2020 IEEE 33rd Computer Security Foundations Symposium (CSF)

Anonymous Lottery In The Proof-of-Stake Setting

Foteini Baldimtsi
Computer Science Department
George Mason University
Fairfax, VA 22030 USA
foteini@baldimtsi.com

Varun Madathil, Alessandra Scafuro , Linfeng Zhou
Department of Computer Science
North Carolina State University
Raleigh, NC 27606 USA
{*vrmadath, ascafur, lzhou15*}*@ncsu.edu*

***Abstract*—When Proof-of-Stake (PoS) underlies a consensus protocol, parties who are eligible to participate in the protocol are selected via a public selection function that depends on the stake they own. Identity and stake of the selected parties must then be disclosed in order to allow verification of their eligibility, and this can raise privacy concerns.**

In this paper, we present a modular approach for addressing the identity leaks of selection functions, decoupling the problem of implementing an anonymous selection of the participants, from the problem of implementing others task, e.g. consensus.

We present an ideal functionality for anonymous selection that can be more easily composed with other protocols. We then show an instantiation of our anonymous selection functionality based on the selection function of Algorand.

***Keywords*-Blockchain, Proof-of-Stake, Privacy**

I. Introduction

Permissionless Blockchain. Permissionless blockchains, first introduced in Bitcoin [1], are open systems where any party is allowed to participate by provably investing in some type of resource. Following this principle, many other permissionless blockchains have been developed which use different type of user resources (computation [1], memory [2], money [3], [4], [5], time [6], etc).

At the core of all permissionless blockchains, lies a mechanism for selecting the party that will decide how to extend the blockchain, that is, decide what will be the next added block. This is a challenging task in a permissionless setting, but thanks to the enforcement of provable user resources and assuming a fixed upper bound of adversarial resources, several selection methods have been shown to be effective. For example, in Bitcoin, the selection method is based on Proof-of-Work. The party to be selected is the one to first solve a computationally hard puzzle (which is fresh for each block and randomly generated by the system protocol). To participate in the selection, a party only needs to start working on the problem. Once a party finds a solution it can announce it and gain the right to extend the blockchain (as well as receive some rewards).

In Proof-of-Stake (PoS) blockchains (e.g., [3], [7]) instead, the selection is performed according to the amount of stake (e.g., tokens) a party owns in the system. For example, assuming that stake is money, a party is selected via a randomized process, with probability that is proportional to the amount of money she owns.

There is, however, a key difference between the two types of selection. In proof-of-work, the proof of winning the selection is a solution to a fresh, random puzzle, which can be completely disconnected from the identity of the winning party. As an example, let us consider the case of Bitcoin. The first step of any party that wishes to participate in the selection process is to assemble the block B_{i+1} to be added next on the blockchain if the party gets selected. This block includes a pointer to the previous block B_i, a set of transactions and a Bitcoin address to which the potential reward will be sent. The exact information included in the block defines the puzzle that the party needs to solve. If a party solves the puzzle before hearing about any other solution, it announces the block and the solution $(B, \text{solution})$. As long as the solution is valid and this was the first party to announce a solution for the next block, the party gets elected. What is interesting to note in this process is that the selection process does not depend on the identity of the party (the address included in the block B can always be a fresh one) – the proof of being selected is simply a valid puzzle solution for a given block. In contrast, in Proof-of-Stake selection it is not possible for parties to disconnect their identity from the proof of winning the selection, since their identity is part of the proof.

Anonymous Selection in Proof-of-stake Settings. The selection function used in known proof-of-stake consensus protocols must satisfy the following properties: privately evaluated, publicly verifiable and fair. The first property says that only the stakeholder can learn if she is eligible to speak next, thus, the selection function can be evaluated only with the knowledge of the secret key. This property is necessary for preventing adaptive corruption of the selected party, and is crucial for achieving consistency and chain-quality properties. The second property says that, a stakeholder PK_i can prove that she is eligible, by producing a proof that can be verified by anyone having access to PK_i (and corresponding stake). The last property says that the probability of being eligible follows a fixed and public metric of eligibility. This

Research Supported by NSF grants #1012798,#1764025

© 2020, Foteini Baldimtsi. Under license to IEEE.
DOI 10.1109/CSF49147.2020.00030

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

metric can be different in different applications. For proof-of-stake consensus, the metric of fairness is that a party wins with probability that is proportional to its stake. In general, PoS protocols can have different eligibility criteria (where the weight is scaled after a certain threshold, so that rich people are not selected too often). Independently of the eligibility criteria, a crucial property is that fairness must hold even in case parties generate their keys maliciously. That is, an adversary should not be able to craft keys that allow her to hit the eligibility criteria with higher probability.

In this work we focus on *anonymizing* the selection function in the Proof-of-Stake setting. Our goal is to provide a formal definition of anonymous selection, and show an instantiation.

A. Our Contributions

We now give an overview of our main contributions.

A Flexible Definition of Anonymous Selection. An anonymous Proof-of-Stake selection function should have all the properties of a regular PoS selection (e.g., privately evaluated, publicly verifiable, fair), but additionally it should guarantee that the proof of selection *hides the identity of the winner.*

To capture all the above properties, we design an ideal functionality that we call $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$. Our ideal functionality allows parties to register their identity P_i (along with associated stake stake_i when relevant). Once all parties are registered, any party can start making eligibility queries which are associated to a "tag" $\mathtt{tag}$. The semantic of a tag depends on the application that invokes the selection procedure. For example, in Algorand [7], a tag is a tuple (round,step,seed) since this is the information that defines when a new selection process must be performed. Similarly, in Ouroboros PoS [3], a tag is of the form (epoch nonce, slot number). A party P_i can ask $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ if she is eligible to speak for a certain $\mathtt{tag}$. $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ is parameterized by an eligibility predicate Eligible, which on input the tag $\mathtt{tag}$ and other information, such as stake, returns a bit $b \in \{0,1\}$. $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ correctly evaluates the predicate Eligible for P_i.

If eligible, P_i can then send $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ a message m, and obtain a proof π, for m and tag $\mathtt{tag}$. For example, in Algorand m could be a protocol message for the underlying Byzantine Agreement, or a block proposed by a leader. Only P_i can check her own eligibility, and this captures the *private evaluation* property. The *fairness* property is captured by the fact that $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ only computes a valid proof for parties that pass Eligible.

The proof π does not have any identity attached – capturing the *anonymity* property. Any other party P (even if not registered in the system) can later query $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ to verify that π is a valid proof for $m, \mathtt{tag}$, and get yes/no as an answer. This captures the *public verifiability* property. Furthermore, only proofs that are generated by $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ will correctly verify, and this captures the correctness and fairness of the system. An ideal functionality for anonymous lottery is also defined in a concurrent work [8] (that we discuss in more details in Section I-B). Their definition differs from ours in the following crucial aspect. In [8], when an eligible party asks the functionality to send m for tag $\mathtt{entry}$, the ideal functionality will broadcast the message m to all parties in the system. This indeed captures what typically happens in a consensus protocol, where messages are broadcast to all parties. However, this approach presents some potential drawbacks. First, anonymous selection and anonymous broadcast seem to be problems of different nature – one is at application level, the other is at network level. In particular, the guarantees that one can achieve against an adversary that can only act at application level might be much stronger than the guarantees one could hope to achieve against an adversary that works at network level. Indeed, it has already been observed in previous work (see Sec VI - C of [9], Sec 5.1 and 5.2 of [5] and Sec 4.2 of [10]) that there is some seemingly inherent leakage at network level that an adversary can exploit. Therefore, the anonymity guarantee promised by the functionality described in [8] might not be necessarily realizable (even in the ideal anonymous broadcast hybrid model). Second, an ideal functionality that enforces broadcast cannot be used in protocols where parties do not need/want to broadcast their eligibility to the entire network.

Our ideal functionality instead provides a proof π of eligibility for a party P_i and does not enforce any further action. This proof is an actual string that P_i can use in another protocol. This makes our ideal functionality more flexible and, we think, more easily composable with other protocols. In our work we first present an ideal functionality that allows parties to be eligible with the same weight. This can capture the lottery of Ouroboros style protocols where parties are selected with the same weight (but are selected more often based on their stake - in different $\mathtt{tag}$). Our functionality also captures the lottery of Algorand if we assume that each user is associated with one unit of stake. To capture the lottery functionality of Algorand where parties with different stake amounts are selected with different weights, we present a modification of our ideal functionality for the multi-stake setting $\mathcal{F}_{\mathsf{Anon\text{-}Selection\text{-}MS}}$ in Appendix B. Note that the ideal functionality of [8] does not capture this selection with multiple weights and cannot be used directly to replace the lottery function of Algorand.

Instantiation from Algorand Selection Function. We provide an implementation of $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ based on the underlying selection function of the Algorand protocol (as described in [7], which works as follows. In Algorand, a party P_i is identified by its public key pk_i. In order to check availability for a certain $\mathtt{tag}$, she uses her private (signing) key sk_i to compute a signature on $\mathtt{tag}$: $\sigma_i = SIG_{\mathsf{sk}_i}(\mathtt{tag})$.

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

This signature is given as input to a random oracle $\mathcal{H}$, i.e. $y = \mathcal{H}(\sigma_i)$. The random output y is then used to check eligibility: if y is below a threshold T, the party is selected, and the proof is simply the pair (y, σ_i). In order to verify such a proof one needs to use pk_i to verify the signature, and this obviously requires leaking the identity of the selected party.

