEnc scheme from an encryption challenger $\mathcal{C}$. The simulator sets up the parameters of our reputation system, such that the CCAEnc used in our reputation system agrees with those received from $\mathcal{C}$. The simulator picks all other parameters directly (under a simulated crs), and proceeds with the signer anonymity game as prescribed, except for the *Ch_SignerAnon* query. In the *Ch_SignerAnon* query, the simulator decrypts the IKEnc ciphertext in the submitted votes, and obtains two sets of unblinded votes: $\vec{U}_0 = (U_{0,1}, \ldots, U_{0,c})$ and $\vec{U}_1 = (U_{1,1}, \ldots, U_{1,c})$. The simulator submits the two sets of unblinded votes to the encryption challenger, and obtains $E_b = \text{CCAEnc.Enc}(\vec{U}_b)$. The simulator builds $E_b$ and $\mathsf{rep}_1$ into the challenge signature of reputation $\Sigma^*$, and uses the simulation secret to simulate the NIZK proofs. If the adversary can distinguish whether it is in Game M or Game 1, then the simulator would succeed in distinguishing which set of unblinded votes $\mathcal{C}$ encrypted. Notice that in the above, we modified the standard IND-CPA security game such that the simulator submits two sets of plaintexts (as opposed to two plaintexts) to the challenger $\mathcal{C}$. This can be derived from the standard IND-CPA security through a simple hybrid argument.

*Game 0 and Game M are indistinguishable.* By reduction to the reputation anonymity of the unblinded scheme.

The simulator obtains all users' $(\mathsf{rcvkey}, \mathsf{votekey})$ from $\mathcal{C}$, the challenger of the unblinded scheme. The simulator builds these into the user credentials of the full reputation system. The simulator picks all other parameters needed directly (under a simulated crs), and proceeds to interact with the adversary prescribed, except in the *Ch_SignerAnon* query.

When answering the *Ch_SignerAnon* query, the simulator first checks if all the votes correspond to the same receiver specified by the adversary. This can be done through the OpenNym algorithm defined in Appendix E.4, as the simulator knows all users' secret credentials. The simulator now decrypts the IKEnc ciphertext in the votes to obtain two sets of unblinded votes, $\vec{U}_0 = (U_{0,1}, \ldots, U_{0,c})$ and $\vec{U}_1 = (U_{1,1}, \ldots, U_{1,c})$.

Had we used a real crs, these unblinded votes must be traceable to a registered voter and recipient (except with negligible probability), due to the traceability of votes and the perfect soundness of NIZK proofs. Under a simulated crs, the unblinded votes must be traceable to a registered voter and recipient as well, since otherwise, we may build a simulator that distinguishes a simulated crs and a real crs. As the simulator knows all users' $\mathsf{rcvkey}$ and $\mathsf{votekey}$, the simulator can identify the voters from these unblinded votes through a brute force enumeration method:

$$\text{If } U = \text{VoteUnblinded}(\mathsf{rcvkey}_i, \mathsf{votekey}_j), \text{ then } U \text{ is an unblinded vote from } j \text{ to } i$$

As a result, the simulator obtains two sets of voters $\vec{j}_0 = (j_{0,1}, \ldots, j_{0,c})$, and $\vec{j}_1 = (j_{1,1}, \ldots, j_{1,c})$. The simulator now submits the two lists of voters to $\mathcal{C}$, and in return, obtains $\mathsf{rep}_b = \text{ShowRep}(\vec{U}_b)$. It builds $E_0$ and $\mathsf{rep}_b$ into the challenge signature of reputation $\Sigma^*$, and uses the simulation secret $\mathsf{simkey}$ to simulate the NIZK proofs. Clearly, if $b = 0$, then the above game is identical to Game 0; otherwise, it is identical to Game M.