To anonymize this selection function, a naïve approach would be to simply send y, and add a zero-knowledge proof for the statement: "y is the correct output of the random oracle evaluated on input a signature σ that verifies under *some* pk_i present in the system (i.e., in the set of all published public keys)." Note that the pre-image σ_i of the random oracle must remain hidden, since it reveals the identity of the stake-holder.

This straightforward approach, however, fails when $\mathcal{H}$ is modeled as a random oracle, since it can only be used as a black-box in the protocol. Thus one cannot prove properties of pre-images of the random oracle unless the random oracle is used as a black-box in the zero-knowledge proof (and no succinct *reusable* black-box proofs are known to exist so far). On the other hand, we stress that one cannot simply replace $\mathcal{H}$ with a concrete hash function in the proof, since the perfect unpredictability property of the random oracle is crucially used in the proof of security, when arguing security against maliciously chosen keys (for example see Sec. 3.2 of [11]).

Thus, as our second contribution we show how to overcome the above issue and avoid using the random oracle in the zero-knowledge proof, while still maintaining the same selection function of Algorand. We devise a method that allows one to prove properties about the "pre-image" of the output of the random oracle, while still using the random oracle as a black-box. Our approach is the following. Instead of proving a statement about a secret function applied on the *input* of the random oracle $\mathcal{H}$ we prove a statement about a secret function applied to the output of $\mathcal{H}$, which can be public. Crucially, we need that the function applied to the output y does not disturb the unpredictability properties we get from the random oracle. To do so, we use trapdoor *permutations*.

Our anonymous selection function therefore works as follows. For each tag $\mathtt{tag}$, there is a public value associated to a party P_i which is $V_i = \mathcal{H}(i|\mathtt{tag})$, and can (but it does not have to) be computed by everyone. Each party P_i also has associated a public key $\mathsf{TRP.pk}_i$ for a trapdoor permutation f. To check if eligible to speak for tag $\mathtt{tag}$, a party P_i proceeds as follows. She uses her trapdoor key $\mathsf{TRP.sk}_i$ to compute $v_i = f^{-1}_{\mathsf{TRP.sk}_i}(V_i)$ and then use randomness v_i to run predicated $\mathsf{Eligible}$, which in Algorand simply consists to check if $v_i < T$. If eligible, P_i computes a succinct non-interactive ZK argument (e.g. [12], [13]) proving that she knows a pre-image of one of the V_i that makes her eligible. Note that values $V_1, \ldots, V_N$ can be computed by everyone since they do not require any secret. In fact they can be pre-computed in advance, and consumed as the protocol proceeds. Note also that the statement of the zero knowledge proofs does not need to contain the list $V_1, \ldots, V_N$ but only their accumulated representation, that is, the root of a Merkle Tree.

B. Related Work

Concurrently and independently to our work there have been two relevant proposals: a framework of anonymous PoS proposed by Ganesh, Orlandi and Tschudi [8] (in Eurocrypt'19) and the "Ouroboros Crypsinous" protocol proposed by Kerber, Kohlweiss, Kiayias and Zikas [5] (in IEEE S&P'19). We will discuss both results and explain how we differ.

The work by Ganesh et al. [8] is the most closely related to ours. They introduce a clean framework to capture and abstract the lottery aspect of proof-of-stake with an ideal functionality, that they call $\mathcal{F}_{\mathsf{lottery}}$. As discussed earlier, the main difference with our formulation is that their ideal lottery functionality captures more than just lottery, since it also enforces broadcast of eligible messages, and this modeling choice could present potential drawbacks. We also note that even though $\mathcal{F}_{\mathsf{lottery}}$ abstracts the lottery from the claimed results, it is unclear how $\mathcal{F}_{\mathsf{lottery}}$ is/can be used as a black-box. Concretely, when claiming that "Ourobors Praos instantiated with private lottery results in a private proof-of-stake protocol" (See Corollary 1 of [8]), the *informal* proof does not actually use the ideal lottery functionality $\mathcal{F}_{\mathsf{lottery}}$, parameterized with the eligibility predicate used in Ouroboros Praos. Rather it uses the specific (game-based) security properties of the specific implementations of the anonymized version of the VRF used in Ouroboros Praos. This raises some confusion about whether one should think that Ouroboros Praos (with Anonymized VRF) is a secure realization of $\mathcal{F}_{\mathsf{lottery}}$ or whether $\mathcal{F}_{\mathsf{lottery}}$ can be used as a building block to realize a "private-proof-of-stake" protocol (though no definition of "private-proof-of-stake" protocol is provided in [8]).

Finally, [8] originally implemented $\mathcal{F}_{\mathsf{lottery}}$ by employing the VRF used in Ouroboros Praos, which is anonymized by simply adding a SNARK to prove that the VRF verifies correctly. Such implementation required the verification algorithm to evaluate the random oracle, and thus suffered from the issue of proving a property about the output of a random oracle (which we discussed above). An updated version of [8] can be found in [14] which replaces the VRF of Ouroboros Praos with the one used in Ouroboros Crypsinous[5] and avoids using the random oracle in the verification circuit used in the SNARK. We instead give an implementation based on Algorand's selection function. Similarly to us, [8] guarantees anonymity only in presence of static adversaries. Note however that our construction

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

could actually provide adaptive security for correctness (not anonymity). We discuss such an extension in Section VI-A.

Summing up, the key differences between our work and [8] is in the definition – our allows composability with more general protocols besides consensus– and in the instantiation – we use the selection function of Algorand, while [8] instantiate it using Ouroboros's VRF.

Ouroboros Crypsinous [5] does not focus on the general problem of anonymous selection in proof-of-stake setting, rather it focuses on defining private proof-of-stake blockchains. They provide an ideal "private ledger functionality" that aims to capture privacy of blocks and transactions. They then show how to build a private blockchain for payments extending the Ouroboros protocol with a confidentiality layer. Confidentiality is preserved in presence of "semi-adaptive" adversaries, that is, adversary that can corrupt a party at any time, but cannot access the state of the corrupted party immediately after corruption. Although their techniques are very interesting, they are tied to the Ouroboros PoS designs and private blockchains. Our work instead does not aim at adding anonymity to the Ouroboros blockchain specifically (though our technique could be used to hide the stakeholder identity in the Ouroboros blockchain).

II. Preliminaries

We start by setting the notation to be used throughout the paper. By PPT we denote a probabilistic polynomial-time algorithm. Let λ be the security parameter and $\|$ denote concatenation. We denote the uniform sampling of a value r from a set D as $r \leftarrow D$ and $r_1, \cdots, r_n \leftarrow D$ indicates that we sample from D a uniformly random subset of n elements. We use bold symbols for vectors of elements. For a vector $\mathbf{v}$, by $\mathbf{v}[i]$ we denote the ith entry of the vector. We say a function f is negligible in λ if for every polynomial p there exists a constant c such that $f(\lambda) < \frac{1}{p(\lambda)}$ when $\lambda > c$. Two ensembles $X = \{X_{\lambda,z}\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$ and $Y = \{Y_{\lambda,z}\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$ of binary random variables are said to be indistinguishable, $X \approx Y$, if for all z it holds that $|\Pr[X_{\lambda,z} = 1] - \Pr[Y_{\lambda,z} = 1]|$ is negligible in λ.

Let $\mathcal{R}$ be an efficiently computable binary relation. For pairs $(\mathsf{stmt}, w) \in \mathcal{R}$ we call stmt the statement and w the witness. Let $\mathcal{L}$ be the language consisting of statements in $\mathcal{R}$.

Non-Interactive Zero Knowledge Proof (NIZK). We recall the definition of a non-interactive zero knowledge proof system (adapted from [15] and [16]).

A non-interactive zero-knowledge proof system for a relation $\mathcal{R}$ is defined as a set of probabilistic polynomial time algorithms $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$. $\mathsf{NIZK.Setup}$ is a common reference string generation algorithm that produces a common reference string crs of length $\Omega(\lambda)$. The prover takes as input $(\mathsf{crs}, \mathsf{stmt}, w)$ and by running $\mathsf{NIZK.Prove}$ produces a proof π. The verifier takes as input $(\mathsf{crs}, \mathsf{stmt}, \pi)$ and runs $\mathsf{NIZK.Verify}$ to verify the proof. The following properties need to be satisfied:

- *Completeness*: The NIZK proof system is complete if an honest prover with a valid witness can convince an honest verifier. For all adversaries $\mathcal{A}$ we have:

$$\Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda);\ (\mathsf{stmt}, w) \leftarrow \mathcal{A}(\mathsf{crs});\ \pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \mathsf{stmt}, w) : \mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1 \wedge (\mathsf{stmt}, w) \in \mathcal{R}] = 1$$

- *Soundness*: A NIZK proof system is sound if it is infeasible to convince an honest verifier when the statement is false. For all polynomial size families $\{\mathsf{stmt}_\lambda\}$ of statements $\mathsf{stmt}_\lambda \notin \mathcal{L}$ and all adversaries $\mathcal{A}$ we have:

$$\Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda); \pi \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{stmt}_\lambda) : \mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_\lambda, \pi) = 1] \leq \mathsf{negl}(\lambda)$$

- *Zero-Knowledge*: A NIZK proof system is zero-knowledge if the proofs π do not reveal any information about the witness. That is, if there exists a polynomial time simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, where $\mathcal{S}_1$ returns a simulated common reference string crs together with a simulation trapdoor τ and an extraction key ek. The trapdoor τ enables $\mathcal{S}_2$ to simulate proofs without access to the witness. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have:

$$\Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda) : \mathcal{A}^{\mathsf{NIZK.Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1] \approx \Pr[(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \mathcal{S}_1(1^\lambda) : \mathcal{A}^{\mathcal{S}(\mathsf{crs},\tau,\cdot,\cdot)}(\mathsf{crs}) = 1]$$

  where $\mathcal{S}(\mathsf{crs}, \tau, \mathsf{stmt}, w) = \mathcal{S}_2(\mathsf{crs}, \tau, \mathsf{stmt})$ for $(\mathsf{stmt}, w) \in \mathcal{R}$ and both oracles output *failure* if $(\mathsf{stmt}, w) \notin \mathcal{R}$. Notice that we define the simulator $\mathcal{S}_1$ as in [16], where $\mathcal{S}_1$ not only outputs a simulated crs and a trapdoor τ, but also an extraction key ek.