## E.6 Voter Anonymity

In the voter anonymity game, the adversary submits two voters $j_0^*$ and $j_1^*$ and a $\mathsf{nym}^*$ in the challenge stage, and obtains a vote from one of these voters $j_b^*$ on the specified $\mathsf{nym}^*$. $\mathsf{nym}^*$ must open to an uncorrupted user $i^*$. There are two places in the simulation that can leak information about the challenger's coin $b$. The first place is obviously the challenge vote. The second place is more obscure: if the adversary submits the challenge vote $\mathsf{vt}^*$ (or some correlated version of it) in a *SignRep* query, it learns a signature of reputation that encodes information about $j_b^*$. We start by proving that the adversary is not able to learn anything from the *SignRep* queries. To this end, we define the following hybrid game:

*Game I.* We modify the original voter anonymity game in the following way. Whenever the adversary makes a *SignRep* query, the challenger decrypts the IKEnc ciphertext in the votes and obtains a list of unblinded votes. Let $c$ denote the number of distinct unblinded votes. The challenger now picks a random recipient, and random $c$ voters. It computes a signature of reputation corresponding to the above recipient and voters. We henceforth refer to a signature of reputation constructed in this way as a random signature of reputation. Notice that in Game I, the *SignRep* queries contain no information about which voter was chosen in the *Ch_VoterAnon* query.

**Lemma E.7.** *Answering SignRep queries with random signatures of reputation does not affect the adversary's advantage in the voter anonymity game by more than a negligible amount.*

*Proof.* By reduction to signer anonymity. Let $q$ denote the total number of *SignRep* queries where $\mathsf{vt}^*$ is involved. Consider the following hybrid sequence. In the $d$-th game ($0 \leq d \leq q$), the challenger truthfully answers the first $k$ *SignRep* queries where $\mathsf{vt}^*$ is involved. For the remaining *SignRep* queries, the simulator returns random signatures of reputation. Due to the hybrid argument, it suffices to show that the $(d-1)$-th and $d$-th games are computationally indistinguishable, where $1 \leq d \leq q$.

*Setup*: The simulator obtains $\mathsf{params}$ and all users' credentials from the signer anonymity challenger $\mathcal{C}$.

*Corrupt*, *Nym*, *Vote*: Compute a result to these queries normally.

*Ch_VoteAnon*: Flip a random coin $b$ and compute a vote from $j_b^*$ normally.

*SignRep*: For the first $d-1$ *SignRep* queries, answer faithfully. For the $d$-th query, let $S = (\mathsf{vt}_1, \ldots, \mathsf{vt}_k)$ denote the list of votes specified by the adversary. The simulator first checks if all votes submitted correspond to the same recipient specified by the adversary by calling the OpenNym procedure defined in Section E.4.

As the simulator knows all users credentials, it can decrypt the IKEnc ciphertext in the votes and obtain a list of unblinded votes. Let $c$ denote the number of distinct unblinded votes.

Now the simulator picks a random recipient $i'$ and $c$ distinct voters $j'_1, \ldots, j'_c$. It constructs $c$ votes from $j'_1, \ldots, j'_c$ to $i'$ respectively. Denote this set of votes as $S'$.

The simulator now submits the message $\mathsf{msg}$, and two sets of votes $S$ and $S'$ to the signer anonymity challenger $\mathcal{C}$. In return, the simulator obtains a signature of reputation $\Sigma$, which the simulator passes along in response to the adversary's query.

Notice that if $\mathcal{C}$ returned a $\Sigma$ corresponding to $S$, then the above simulation is identical to Game $d$. Otherwise, it is identical to Game $d-1$. Therefore, the adversary's advantage should differ only by a negligible amount in these two adjacent games. □

**Remark 3.** *In the above, the challenger computes a random signature of reputation by selecting $c$ random voters $j_1, \ldots, j_c$ and a random recipient $i$, computing $c$ votes from these voters to $i$, and then directly calling the* SIGNREP *algorithm to construct the signature of reputation.*

*Under a simulated crs, the challenger can use the following alternative strategy: It computes $c$ unblinded votes $U_1, \ldots, U_c$ from $j_1, \ldots, j_c$ to $i$ respectively. Then, it calls* CCAENC.ENC *to encrypt these unblinded votes, and calls* $\text{SHOWREP}(U_1, \ldots, U_c)$ *to construct* $\mathsf{rep}$. *Finally, the challenger uses* $\mathsf{simkey}$ *to simulate the NIZK, and calls the one-time signature scheme to sign everything.*

*Under a simulated crs, the signature of reputation computed in the above two ways are identically distributed.*

*Game II.* Notice that in Game I, the challenger decrypts the IKENC ciphertext in the votes to uncover the unblinded votes. In Game II, the challenger picks the parameters of the system such that it knows the decryption key to the CCAENC scheme. Instead of decrypting the IKENC ciphertext, the challenger decrypts the CCAENC ciphertext instead, and counts the number distinct voters $c$. Then it returns to the adversary a random signature of reputation consisting of exactly $c$ votes.