We require the NIZK arguments to satisfy the following simulation extractability property as defined in [16].

- *Simulation Extractability*: Simulation extractability is a strong notion which requires that even after seeing many simulated proofs (even for false theorems), whenever the adversary generates a new proof, a simulator is able to extract a witness. More formally, a NIZK proof system is said to be *simulation extractable* if it satisfies computational zero-knowledge and additionally, there exists a polynomial-time algorithm $\mathsf{Extract}$, such that for any polynomial-time adversary $\mathcal{A}$, it holds that

$$\Pr[(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \mathcal{S}_1(1^\lambda); (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(\mathsf{crs},\tau,\cdot)}(\mathsf{crs}, \mathsf{ek}); w \leftarrow \mathsf{Extract}(\mathsf{crs}, \mathsf{ek}, \mathsf{stmt}, \pi) : \mathsf{stmt} \notin Q \wedge \wedge (\mathsf{stmt}, w) \notin \mathcal{L} \wedge \mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1] = \mathsf{negl}(n)$$

  where Q is the list of queries made by $\mathcal{A}$.

Trapdoor Permutation. We adapt the definition of trapdoor permutation from Bellare and Yung [17].

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

Definition 1 (Trapdoor Permutation)**.** *We say that* $(\mathsf{TRP.KeyGen}, f, f^{-1})$ *is a trapdoor permutation if the following conditions hold:*

- *Generation: For all* $\lambda > 0$*, the output of* $\mathsf{TRP.KeyGen}$ *on input* 1^λ *is a pair of* λ*-bit strings* $\mathsf{TRP.pk}, \mathsf{TRP.sk}$.
- *Permutation: For all* $\lambda > 0$ *and* $(\mathsf{TRP.pk}, \mathsf{TRP.sk}) \in \mathsf{TRP.KeyGen}(1^\lambda)$*, the maps* $f_{\mathsf{TRP.pk}}(\cdot)$ *and* $f^{-1}_{\mathsf{TRP.sk}}(\cdot)$ *are permutations of* $\{0,1\}^\lambda$ *which are inverse of each other. That is* $f^{-1}_{\mathsf{TRP.sk}}(f_{\mathsf{TRP.pk}}(x)) = x$ *and* $f_{\mathsf{TRP.pk}}(f^{-1}_{\mathsf{TRP.sk}}(y)) = y$ *for all* $x, y \in \{0,1\}^\lambda$.
- *Security: For all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*,* $\exists$ *a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[f_{\mathsf{TRP.pk}}(x) = y : (\mathsf{TRP.pk}, \mathsf{TRP.sk}) \leftarrow \mathsf{TRP.KeyGen}(1^\lambda$$
$$y \leftarrow \{0,1\}^\lambda; x \leftarrow \mathcal{A}(1^\lambda, \mathsf{TRP.pk}, y)] \leq \mathsf{negl}(\lambda)$$

III. Ideal Functionality for PoS Anonymous Selection

We present a definition of our anonymous selection functionality $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ in the UC-framework of [18].

Assumptions. We start by stating our assumptions.

Stake: We consider a setting where each party is associated with exactly one unit of stake. This ideal functionality can be used to replace selection in lottery based protocols like [11] and [7] if, for [7], we assume that each party is associated with one unit of stake. We take this approach to describe a lottery functionality to pick winners with the same weight (=1). This simpler functionality will allow to showcase how anonymity can be achieved in a simpler protocol, without trivializing the problem. To keep the notation general we use stake_i to denote the stake of party P_i. We also describe a modification of our ideal functionality to capture the lottery of [7] in the multi-stake scenario (under the assumption that majority of stake belongs to honest parties) where parties are selected with some weight in Section VI-B and Appendix B.

Registration: Before the execution of the functionalities, all parties register along with their stakes - (P_i, stake_i) with the functionality. By n we denote the number of registered parties. Similar to [8] we consider a *static setting* where new parties cannot register once the functionalities are being executed.

Corruption model: We assume static corruption, i.e. the corrupted parties are fixed throughout the entire execution. Note that we can achieve security against an adaptive adversary for correctness but achieve only static security for anonymity.

Our proposed functionality. We describe our anonymous selection functionality $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ in Figure 1. By $\mathtt{tag}$ we denote all public values corresponding to one execution of the protocol such as round and step number, random seed for the current round etc. Each registered party checks if it is "eligible" to speak for a tag $\mathtt{tag}$ by using the $\mathsf{EligibilityCheck}$ command which returns a bit $b \in \{0,1\}$. If the party is eligible to speak for $\mathtt{tag}$, then $b = 1$, otherwise $b = 0$. We stress that we add stake as an input for generality only. As mentioned above we assume that $\mathsf{stake}_i = 1$.

Functionality $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$

The ideal functionality is parameterized by an Eligible predicate and maintains the following elements: (1) A global set of registered parties $\mathcal{P} = ((P_1, \mathsf{stake}_1), \ldots, (P_n, \mathsf{stake}_n))$. (2) A table T, which has one row per party and a column for each $\mathtt{tag} \in \mathbb{N}$ given by parties when checking eligibility. The table stores the eligibility information of each party in each $\mathtt{tag}$. (3) A list $\mathcal{L}$, to store a proof π corresponding to a message msg in some $\mathtt{tag}$.

- Upon receiving $(\mathsf{EligibilityCheck}, sid, \mathtt{tag})$ from a party P_i do the following :
  1) If $P_i \in \mathcal{P}$ and $T(P_i, \mathtt{tag})$ is undefined, sample $r \in \{0,1\}^\ell$ run $\mathsf{Eligible}(r, \mathsf{stake}_i, \mathtt{tag})$ to get $b \in \{0,1\}$. Set $T(P_i, \mathtt{tag}) = b$
  2) Output $(\mathsf{EligibilityCheck}, T(P_i, \mathtt{tag}))$ to P_i
- Upon receiving $(\mathsf{CreateProof}, sid, \mathtt{tag}, msg)$ from a party P_i:
  1) If $T(P_i, \mathtt{tag}) = 1$, send $(\mathsf{Prove}, \mathtt{tag}, msg)$ to $\mathcal{A}$. Else, send $(\mathsf{Declined}, \mathtt{tag}, msg)$ to P_i.
  2) Upon receiving $(\mathsf{Done}, \psi, \mathtt{tag}, msg)$ from $\mathcal{A}$. Set $\pi \leftarrow \psi$ and record $(\pi, \mathtt{tag}, msg)$ in $\mathcal{L}$. Send $(\mathsf{Proof}, \pi, \mathtt{tag}, msg)$ to P_i
- Upon receiving $(\mathsf{Verify}, sid, \pi, \mathtt{tag}, msg)$ from some party P':
  1) If $(\pi, \mathtt{tag}, msg) \in \mathcal{L}$ output $(\mathsf{Verified}, sid, (\pi, \mathtt{tag}, msg), 1)$ to P'.
  2) If $(\pi, \mathtt{tag}, msg) \notin \mathcal{L}$, send $(\mathsf{Verify}, sid, (\pi, \mathtt{tag}, msg))$ to $\mathcal{A}$ and wait for a witness w from the adversary $\mathcal{A}$. Check if w is valid as follows:
     - Parse $w = (P_i, \mathtt{tag}, msg)$ and check that $T(P_i, \mathtt{tag}) = 1$
     - If yes, store $(\pi, \mathtt{tag}, msg)$ in the list $\mathcal{L}$ and send $(\mathsf{Verified}, sid, (\pi, \mathtt{tag}, msg), 1)$ to P'.

     If either of these checks are false output $(\mathsf{Verified}, sid, (\pi, \mathtt{tag}, msg), 0)$ to P'.

Figure 1. Anonymous selection functionality

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

If a party is eligible to speak and wishes to send the message msg, she can later query the ideal functionality via the command $(\mathsf{CreateProof}, \mathtt{tag}, msg)$ to obtain a proof π that she can use in any other protocol. Note that the $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}^{\mathsf{Eligible}}$ will provide such proof if the party was marked as Eligible for the tag. Any party that receives such a pair (msg, π) for a tag, can verify that the proof is correct by simply querying $(\mathsf{Verify}, \pi, \mathtt{tag}, msg)$ to $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}^{\mathsf{Eligible}}$. Note that the verification does not require any information on the identity of the sender of the proof, thus capturing the property of anonymity. Moreover, we note that the ideal functionality only maintains a list of proofs and does not store the identity of the party along with the proof in the list $\mathcal{L}$.

Our implementation additionally requires the standard random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ which is defined in Figure 2.

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

The functionality is parameterized by the security parameter λ. We write $T[x] = \perp$ to denote the fact that no pair of the form $(x, \cdot)$ is in T.

- Upon receiving (EVAL, x) from a party P do:
  1) If $T[x] = \perp$, sample a value y uniformly at random from $\{0,1\}^\lambda$, set $T[x] \leftarrow y$ and add $(x, T[x])$ to T.
  2) Return $(\mathsf{EVAL}, x, T[x])$ to the requester.

Figure 2. The random oracle functionality

IV. Realization of $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}^{\mathsf{Eligible}}$

In this section we propose a protocol $\Pi_{\mathsf{Anon\text{-}Selection}}^{\mathsf{Eligible}}$ to realize the ideal functionality $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}^{\mathsf{Eligible}}$ from Section III. Our realization is inspired by the selection algorithm of Algorand [7], which is run by every party to check if they are selected into a committee. To ease presentation we first describe how the selection algorithm of Algorand works in Section IV-A before we present our implementation $\Pi_{\mathsf{Anon\text{-}Selection}}^{\mathsf{Eligible}}$ in Section IV-B.

A. Selection Function in Algorand

In Figure 3 we describe the selection function used in Algorand (that we recast using our notation). Specifically, we consider the function described in the so called "theoretical paper" [7] where it is assumed that each public key is associated with *one unit of stake*. This implies that during the selection process each party is either selected to participate in the next round of the Algorand protocol or not[1]. This is a quite natural property of lottery protocols where each party holds one or more tickets (public keys for the case of Algorand) and some of them are selected while others aren't. Though our main construction is in this single-stake setting we give an intuition to extend the protocol to the multistake setting in Section VI-B.

[1]Note that in a later paper which describes the implementation of the Algorand system [19], it is assumed that each public key can have variable amounts of stake and during the selection process each selected party receives a weight that defines the power of the party in the later steps.

Algorand's selection algorithm

A party (P_i, stake_i) runs the selection algorithm to check if it is eligible to send messages in a following step of Algorand's protocol. Each party maintains a Table that stores the tag the party is eligible to speak in.

$\underline{\mathsf{Initialization}(1^\lambda)}$

- Generate signature key pair $(\mathsf{SIG.vk}_i, \mathsf{SIG.sk}_i) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$.
- Generate a master key pair $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
- Generate ephemeral signature key pairs for $|U|$ number of tag, $\{\mathsf{ESIG}.sk_ij, \mathsf{ESIG}.vk_ij\}_{j \in 1 \ldots |U|}$.
- Publish $(\mathsf{MPK}, \{\mathsf{ESIG}.vk_ij\}, \mathsf{SIG.vk}_i)$

$\underline{\mathsf{CheckEligibility}(\mathtt{tag})}$

- Compute $sorthash_i = H(\mathsf{SIG}_{sk_i}(\mathtt{tag}))$
- Run $b_i \leftarrow \mathsf{Eligible}(sorthash_i, \mathsf{stake}_i, \mathtt{tag})$.
- Store $\mathsf{Table}(\mathtt{tag}) = b_i$.

$\underline{\mathsf{CreateProof}(\mathtt{tag}, msg)}$

- If $\mathsf{Table}(\mathtt{tag}) = 0$, output $\perp$.
- If $\mathsf{Table}(\mathtt{tag}) = 1$, output $(\mathsf{ESIG}_{sk_i\mathtt{tag}}(msg), \mathsf{SIG}_{sk_i}(\mathtt{tag}), sorthash_i, \mathtt{tag})$

$\underline{\mathsf{Verify}(\pi, \mathtt{tag}, msg)}$

- Check that $\mathsf{SIG}_{sk_i}(\mathtt{tag})$ is a valid signature.
- Check that $H(\mathsf{SIG}_{sk_i}(\mathtt{tag})) = sorthash$.
- Check that $\mathsf{Eligible}(sorthash_i, 1, \mathtt{tag}) = 1$.
- Output 1, if all checks pass.

Figure 3. Algorand's selection algorithm

The Algorand selection process works as follows. Parties run the Initialization protocol to generate their keys as soon as they join the network and publish their public keys: a master public key and a signature verification key, as $(MPK, \mathsf{SIG.pk}_i)$. Using the master public key, each party generates a large number of ephemeral keys ($U = 10^6 \times m$ according to [7], where m is the expected number of steps of each execution of the protocol).

Parties run the selection function at different stages of the protocol to check if they are selected to speak in a specific stage. In Algorand a stage is identified by step number, round number and a random seed . We will use the notation tag

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

to capture the item for which the party wishes to test if she is selected. To check if a party can speak for an item tag the party first computes a value $v_i = \mathcal{H}(\sigma_i)$ where σ_i is the signature computed over tag using the signing key. The random value v_i is then used as input to the function Eligible described in Figure 4.

Note that the Eligible function takes as input stake_i as well, but is not explicitly used since in our setting we assume that only one unit of stake is associated with each public key.

Parties who are eligible to speak can obtain a publicly verifiable proof via the algorithm CreateProof, which takes in input the message they want to send. Note that CreateProof also includes an ephemeral signature on the message to ensure adaptive security, that is if the party is corrupted later, it cannot create a valid signature since the ephemeral key is deleted immediately.

To verify, parties use procedure Verify, which will check the validity of the signatures using the public key and check that the hash satisfies the properties required by the function Eligible.

B. Our Anonymized Selection Protocol

We now describe the protocol that realizes the $\mathcal{F}_{\text{Anon-Selection}}^{\text{Eligible}}$ functionality using the selection function (Eligible) of Algorand. Following [7], we assume that each public key is associated with one unit of stake. Note that this does not trivialize the problem, since it is still necessary to hide the identity of the user eligible to speak. Thus in our instantiation and all further descriptions, assume that $\mathsf{stake}_i = 1$.

Function $\mathsf{Eligible}(v_i, \mathsf{stake}_i, \mathtt{tag})$

Global variables for the protocol are totalStake and τ. totalStake defines the total stake of the parties in the network and τ is the expected number of parties to be selected (this depends on the tag).

1 : $p \leftarrow \dfrac{\tau}{\mathsf{totalStake}}$
2 : $b_i \leftarrow 0$
3 : **if** $\dfrac{v_i}{2^{len(v_i)}} < p$ **then**
4 : $b_i \leftarrow 1$
5 : return b_i

Figure 4. The eligibility predicate

We assume there is a mechanism in place to register the public keys of the parties. This should ensure that the party does not create more public keys than the stake it owns. This bootstrapping ensures that the list of public keys is fixed before the execution of the protocol and all parties can see this list of public keys.

We require the following cryptographic primitives:

1) Non-interactive zero knowledge proofs that allow the operations NIZK.Setup, NIZK.Prove and NIZK.Verify.
2) A trapdoor permutation scheme that allows parties to evaluate a trapdoor permutation on an input x using their public key TRP.pk by evaluating $y = f_{\mathsf{TRP.pk}}(x)$. The parties can compute the inverse of y using the corresponding secret key TRP.sk by evaluating $x = f^{-1}_{\mathsf{TRP.sk}}(y)$.
3) A signature scheme that allows parties to sign a message using their secret key (SIG.sk) - $\sigma = \mathsf{SIG.sign}(\mathsf{SIG.sk}, m)$ and verification is done by $\mathsf{SIG.Ver}(\mathsf{SIG.pk}, \sigma, m)$
4) A commitment scheme that allows parties to commit to a message x, by computing $C = \mathsf{Com}(x, s)$ and a pseudorandom function F, that parties can evaluate using their secret key PRF.sk by computing $C^{prf} = F(\mathsf{PRF.sk}, x)$

We now describe the different steps of the protocol:

Setup(1^λ) : The public parameters pp contain the common reference string of NIZK, $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and a public list L initialized to $\emptyset$.

Initialization(P_i) (Key Generation) : P_i runs the key generation algorithm, KeyGen, as soon as he registers with the network. KeyGen takes as input the public parameters pp. For each unit of stake that P_i owns, it does the following : Run KeyGen to output a PRF secret key $\mathsf{PRF.sk}_i \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$, compute a commitment to the PRF secret key $C_i^{prf} = \mathsf{Com}(\mathsf{PRF.sk}_i; s_{prf})$, a trapdoor permutation key pair $(\mathsf{TRP.pk}_i, \mathsf{TRP.sk}_i) \leftarrow \mathsf{TRP.KeyGen}(1^\lambda)$ and a signature key pair $(\mathsf{SIG.sk}_i, \mathsf{SIG.vk}_i) \leftarrow \mathsf{SIG.KeyGen}$. P_i then publishes $\mathsf{pk}_i = (\mathsf{TRP.pk}_i, \mathsf{SIG.vk}_i, C_i^{prf})$ and stores the secret key $\mathsf{sk}_i = (\mathsf{PRF.sk}_i, \mathsf{TRP.sk}_i, s_{prf}, \mathsf{SIG.sk}_i)$. The pk_i is published to a public list L. A Merkle tree, MTree(pk) with root $\mathsf{rt}_{\mathsf{pk}}$ is created with this list

$$L = \{(C_i^{prf}, \mathsf{SIG.vk}_1, \mathsf{TRP.pk}_1), \cdots, (C_n^{prf}, \mathsf{SIG.vk}_n, \mathsf{TRP.pk}_n)\}$$

and can be viewed by all parties in the world. The initialization protocol is described in Figure 6.