Game II is identically distributed as Game I under a real crs, due to the perfect soundness of the NIZK proofs and the traceability of the votes.

Later, under the simulated crs, the simulator sticks to decrypting the CCAENC ciphertext for opening the votes.

We now show that the challenge vote $\mathsf{vt}^*$ does not reveal sufficient information for the adversary to distinguish whether $\mathsf{vt}^*$ comes from $j_0^*$ or $j_1^*$. To demonstrate this, we will perform simulations under a simulated crs.

*GameSim.* The challenger now plays the above-defined Game II with the adversary under a simulated crs. This does not affect the adversary's advantage by more than a negligible amount.

We now show that the adversary's advantage in GameSim is negligible. There are two ciphertexts in the challenge vote $\mathsf{vt}^*$, the IKENC ciphertext $C_1$, and the CCAENC ciphertext $C_2$. These two ciphertexts are the only places that may leak information about which voter is chosen for the challenge query.

We define the following hybrid sequence.

*Game 0*: The challenger chooses $j_0^*$ for the challenge query.

*Game M*: When answering the challenge query, the challenger uses $\mathsf{votekey}_{j_1^*}$ to compute $C_1$ (through a homomorphic transformation as prescribed by the VOTE algorithm). However, it calls CCAENC.ENC to encrypt $x_{j_0^*}$, and produces $C_2$. The challenger now uses the simulation secret $\mathsf{simkey}$ to simulate the NIZK. Eventually, the challenger uses the one-time signature scheme to sign everything and returns the result to the adversary.

*Game 1*: The challenger chooses $j_1^*$ for the challenge query.

*Game M is indistinguishable from Game 1.* By reduction to the security of the selective-tag CCA encryption scheme.

*Setup*: Simulator learns the public key of the CCAEnc scheme from a challenger $\mathcal{C}$ of the encryption scheme. The simulator selects $\mathsf{sk_{ots}}^*, \mathsf{vk_{ots}}^*$, computes the selected tag $\mathsf{tag}^* = H(\mathsf{vk_{ots}}^*)$, and commits $\mathsf{tag}^*$ to the challenger $\mathcal{C}$. The simulator sets up all other parameters as normal, and registers $n$ users.

*Corrupt*, *Nym*, *Vote*: As the simulator knows all users' secret credentials, it can compute answers to these queries normally.

*Ch_VoterAnon*: The simulator obtains two voters $j_0^*$, $j_1^*$ and a $\mathsf{nym}$ from the adversary. The simulator forward $x_{j_0^*}$ and $x_{j_1^*}$ to the encryption challenger $\mathcal{C}$, and gets back $C^* = \text{CCAEnc.Enc}(\mathsf{pk_{cca}}, x_{j_b^*}, \mathsf{tag}^*)$. It builds the ciphertext $C^*$ into the challenge vote. Now the simulator uses $\mathsf{votekey}_{j_1^*}$ to compute the IKEnc ciphertext $C_1$, and uses $\mathsf{simkey}$ to simulate the NIZK proofs. Finally, it calls the one-time signature scheme to sign everything, and returns the resulting vote to the adversary.

*SignRep*: The query includes a list of votes. Check if all votes correspond to the same recipient specified by the adversary by calling the OpenNym procedure as defined in Appendix E.4.

Now the simulator needs to count the number of distinct voters. If the vote is the same as the challenge vote, consider that vote to be from either of the challenge voters. This will not affect the total count of distinct voters, due to the requirements of the voter anonymity game.

If the vote is not equal to the challenge vote, the simulator calls the decryption oracle of the CCAEnc scheme. The tag (under which the decryption oracle is called) must be different from the selected tag $\mathsf{tag}^*$. We show this below in Lemma E.8.