EligibilityCheck(P_i, tag) : For each tag P_i runs the ProcessRO algorithm (see Figure 5) to compute a vector $\vec{V}_{\mathtt{tag}} = (\vec{V}_{\mathtt{tag}}[1], \vec{V}_{\mathtt{tag}}[2], \cdots, \vec{V}_{\mathtt{tag}}[n])$, where n is the total number of keys (also equal to totalStake, since one key is generated for one unit of stake) in the system. The $\vec{V}_{\mathtt{tag}}$ is stored as a Merkle tree, $\mathsf{MTree}(\vec{V}_{\mathtt{tag}})$ with root $\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}$. The idea is that for each key, P_i owns in MTree(pk), there is a corresponding $\vec{V}_{\mathtt{tag}}[i]$ in the same position in $\mathsf{MTree}(\vec{V}_{\mathtt{tag}})$. This vector $\vec{V}_{\mathtt{tag}}$ is computed for each tag and serves as a trapdoor permutation whose inverse (v_i) is computed by the party. Only party P_i can compute the inverse of this permutation since only P_i knows the trapdoor secret key. To ensure that the party uses the correct secret key, we require

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

that the position of $\vec{V}_{\mathtt{tag}}[i]$ and the tuple containing $\mathsf{TRP.pk}_i$ are the same in the corresponding merkle trees.

Protocol ProcessRO($\mathtt{tag}$)

The algorithm takes as input a $\mathtt{tag}$ and does the following:

- For all $i \in [n]$:
  1) Query $(\mathsf{Eval}, \mathtt{tag} \| i)$ to the ideal functionality $\mathcal{F}_{\mathsf{RO}}$.
  2) Receive message $(\mathsf{Eval}, \mathtt{tag} \| i, V_i)$ from $\mathcal{F}_{\mathsf{RO}}$.
- Output a vector $\vec{V}$ where each element $\vec{V}[i] = V_i$.

Figure 5. ProcessRO algorithm

Protocol Initialization(P_i, sid)

1 : Generate $(\mathsf{TRP.pk}_i, \mathsf{TRP.sk}_i) \leftarrow \mathsf{TRP.KeyGen}(1^\lambda)$
2 : Generate $(\mathsf{PRF.sk}_i) \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$
3 : Generate $(\mathsf{SIG.pk}_i, \mathsf{SIG.sk}_i) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$
4 : Sample $s \leftarrow \{0,1\}^\lambda$ and compute $C_i^{prf} := \mathsf{Com}(s, \mathsf{PRF.sk}_i)$

The protocol publishes the variables : $\mathsf{pk}_i := (\mathsf{TRP.pk}_i, \mathsf{SIG.vk}_i, C_i^{prf})$ as leaves of $\mathsf{MTree(pk)}$ and returns $\mathsf{sk}_i = (\mathsf{TRP.sk}_i, \mathsf{SIG.sk}_i, (s, \mathsf{PRF.sk}_i))$ to P_i.

Figure 6. Initialization protocol for $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$

Using this $\vec{V}_{\mathtt{tag}}[i]$ P_i computes

$$v_i = f^{-1}_{\mathsf{TRP.sk}_i}(\vec{V}_{\mathtt{tag}}[i]). \tag{1}$$

P_i then evaluates the Eligible function as shown in Figure 4 to check if the party can speak for item $\mathtt{tag}$. The eligibility check is shown in Figure 7.

Protocol EligibilityCheck($P_i, sid, \mathtt{tag}$)

1 : Call $\mathsf{processRO}(\mathtt{tag})$ and receive $\vec{V}_{\mathtt{tag}}$
2 : Compute $v_i = f^{-1}_{\mathsf{TRP.sk}_i}(\vec{V}_{\mathtt{tag}}[i])$
3 : Call $\mathsf{Eligible}(v_i, \mathsf{stake}_i, \mathtt{tag})$ and receive $b_{\mathtt{tag}}$
4 : Output $(b_{\mathtt{tag}}, v_i, \vec{V}_{\mathtt{tag}})$.

Figure 7. Eligibility check for $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$

$\mathsf{CreateProof}(P_i, \mathtt{tag}, msg_i, v_i, \vec{V}_{\mathtt{tag}})$: If P_i is eligible to speak for item $\mathtt{tag}$ it commits to its winning ticket v_i. The commitment is implemented using a pseudorandom function F and is constructed as follows : $C_i^v = F(\mathsf{PRF.sk}_i, v_i \| \mathtt{tag})$.

We are required to hide the value v_i so that identity of the party is not revealed. (One may simply run $f_{\mathsf{TRP.pk}_i}(v_i)$ for all identities and identify who sent the message).

At the same time we require the commitment to be deterministic, else a malicious party may speak multiple times in the same $\mathtt{tag}$, with the same v_i by just using a different randomness each time. To ensure that a malicious party cannot send multiple (potentially conflicting) messages, we require the commitment to be deterministic and hence use a PRF.

P_i then constructs a NIZK that proves the following statements:

- "I know v_i, such that $\vec{V}_{\mathtt{tag}}[i] = f_{\mathsf{TRP.pk}_i}(v_i)$"
- "I am eligible to speak for $\mathtt{tag}$ according to randomness v_i."
- "C_i^v is correctly computed as $F(\mathsf{PRF.sk}_i, v_i \| \mathtt{tag})$"
- "I know the path from pk_i which is the leaf of a Merkle tree $\mathsf{MTree(pk)}$, that contains commitment of my PRF secret key, trapdoor public key and signature verification key, to the root of the Merkle tree."
- "I know the path from $\vec{V}_{\mathtt{tag}}[i]$, which is the leaf of a Merkle tree $\mathsf{MTree}(\vec{V}_{\mathtt{tag}})$ that contains all the elements in vector $\vec{V}_{\mathtt{tag}}$, to the root of the Merkle tree."

More formally, the NIZK statement and proof for the language $\mathcal{L}$ characterized by the relation $\mathcal{R}$ is computed as follows:

$$\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w) \tag{2}$$

- statement $x = (\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{pk}, \mathtt{tag}, msg, C_i^v, \vec{V}_{\mathtt{tag}})$.
- witness $w = (i, \mathsf{PRF.sk}_i, v_i, \sigma, s_{prf}, \mathsf{pk}_i, \mathsf{path}_{\mathsf{pk}}, \mathsf{path}_{\vec{V}_{\mathtt{tag}}})$, where $\mathsf{pk}_i = (\mathsf{TRP.pk}_i, \mathsf{Sig.vk}_i, C_{prf})$.
- $\mathcal{R}(x, w) = 1$ if and only if:
  1) $C_i^v = F(\mathsf{PRF.sk}_i, v_i \| \mathtt{tag})$
  2) $C_i^{prf} = \mathsf{Com}(\mathsf{PRF.sk}_i; s_{prf})$
  3) $V_i = f_{\mathsf{TRP.pk}_i}(v_i)$
  4) $V_i = \vec{V}_{\mathtt{tag}}[i]$
  5) $\mathsf{Eligible}(v_i, \mathsf{stake}_i, \mathtt{tag}) = 1$
  6) $\sigma = \mathsf{SIG.Sign}(\mathsf{SIG.sk}_i, msg \| \mathtt{tag})$
  7) $\mathsf{SIG.Ver}(\mathsf{SIG.vk}_i, \sigma, msg \| \mathtt{tag}) = 1$
  8) $\mathsf{validPath}_h(\mathsf{path}_{\mathsf{pk}}, \mathsf{rt}_{pk}, \mathsf{pk}_i) = 1$
  9) $\mathsf{validPath}_h(\mathsf{path}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \vec{V}_{\mathtt{tag}}[i]) = 1$

The protocol for creating the proof is shown in Figure 8.

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

Protocol CreateProof$(P_i, sid, msg_i, \mathtt{tag}, v_i, \vec{V}_{\mathtt{tag}})$

1 : Compute $C_i^v = F(\mathsf{PRF.sk}_i, v_i \| \mathtt{tag})$
2 : Let $\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}$ be the root of $\mathsf{MTree}(\vec{V}_{\mathtt{tag}})$
3 : Let $\mathsf{path}_{\vec{V}_{\mathtt{tag}}[i]}$ be the path to $\vec{V}_{\mathtt{tag}}[i]$ in $\mathsf{MTree}(\vec{V}_{\mathtt{tag}})$
4 : Let $\mathsf{rt}_{\mathsf{pk}}$ be the root of $\mathsf{MTree}(\mathsf{pk})$
5 : Let $\mathsf{path}_{\mathsf{pk}_i}$ be the path to pk_i in $\mathsf{MTree}(\mathsf{pk})$.
6 : Compute $\sigma_i = \mathsf{SIG.Sign}(\mathsf{SIG.sk}_i, msg_i \| \mathtt{tag})$
7 : Let $x = (\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{\mathsf{pk}}, \mathtt{tag}, msg_i, C_i^v)$
8 : Let $w = (i, \mathsf{PRF.sk}_i, v_i, \mathsf{pk}_i, \mathsf{path}_{\mathsf{pk}_i}, \mathsf{path}_{\vec{V}_{\mathtt{tag}}[i]}, \sigma_i)$
9 : Compute $\pi_{\mathsf{NIZK}} := \mathsf{NIZK.Prove}(\mathsf{crs}, x, w)$
as shown in Equation 2
10 : Set $\pi_i := (\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{\mathsf{pk}}, C_i^v, \pi_{\mathsf{NIZK}})$ and output π_i

Figure 8. Creating a proof in $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$

$\underline{\mathsf{Verify}(\mathtt{tag}, msg, \pi)}$: Party P_i on receiving a message from another party first runs ProcessRO algorithm to compute $\vec{V}_{\mathtt{tag}}$. The zero knowledge proof π is parsed as $(\mathsf{rt}_{\vec{V}}, \mathsf{rt}_{pk}, C, \pi_{\mathsf{NIZK}})$. P_i then sets the statement x to be $(\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{pk}, \mathtt{tag}, msg, C)$ and checks if NIZK.Verify $(\mathsf{crs}, x, \pi_{\mathsf{NIZK}}) = 1$. If it checks out then P_i accepts the message, else it rejects the message. The protocol for verification of a message is shown in Figure 10.