The decryption oracle returns a set of $x_j$ values that identify the set of voters. The simulator counts the number of distinct voters $c$, and constructs a random signature of reputation consisting of $c$ distinct voters.

Clearly, if $\mathcal{C}$ returned $\text{CCAEnc.Enc}(\mathsf{pk_{cca}}, x_{j_0^*}, \mathsf{tag}^*)$, the above simulation is identically distributed as Game M. Otherwise, it is identically distributed as Game 1.

**Lemma E.8.** *In the above simulation, whenever the simulator queries the decryption oracle of the* CCAEnc*, the tag of the encryption differs from* $\mathsf{tag}^*$ *except with negligible probability.*

*Proof.* Due to the security of the one-time signature scheme, the vote (which is not equal to $\mathsf{vt}^*$) must be signed under a key $\mathsf{sk_{ots}}' = \mathsf{sk_{ots}}^*$. Let $\mathsf{vk_{ots}}'$ denote the corresponding verification key. Then the tag used in the CCAEnc scheme $\mathsf{tag}' = H(\mathsf{vk_{ots}}')$ must be different from $\mathsf{tag}^*$ due to the collision resistance of the hash function. A more detailed proof of this can be found in Groth's group signature paper [Gro07]. □

*Game 0 is indistinguishable from Game M.* We now show that if there exists an adversary that can distinguish Game 0 from Game M, we can build a simulator that breaks either the IND-CPA of the IKEnc scheme, or the vote anonymity of the unblinded scheme.

Recall that in the security definition of voter anonymity, the adversary can win the voter anonymity game in two cases depending on whether the recipient is corrupted or not.

Below, we build a simulator which guesses ahead of the game whether the challenge query will correspond to an uncorrupted recipient or a corrupted recipient. Depending on the guess, the simulator will use different strategies for the simulation. If later the simulator's guess turns out to be wrong, the simulator simply aborts. The simulator has probability at least a half of guessing right.

We now describe the simulator's strategy for each of the two cases.

*Case 1*: Uncorrupted recipient.

We build a reduction to the IND-CPA security of the IKEnc scheme. The simulator guesses upfront which recipient will be submitted in the *Ch_VoterAnon* query. Denote this challenge recipient as $i^*$. The simulator will abort if the guess later turns out to be wrong.

*Setup*: From an encryption challenger $\mathcal{C}$, the simulator obtains the public parameters $\mathsf{params}_{\text{ike}}$, and a user public key $\mathsf{upk}_{\text{ike}}{}^*$ which it tries to attack. The simulator lets this $\mathsf{upk}_{\text{ike}}{}^*$ to be user $i^*$'s user public key. For all other users, the simulator picks their $\mathsf{upk}_{\text{ike}}$ and $\mathsf{usk}_{\text{ike}}$ by directly calling the IKEnc.GenKey algorithm.

The simulator chooses parameters of the CCAEnc scheme such that it knows the decryption key. The simulator generates the remaining parameters directly.

*Nym*, *Vote*: The simulator can answer these queries directly.

*Corrupt*: If the adversary corrupts the $i^*$-th user, abort. Otherwise, return the secret credential for the specified user to the adversary.

*SignRep*: The simulator first checks if all votes correspond to the same recipient specified by the adversary, by using the OpenNym procedure defined in Section E.4. Next, the simulator calls the decryption algorithm of the CCAEnc scheme, to decrypt the CCAEnc ciphertext in the submitted votes. The simulator counts the number of distinct unblinded votes, and builds a random signature of reputation consisting of the same number of voters.

*Ch_VoterAnon*: The simulator gets two voters $j_0^*$ and $j_1^*$, and a $\mathsf{nym}$. If the $\mathsf{nym}$ does not open to $i^*$, abort. (OpenNym is implemented using the procedure described in Section E.4). The simulator now computes the unblinded votes $U_{j_0^*,i}$ and $U_{j_1^*,i}$ and submits them to the encryption challenger $\mathcal{C}$. The simulator gets back a ciphertext IKEnc.Enc($\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike}}{}^*, U_{j_b^*,i}$), this will be the $C_1$ ciphertext in the resulting vote. The simulator now computes the CCAEnc ciphertext directly on $x_{j_0^*}$, and simulates the NIZK proofs. Eventually, the simulator calls the one-time signature scheme to sign everything, and returns the resulting vote to the adversary.

It is not hard to see that if $\mathcal{C}$ encrypted $U_{j_0^*,i}$, then the above game would be identically distributed as Game 0. Otherwise, it is identically distributed as Game 1.

*Case 2*: Corrupted recipient.

By reduction to voter anonymity of the unblinded scheme.

*Setup*. The simulator obtains all users' $\mathsf{rcvkey}$ and $\mathsf{vpk}$ from the vote anonymity challenger $\mathcal{C}$ of the unblinded scheme. The simulator now guesses the challenge voters $j_0^*$ and $j_1^*$ that the adversary will specify in the *Ch_VoterAnon* query, as well as the challenge recipient $i^*$. If the simulator's guesses later turn out to be wrong, the simulation simply aborts. Now, through *Corrupt* queries to $\mathcal{C}$, the simulator corrupts all users' $\mathsf{vsk}$ except for $j_0^*$ and $j_1^*$. The simulator makes *VoteUnblinded* queries to $\mathcal{C}$, and obtains an unblinded vote from $j_0^*$ and $j_1^*$ to all users except $i^*$. The simulator picks the parameters of the CCAEnc such that it knows its secret decryption key. The remaining system parameters are picked directly.

*Corrupt.* If the adversary queries $j_0^*$ or $j_1^*$, abort the simulation. Otherwise, return the specified user's credential to the adversary.

*Nym.* Compute directly.

*Vote.* The adversary submits a voter $j$ and a $\mathsf{nym}$ which opens to $i$. The simulator can decide $i$ by calling the alternative OpenNym algorithm (see Section E.4). If $j \in \{j_0^*, j_1^*\}$ and $i = i^*$, abort the simulation. Otherwise, the simulator has queried $\mathcal{C}$, and obtained an unblinded vote $U$ from $j$ to $i$. The simulator computes an IKEnc encryption of the unblinded vote $U$, by directly encrypting it. The simulator computes the CCAEnc ciphertext on $x_j$ directly. The simulator uses $\mathsf{simkey}$ to simulate the NIZK proofs.

*Ch_VoterAnon.* The simulator calls the alternative OpenNym algorithm (see Section E.4) to decide the recipient $i$. The simulator now forwards the two specified voters $j_0^*$ and $j_1^*$ and the recipient $i$ to $\mathcal{C}$, to obtain an unblinded vote $U_b^*$ corresponding to one of the voters $j_b^*$. The simulator now encrypts the $U_b^*$ by directly calling the IKEnc.Enc algorithm. The simulator computes the CCAEnc ciphertext on $x_{j_0^*}$ directly, and uses $\mathsf{simkey}$ to simulate the NIZK.

*SignRep.* The simulator calls the alternative OpenNym algorithm (see Section E.4) to check that all specified votes open to the specified recipient $i$. Now the simulator calls the decryption algorithm of the CCAEnc and obtains a set of voters. The simulator counts the number of distinct voters $c$, and computes a random signature of reputation with $c$ voters. As we mentioned in Remark 3, the simulator only needs to know $c$ unblinded votes for a random recipient $i$, to compute a random signature of reputation containing $c$ voters.

## E.7 Receiver Anonymity

By reduction to the IK-CPA security of the IKEnc scheme.

We perform the simulation under a simulated crs. This does not affect the adversary's advantage by more than a negligible amount.

*Setup.* The simulator guesses which two users $i_0^*$ and $i_1^*$ the adversary will submit in the *Ch_RecvAnon* query. The simulator obtains two user public keys $\mathsf{upk}_{\mathrm{ike},0}^*$ and $\mathsf{upk}_{\mathrm{ike},1}^*$ from the IK-CPA challenger $\mathcal{C}$. It lets $i_0^*$'s user public key to be $\mathsf{upk}_{\mathrm{ike},0}^*$, and $i_1^*$'s user public key to be $\mathsf{upk}_{\mathrm{ike},1}^*$. The simulator calls IKEnc.GenKey to generate the ($\mathsf{upk}_{\mathrm{ike}}$, $\mathsf{usk}_{\mathrm{ike}}$) pairs for all other users. As a result, the simulator knows all users' $\mathsf{usk}_{\mathrm{ike}}$ except for $i_0^*$ and $i_1^*$.