The overall protocol is described below in Figure 9.

Protocol $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}(sid)$

A party P_i executes the protocol $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ in the following way:

1 : Call Initialization(P_i, sid) to get $(\mathsf{pk}_i, \mathsf{sk}_i)$
2 : To publish a message msg_i in tag :
3 : Call EligibilityCheck$(P_i, sid, \mathtt{tag})$ to get $b_{\mathtt{tag}}, \vec{V}_{\mathtt{tag}}$ and v_i.
4 : **if** $b_{\mathtt{tag}} = 1$ **then**
call CreateProof$(P_i, sid, msg_i, \mathtt{tag}, v_i, \vec{V}_{\mathtt{tag}})$ to get π_i
5 : Output $(msg_i, \mathtt{tag}, \pi_i)$
6 : To verify a message$(msg, \mathtt{tag}, \pi)$ in tag :
7 : Call Verify$(sid, \mathtt{tag}, msg, \pi)$ and output the bit it returns.

Figure 9. Anonymous Selection protocol - $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$

Protocol Verify$(sid, \mathtt{tag}, msg, \pi)$

1 : Call ProcessRO(tag) and receive $\vec{V}_{\mathtt{tag}}$
2 : Parse $\pi = (\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{\mathsf{pk}}, C, \pi_{\mathsf{NIZK}})$
3 : Set $x = (\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{\mathsf{pk}}, C, \mathtt{tag}, msg)$
4 : Check that $\mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi_{\mathsf{NIZK}}) \stackrel{?}{=} 1$
5 : If yes, output 1; else output 0

Figure 10. Verifying a proof in $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$

V. Proof(Sketch)

Theorem 1. *The protocol $\pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ (Fig. 9) UC-realizes the $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ functionality (Fig. 1) in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model, assuming anonymous multicast communications, secure pseudorandom functions, secure simulation-sound extractable NIZKs, trapdoor permutations and unforgeable signatures, in the presence of a PPT adversary.*

Overview of the Simulator. In order to prove UC-security we need to show that there exists a PPT simulator interacting with $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ that generates a transcript that is indistinguishable from the transcript generated by the real world adversary running the protocol $\pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$.

We first give a high-level description of the simulator (described in Figure 12 and Figure 13) - $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$. Our simulator leverages the programmability of the random oracle $\mathcal{F}_{\mathsf{RO}}$ and the extractability and simulatability of the underlying NIZK. Hence, the simulator $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ will make use of the NIZK simulators $(\mathcal{S}_1, \mathcal{S}_2)$ to correctly setup the CRS in simulation mode and to simulate NIZK proofs and the algorithm Extract to extract the witness from proofs received from the adversary.

$\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ first sets up a crs using $\mathcal{S}_1$. Then, for each honest party P_i present in the system (recall that we are in the static setting, so on the onset the simulator knows the set of honest parties), the simulator generates their public key: (TRP.pk,SIG.vk, C^{prf}). Differently from a honest key, C^{prf} is a commitment to 0 instead of the PRF secret key.

When the simulator receives (Prove,msg,tag) from the ideal functionality $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$, it must provide a proof for the pair $(msg, \mathtt{tag})$, even without knowing the identity of the party requiring this proof. The simulator will use the underlying zero-knowledge simulator $\mathcal{S}_2$ to compute the proof π and return it to the ideal functionality, as well as storing π in a list $\mathcal{L}$ of proofs computed so far.

The simulator detects whether a malicious party is attempting to learn if she is eligible to speak, by monitoring the queries to the random oracle $\mathcal{F}_{\mathsf{RO}}$. When the query has the form $(\mathtt{tag}, i)$ for a index i such that P_i is corrupted, the simulator will first query $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ with command EligibilityCheck to check if P_i is eligible to speak for tag.

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

If so, the simulator will program the random oracle with a value that makes P_i pass the selection function.

Finally, when the adversary sends Verify for a pair (msg, π), the simulator $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ first checks if this proof is in the list $\mathcal{L}$. If so, answers Verified to the party. Otherwise, the simulator checks if π is valid, by running NIZK.Verify. If the proof is valid, the simulator attempts to extract the witness using Extract. If the extraction fails, the simulator will abort with a message ExtractionFailure. Else, if the extracted witness contains key material from an honest party, then the simulator abort with a message SoundnessFailure. Else, the simulator sends CreateProof to $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ on input msg. If $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ replies Declined, then abort the protocol.

$\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ also simulates the random oracle, where for any query $(i, \mathtt{tag})$ where P_i belongs to the set of honest party, the simulator replies with a uniformly sampled random value.

We prove indistinguishability of the simulation through a series of hybrids. The crux of the proof is to show that the probability of the simulator aborting is negligible. In the following we summarize the failure events and give an intuition on why they happen with negligible probability.

- ROFailure - The simulator aborts with this message, if the output of a $\mathcal{F}_{RO}$ query from a malicious party is already stored in the table for some other previously queried value. We show that this happens with negligible probability in Lemma 1.
- ExtractionFailure - The simulator aborts with this message if it is unable to extract a witness from zero knowledge proof, using its simulated crs. This occurs with negligible probability since we assume simulation-extractable zero knowledge proofs.
- SoundnessFailure - The simulator aborts with this message if the extracted witness corresponds to that of an honest party, and the NIZK proof was not in the list of proofs that is maintained by the simulator. This would imply that the real-world adversary spoofed a valid witness for an honest party, which implies that the real-world adversary has either broken the one-wayness of the trapdoor permutation, unforgeability of the signature scheme, or the collision resistance of the hash function used in the Merkle tree.
- GetProofFailure - The simulator aborts with this message if the ideal functionality $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ replies with message Declined for a query that corresponds to an eligible party. This occurs with negligible probability if $\mathcal{F}_{RO}$ was programmed correctly and is shown in Lemma 5.

Finally, we stress that in our proof we will assume that all messages are exchanged via a secure anonymous multicast channel. For the full proof please refer to Appendix A.

VI. Extensions for adaptivity and the multi-stake setting

A. Correctness in Presence of Adaptive Adversaries

Our protocol in section IV-B assumes static corruption of parties. Below we give an intuition on how to achieve adaptive security for the correctness of the protocol, though not for anonymity (since we do not use adaptive secure NIZKs).

Ephemeral Keys of Algorand[7] - Algorand uses ephemeral keys and secure erasures to achieve adaptive security for the safety property. Parties sign a message in a step of a round with an ephemeral key and then erase this key as soon as they send their message.

A party P_i generates a master public key and master secret key (MPK_i, MSK_i) at initialization. Using the MSK, P_i generates ephemeral keys of the form $\mathsf{sk}_i^{r,s}$. Here $r \in [r' + 1, r' + 10^6]$ for some r' and $s \in [1, m]$ where m is the upper bound in number of steps in a round. Once the keys are generated, P_i erases MSK. P_i also erases $\mathsf{sk}_i^{r,s}$ at the end of the step. To verify a message signed using $\mathsf{sk}_i^{r,s}$ a party needs to know the MPK and r, s.

Adaptive secure protocol idea : To achieve adaptive security in our protocol we assume erasures and ephemeral signatures as in Algorand. We describe the modifications to the protocol below:

Initialization : A party P_i generates a signature key pair $(\mathsf{SIG.msk}_i, \mathsf{SIG.mvk}_i) \leftarrow \mathsf{SIG.KeyGen}$ and a fixed number (say t) of ephemeral secret keys $\{\mathsf{esk}_i^j\}_{j=1}^t$ such that for any j, $\mathsf{Verify}(\mathsf{SIG.mvk}_i, \mathsf{Sig}(\mathsf{esk}_i^j, m)) = 1$. P_i erases $\mathsf{SIG.msk}_i$ after computing these ephemeral keys. After t number of tag has elapsed, the party generates new signature keys. We assume each key $\mathsf{esk}^{\mathtt{tag}}$ is linked to an tag as in Algorand where - $(\mathsf{mvk}, \mathtt{tag})$ is used to verify a signature signed using $\mathsf{esk}^{\mathtt{tag}}$.

Create Proof : The party P_i now has to prove an additional statement which says - "I know a master public key that can verify a signature signed by an ephemeral key for a particular tag". More formally:

Statement : $x = (\mathsf{rt}_{\mathsf{pk}}, \mathtt{tag})$

Witness : $w = (\mathsf{esk}_i^{\mathtt{tag}}, \mathsf{pk}_i = (\cdot, \mathsf{SIG.mvk}_i, \cdot), \mathsf{path}_{\mathsf{pk}})$

Proof : $\mathsf{Verify}(\mathsf{SIG.mvk}_i, \mathsf{Sig}(\mathsf{esk}_i^{\mathtt{tag}}, m \| \mathtt{tag})) = 1$ and $\mathsf{validPath}_h(\mathsf{path}_{\mathsf{pk}}, \mathsf{rt}_{pk}, \mathsf{pk}_i) = 1$

Remark. We note that in order to achieve adaptive security, we need to make new assumptions and additionally pay higher computational costs. In terms of efficiency, each party now has to maintain a large number of ephemeral keys. They need to update these keys after a certain number of tag. In terms of assumptions we need to assume secure erasures and that all parties erase their keys after they use them in a specific tag. We do not have these issues if we assumed a weaker static adversary, but are necessary for an adaptive adversary.

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

Since we only claim adaptive security for the correctness of the protocol, the simulator $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ can simulate the protocol. The only change would be that the simulator now sets up a master verification key in the pk for all parties.

B. Multi-stake setting

Our protocol in section IV-B assumes that each pk_i is associated with exactly one unit of stake. In the following we give an intuition as to how we can associate multiple units of stake to each public key. The key idea is that if a party has multiple stake then the party is selected with a weight denoted wt_i, similar to the sortition algorithm shown in [19] (See Fig 11). The challenge we observe here is that we cannot reveal wt_i, since wt_i is proportional to the stake of the party. Therefore for a party P_i to publish a message it sends wt_i *unlinkable* proofs for the same message msg_i. We refer to each unit of wt as an index. We must ensure that the party P_i does not send more than wt_i messages nor does it send different messages with different proofs for the same index, else a malicious party could send more messages than wt_i.

Multi-stake protocol idea: We describe the modifications required of the protocol $\Pi^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ described above:

Initialization : We assume that parties create commitments to their stake, $\mathsf{cm}_i = \mathsf{Com}(v_i)$ and publish this commitment to create a merkle tree of coin commitments ($\mathsf{MTree(cm)}$ with root $\mathsf{rt_{cm}}$).

Eigibility Check: The function Eligible now returns $\mathsf{wt}_i^{\mathsf{tag}}$ instead of b_{tag}

Create Proof: We first modify the "deterministic commitment" - C_i^v as $C^v_{i,\mathsf{index}} = F(\mathsf{PRF.sk}_i, \|v_i\|\mathsf{tag}\|\mathsf{index})$. The party evaluates wt_i number of $C^v_{i,\mathsf{index}}$ (basically for each $\mathsf{index} \in [1, \mathsf{wt}_i]$. This ensures that a party can create exactly one proof for one index.

P_i now proves the following statements:

- "I know v_i, such that $\vec{V}_{\mathsf{tag}}[i] = f_{\mathsf{TRP.pk}_i}(v_i)$"
- "I know the path from cm_i which is the leaf of a Merkle tree $\mathsf{MTree(cm)}$, that contains commitment of my stake_i to the root of the Merkle tree."
- "I am eligible to speak for tag according to randomness v_i and stake stake_i with weight wt_i."
- "$C^v_{i,\mathsf{index}}$ is correctly computed as $F(\mathsf{PRF.sk}_i, v_i\|\mathsf{tag}\|\mathsf{index})$ and $\mathsf{index} \in [1, \mathsf{wt}_i]$"
- "I know the path from pk_i which is the leaf of a Merkle tree $\mathsf{MTree(pk)}$, that contains commitment of my PRF secret key, trapdoor public key and signature verification key, to the root of the Merkle tree."
- "I know the path from $\vec{V}_{\mathsf{tag}}[i]$, which is the leaf of a Merkle tree $\mathsf{MTree}(\vec{V}_{\mathsf{tag}})$ that contains all the elements in vector $\vec{V}_{\mathsf{tag}}$, to the root of the Merkle tree. "

Function $\mathsf{Eligible}(v_i, \mathsf{stake}_i, \mathsf{tag})$

Global variables for the protocol are totalStake and τ. totalStake defines the total stake of the parties in the network and τ is the expected number of parties to be selected (this depends on the tag). Here $B(k; \mathsf{stake}, p) = \binom{\mathsf{stake}}{k} p^k (1-p)^{\mathsf{stake}-k}$ and $\sum_{k=0}^{\mathsf{stake}} B(k; \mathsf{stake}, p) = 1$ as in [19]

$p \leftarrow \frac{\tau}{\mathsf{totalStake}}$
$\mathsf{wt}_i \leftarrow 0$
while $\frac{v_i}{2^{len(v_i)}} \notin [\sum_{k=0}^{\mathsf{wt}_i} B(k; \mathsf{stake}_i, p), \sum_{k=0}^{\mathsf{wt}_i+1} B(k; \mathsf{stake}_i, p))$
  $\mathsf{wt}_i = \mathsf{wt}_i + 1$
return wt_i

Figure 11. The eligibility function for multi stake

More formally, the NIZK statement and proof for the language $\mathcal{L}$ characterized by the relation $\mathcal{R}$ is computed as follows (we denote in red the differences in the statements we prove for the single stake setting):

$$\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w) \tag{3}$$

- statement $x = (\mathsf{rt}_{\vec{V}_{\mathsf{tag}}}, \mathsf{rt}_{pk}, \mathsf{rt_{cm}}, \mathsf{tag}, msg, C^v_{i,\mathsf{index}}, \vec{V}_{\mathsf{tag}})$
- witness $w = (i, \mathsf{wt}_i, \mathsf{stake}_i, \mathsf{index}, \mathsf{PRF.sk}_i, v_i, \sigma, s_{prf}, \mathsf{pk}_i, \mathsf{path_{pk}}, \mathsf{path}_{\vec{V}_{\mathsf{tag}}}, \mathsf{path_{cm}}, \mathsf{cm}_i)$, where $\mathsf{pk}_i = (\mathsf{TRP.pk}_i, \mathsf{Sig.vk}_i, C_{prf})$.
- $\mathcal{R}(x, w) = 1$ if and only if:
  1) $C^v_{i,\mathsf{index}} = F(\mathsf{PRF.sk}_i, v_i\|\mathsf{tag}\|\mathsf{index})$
  2) $\mathsf{index} \in [1, \mathsf{wt}_i]$
  3) $C_i^{prf} = \mathsf{Com}(\mathsf{PRF.sk}_i; s_{prf})$
  4) $\mathsf{cm}_i = \mathsf{Com}(\mathsf{stake}_i)$
  5) $V_i = f_{\mathsf{TRP.pk}_i}(v_i)$
  6) $V_i = \vec{V}_{\mathsf{tag}}[i]$
  7) $\mathsf{Eligible}(v_i, \mathsf{stake}_i, \mathsf{tag}) = \mathsf{wt}_i$
  8) $\sigma = \mathsf{SIG.Sign}(\mathsf{SIG.sk}_i, msg\|\mathsf{tag})$
  9) $\mathsf{SIG.Ver}(\mathsf{SIG.vk}_i, \sigma, msg\|\mathsf{tag}) = 1$
  10) $\mathsf{validPath}_h(\mathsf{path_{pk}}, \mathsf{rt}_{pk}, \mathsf{pk}_i) = 1$
  11) $\mathsf{validPath}_h(\mathsf{path}_{\vec{V}_{\mathsf{tag}}}, \mathsf{rt}_{\vec{V}_{\mathsf{tag}}}, \vec{V}_{\mathsf{tag}}[i]) = 1$
  12) $\mathsf{validPath}_h(\mathsf{path_{cm}}, \mathsf{rt_{cm}}, \mathsf{cm}_i) = 1$

Remarks. Note that this protocol does not realize the ideal functionality defined in Figure 1, instead it realizes a modified functionality for multi-stake presented in Appendix B. Note that for multiple stake we assume that we have commitments to stake as in Zerocash [9]. For this work we do not consider updates to stake.

The simulator $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ will be modified in the following way to simulate the multi-stake protocol. In initialization create a $\mathsf{cm}_i = \mathsf{Com}(0^\lambda; r)$ for each honest

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

party P_i and publish it. Create a merkle tree $\mathsf{MTree}(\mathsf{cm})$ with root $\mathsf{rt}_{\mathsf{cm}}$. The statement of the proofs created in response to Prove will now include $\mathsf{rt}_{\mathsf{cm}}$. For Verify, when the simulator extracts the witness it now includes stake_i, wt_i, $\mathsf{path}_{\mathsf{cm}}$ and cm_i. $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ now sends to $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ a message $(\mathsf{CreateProof}, sid, \mathtt{tag}, msg)$ at most wt_i times. If it receives $(\mathsf{Declined}, sid, \mathtt{tag}, msg)$ for any of these queries, output "GetProofFailure". In the simulation of $\mathcal{F}_{\mathsf{RO}}$ the simulator now receives wt_i when it sends $(\mathsf{EligibilityCheck}, sid, \mathtt{tag})$ to the $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$. The simulator then finds an r such that $\mathsf{Eligible}(r, \mathsf{stake}_i, \mathtt{tag}) = \mathsf{wt}_i$ (pick an r in the interval $[\sum_{k=0}^{\mathsf{wt}_i} B(k; \mathsf{stake}_i, p), \sum_{k=0}^{\mathsf{wt}_i+1} B(k; \mathsf{stake}_i, p))$)

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," no. 2012, p. 28, 2008.