The simulator picks the parameters of the CCAEnc.Enc encryption scheme, such that it knows the secret decryption key. The simulator picks the remaining system parameters directly.

*Corrupt.* If the query is on $i_0^*$ or $i_1^*$, abort. Otherwise, return the user's credential to the adversary.

*Nym*, *Vote.* Compute directly.

*SignRep.* In case any of the $\mathsf{nym}$s specified is equal the challenge $\mathsf{nym}$, abort.

Otherwise, the simulator calls the OpenNym procedure defined in Section E.4 to determine the recipient and check that all of the nyms correspond to the same recipient $i$ as specified by the adversary.

The simulator decrypts the CCAEnc ciphertext in each vote and obtains a set of voters $j_1, \ldots, j_c$. With knowledge of all users votekey and rcvkey, the simulator can compute the unblinded votes from $j_1, \ldots, j_c$ to $i$. Now it computes the CCAEnc ciphertext and the rep parts of the signature based on the unblinded votes. The simulator uses the simkey to simulate the NIZK proofs.

*Ch_RecvAnon.* The adversary specifies two users, $i_0^*$ and $i_1^*$. If $i_0^*$ and $i_1^*$ disagree with the simulator's guesses, abort. The simulator specifies $(\mathsf{rcvkey}_{i_0^*}, 1, 1)$ and $(\mathsf{rcvkey}_{i_1^*}, 1, 1)$ to $\mathcal{C}$ and obtains a challenge ciphertext
$C = \text{IKEnc.Enc}(\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike},i_b^*}, (\mathsf{rcvkey}_{i_b^*}, 1, 1))$. Recall that the two encryptions of 1 are needed to rerandomize the ciphertext later when a voter performs homomorphic transformation on the ciphertext. (Due to the hybrid argument, the IK-CPA game may be modified such that the encryption adversary submits longer messages consisting of multiple elements in $\mathbb{G}$ in the challenge phase.) The simulator builds the challenge ciphertext $C$ into the nym, and simulates the NIZK proofs. Finally, it calls the one-time signature scheme to sign everything, and returns the resulting nym to the adversary.

*Guess.* The adversary outputs a guess $b'$. The simulator outputs the same guess.

It is not hard to see that if the adversary succeeds in guess $b'$ with more than negligible advantage, the simulator would have more than negligible advantage in the IK-CPA game.

## E.8 Reputation Soundness

The adversary plays the reputation soundness game, and outputs a forged signature of reputation $\Sigma^*$ at the end of the game. Suppose $\Sigma^*$ signs the message $\mathsf{msg}^*$ and the reputation count $c^*$.

If the adversary wins the game, it is a requirement that the adversary has made a *SignRep* query on the message $\mathsf{msg}^*$ and reputation count $c^*$. Therefore, $\Sigma^*$ cannot be equal to any signature of reputation returned by a *SignRep* query. Due to the traceability and non-frameability of signature of reputation, $\Sigma^*$ must open to a signer $i^*$ within the adversary's coalition, that is, a signer that has been corrupted through a *Corrupt* query.

Now apply OpenSigRep to open $\Sigma^*$ to a set of $c^*$ distinct voters. This fails with negligible probability due to the traceability of the signature of reputation. As $c^* > \ell_1 + \ell_2$, there must exist an uncorrupted voter $j$ who voted for $i^*$, and the adversary has not made a *Vote* query from $j$ to any nym that opens to $i^*$. But this breaks the unforgeability of signature of reputation.

# F Security Proofs for the Space Efficient Scheme

**Theorem F.1.** *The algorithms* Setup, GenCred, GenNym, Vote, SignRep$'$, *and* VerifyRep$'$ *constitute an* $\varepsilon$-sound *scheme for signatures of reputation.*

*Proof.* We prove this in the random oracle model through a reduction from the soundness of the regular scheme, which is proven in Appendix E. Assume we have some adversary $\mathcal{A}$ trying to break the $\varepsilon$-soundness game. $\mathcal{A}$ will only be able to compute hash values by querying the random oracle $H$. As $\mathcal{A}$ runs, $H$ records the queries $\mathcal{A}$ makes in a table and returns responses selected uniformly at random (except for repeated queries, in which case the previous value is returned).

Eventually $\mathcal{A}$ outputs a message $\mathsf{msg}$ and a forged signature of reputation $\Sigma$. Let $c = \textsc{VerifyRep}'(\mathsf{params}, \mathsf{msg}, \Sigma)$. Assume that $c \neq \perp$, $(1-\varepsilon)c > \ell_1 + \ell_2$, and the hashes and challenge set computation in $\Sigma$ verify correctly. We will bound the probability that the all the votes in the challenge set verify.

The challenger looks through the hash values included and is able to find all their preimages based on the queries recorded by $H$. From these the challenger reconstructs the full hash tree and all $c$ original leaf values $\omega_1, \ldots, \omega_c$. For each $1 \leq i \leq c$, the challenger verifies $\theta_i$ and $\zeta_i$ and checks that $R_{\rho_i} < R_{\rho_{i+1}}$. Let $c' \leq c$ be the number of these votes which pass both checks.

We distinguish two cases:

1. $c' \geq (1-\varepsilon)c$
2. $c' < (1-\varepsilon)c$

Case 1 must occur with probability less than or equal to some negligible function $\nu(\lambda)$, otherwise the challenger could output the $c' > \ell_1 + \ell_2$ valid votes and then it would be an adversary which would break the regular reputation soundness property.

We now bound the probability of all the challenge votes verifying in Case 2. Since the challenge indices were selected by evaluating $H$ on unique inputs, $I$ is a uniformly random set of $\ell$ votes. So the probability that these are all among the $c'$ valid votes is

$$\frac{c'}{c} \cdot \frac{c'-1}{c-1} \cdot \frac{c'-2}{c-2} \cdots \frac{c'-(\ell-1)}{c-(\ell-1)} = \prod_{i=0}^{\ell-1} \frac{c'-i}{c-i} \quad .$$

Since $c' < (1-\varepsilon)c$, this probability strictly less than

$$\prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c - i}{c-i} \quad .$$

We now show that the above value is at most $e^{-\lambda}$.

Since $\ell$ was selected as $\lceil \frac{\lambda}{\varepsilon} \rceil$, we may reason as follows.

$$\begin{aligned}
\ell = \left\lceil \frac{\lambda}{\varepsilon} \right\rceil &\geq \left\lceil \frac{\lambda}{-\log(1-\varepsilon)} \right\rceil \quad \text{(using the identity } \log(1+x) \leq x \text{ for } x > -1) \\
\implies \ell &\geq \frac{\lambda}{-\log(1-\varepsilon)} \\
\implies -\lambda &\geq \ell \log(1-\varepsilon) = \sum_{i=0}^{\ell-1} \log(1-\varepsilon) = \sum_{i=0}^{\ell-1} \log \frac{(1-\varepsilon)c}{c} \\
&\geq \sum_{i=0}^{\ell-1} \log \frac{(1-\varepsilon)c - i}{c-i} = \log \prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c - i}{c-i}
\end{aligned}$$

$$\begin{aligned}
\implies \log \prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c - i}{c-i} &\leq \log e^{-\lambda} \\
\implies \prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c - i}{c-i} &\leq e^{-\lambda}
\end{aligned}$$

So the probability of all the challenge votes verifying and Case 2 occurring is strictly less than $e^{-\lambda}$.

So overall, the probability of all the challenge votes verifying is less than $e^{-\lambda} + \nu(\lambda)$, and thus the probability that $\Sigma$ verifies is less than $e^{-\lambda} + \nu(\lambda)$. Since $\nu(\lambda)$ is negligible in $\lambda$, $e^{-\lambda} + \nu(\lambda)$ is also negligible in $\lambda$. $\square$