[2] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Annual Cryptology Conference*, pp. 585–605, Springer, 2015.

[3] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, pp. 357–388, 2017.

[4] "Algorand blockchain." https://www.algorand.com.

[5] T. Kerber, A. Kiayias, M. Kohlweiss, and V. Zikas, "Ouroboros crypsinous: Privacy-preserving proof-of-stake," in *2019 2019 IEEE Symposium on Security and Privacy (SP)*, (Los Alamitos, CA, USA), pp. 984–1001, IEEE Computer Society, may 2019.

[6] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual International Cryptology Conference*, pp. 757–788, Springer, 2018.

[7] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019.

[8] C. Ganesh, C. Orlandi, and D. Tschudi, "Proof-of-stake protocols for privacy-aware blockchains," in *Advances in Cryptology – EUROCRYPT 2019*, vol. 11476 of *Advances in Cryptology – EUROCRYPT 2019*, pp. 690–719, Springer, 2019.

[9] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE S&P*, 2014.

[10] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 649–678, Springer, 2019.

[11] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2018.

[12] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pp. 781–796, 2014.

[13] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 305–326, Springer, 2016.

[14] C. Ganesh, C. Orlandi, and D. Tschudi, "Proof-of-stake protocols for privacy-aware blockchains." Cryptology ePrint Archive, Report 2018/1105, 2018.

[15] J. Groth, R. Ostrovsky, and A. Sahai, "New techniques for noninteractive zero-knowledge," *Journal of the ACM (JACM)*, vol. 59, no. 3, p. 11, 2012.

[16] A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. Papamanthou, R. Pass, s. abhi, and E. Shi, "Coco: A framework for building composable zero-knowledge proofs." Cryptology ePrint Archive, 2015.

[17] M. Bellare and M. Yung, "Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation," *Journal of Cryptology*, vol. 9, no. 3, pp. 149–166, 1996.

[18] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 2001 IEEE International Conference on Cluster Computing*, pp. 136–145, IEEE, 2001.

[19] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *ACM-OSP*, 2017.

[20] F. Baldimtsi, V. Madathil, A. Scafuro, and L. Zhou, "Anonymous lottery in the proof-of-stake setting." Cryptology ePrint Archive, Report 2020/533, 2020. https://eprint.iacr.org/2020/533.

APPENDIX

A. Security Proof

Proof:

Indistinguishability Proof. We prove that the real execution of the protocol in the $\mathcal{F}_{\mathsf{RO}}$-hybrid world is indistinguishable from the execution in the simulated world through a series of hybrids.

- Let the hybrid $\mathbf{H}_0$ denote the real world execution.
- Hybrid $\mathbf{H}_1$ is the same as $\mathbf{H}_0$ except that any calls to random oracle $\mathcal{F}_{RO}$ is replaced with simulated responses as shown in Figure 13. When the simulation aborts, it outputs "ROFailure". Note that $\mathbf{H}_0$ and $\mathbf{H}_1$ can be distinguished in the event of "ROFailure". We prove in Lemma 1 that $\mathbf{H}_0$ and $\mathbf{H}_1$ are indistinguishable, since the event "ROFailure" occurs with negligible probability.
- Hybrid $\mathbf{H}_2$ executes in the same way as $\mathbf{H}_1$, except that the crs is now replaced by a simulated crs and all honest proofs are now simulated.

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

$\mathbf{H}_1$:

$$\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$$

$$\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, stmt, w)$$

$\mathbf{H}_2$:

$$(\mathsf{crs}, ek, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$$

$$\pi \leftarrow \mathcal{S}_2(\mathsf{crs}, \tau, stmt)$$

$\mathbf{H}_2$ and $\mathbf{H}_1$ are computationally indistinguishable due to the computational zero-knowledge property of NIZK as proved in Lemma 2.

- Hybrid $\mathbf{H}_3$ executes in the same way as $\mathbf{H}_2$, except that the simulator extracts a witness for a message (msg, π) that was not generated by it. That is, it runs Extract (crs, ek, stmt, π_{NIZK}). If the output is $\perp$, then the experiments terminates with output "ExtractionFailure". We prove in Lemma 3 that this occurs with negligible probability, and hence $\mathbf{H}_3$ and $\mathbf{H}_2$ are computationally indistinguishable.
- Hybrid $\mathbf{H}_4$ works in the same way as $\mathbf{H}_3$, except that if Extract(crs, ek,stmt, π) = w, and the witness corresponds to an honest party whose π is not in $\mathcal{L}$, then the experiment terminates with output " SoundnessFailure". We prove by Lemma 4 that this happens with negligible probability, and hence $\mathbf{H}_4$ and $\mathbf{H}_3$ are computationally indistinguishable.
- Hybrid $\mathbf{H}_5$ works the same way as $\mathbf{H}_4$, except that the simulator sends $(\mathsf{CreateProof}, \mathtt{tag}, msg)$ to the ideal functionality (and when asked will later provide π). If the functionality replies with (Declined), output "GetProofFailure". We prove by Lemma 5 that this happens with negligible probability, which therefore implies that $\mathbf{H}_5$ and $\mathbf{H}_4$ are computationally indistinguishable.
- Hybrid $\mathbf{H}_6$ works the same way as $\mathbf{H}_5$, except that the outputs of the pseudorandom functions are replaced by totally random strings.

  $\mathbf{H}_5$:

  $$C_i^v = F(\mathsf{PRF.sk}_i, v_i \| \mathtt{tag})$$

  $\mathbf{H}_6$:

  $$C_i^v \leftarrow \{0,1\}^\lambda$$

  By the pseudorandomness property of pseudorandom functions as shown in Lemma 6, $\mathbf{H}_6$ and $\mathbf{H}_5$ are computationally indistinguishable.
- Hybrid $\mathbf{H}_7$ works the same way as $\mathbf{H}_6$, except that the commitments to PRF secret keys are replaced by commitments to the zero string.

  $\mathbf{H}_6$:

  $$C_i^{prf} \leftarrow \mathsf{Com}(\mathsf{PRF.sk}_i; s_{prf})$$

  $\mathbf{H}_7$:

  $$r \leftarrow \{0,1\}^\lambda$$

  $$C_i^{prf} \leftarrow \mathsf{Com}(0^\lambda; r)$$

  By the hiding property of the commitment scheme, as shown in Lemma 7, $\mathbf{H}_7$ and $\mathbf{H}_6$ are computationally indistinguishable.

Note that $\mathbf{H}_7$ is identical to the simulated world as described in Figure 12. By a summation over the previous hybrids we show that $\mathbf{H}_0 \approx \mathbf{H}_7$ by presenting the following supporting lemmas.

Lemma 1. *If $\mathcal{F}_{\mathsf{RO}}$ is modeled as random oracle then the event* ROFailure *happens with negligible probability.*

Proof: Recall from Figure 13 that when $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ receives a query x for $\mathcal{F}_{\mathsf{RO}}$ it parses $x = (\mathtt{tag}, i)$. Three cases may arise:

1) **Case 1:** P_i **is honest** : In this case, $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ simply outputs a random value y. This output is distributed identically to the output of $\mathcal{F}_{\mathsf{RO}}$.
2) **Case 2:** P_i **is malicious and eligible** : In this case $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ repeatedly samples a random value r until $\mathsf{Eligible}(r, \mathsf{stake}, \mathtt{tag}) = 1$. This is done by picking an r such that $r < p \cdot 2^{len(v)}$ (See Definition of of Eligible in Figure 4).
   Now, note that since $r = v_i$ is a random value, and since f is a permutation, it follows that V_i is also random and thus is distributed identically to the output of $\mathcal{F}_{\mathsf{RO}}$.
3) **Case 3:** P_i **is malicious and not eligible** : Similar argument as Case 2.

The bad case is when the simulator $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$ obtains a value $V = f(\mathsf{TRP.pk}, r)$ that was already provided in output for a previous $\mathcal{F}_{\mathsf{RO}}$ query (i.e., there exists a pair $(x', V) \in \mathcal{Q}$). In this case the simulator aborts and outputs ROFailure.

The probability of such event is $\frac{q}{2^\lambda}$, where q is the number of queries to the random oracle made by the adversary, which is negligible.

■

Lemma 2. *Assuming the zero-knowledge property of NIZK proof, hybrid $\mathbf{H}_1$ and hybrid $\mathbf{H}_2$ are computationally indistinguishable.*

Proof: Assuming that there exists a PPT adversary $\mathcal{A}_{12}$ such that :

$$Pr[\mathcal{A}(\mathbf{H}_1) = 1] - Pr[\mathcal{A}(\mathbf{H}_2) = 1] > p$$

then we can construct a PPT reduction $\mathcal{B}_{\mathsf{NIZK}}$ that uses $\mathcal{A}_{12}$ as a subroutine to break the zero-knowledge property of NIZK. We prove this by showing a challenger that interacts with the $\mathcal{B}_{\mathsf{NIZK}}$ adversary and outputs fail with negligible probability.

Challenger $\mathcal{C}_{\mathsf{NIZK}}$

1) Flip a coin $b \leftarrow \{0,1\}$. If $b = 0$: crs is generated by $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ else: $(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \mathcal{S}_1(1^\lambda)$

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

2) Give crs to $\mathcal{B}_{\mathsf{NIZK}}$ and get back b'. If $b = b'$ output fail

$\mathcal{B}_{\mathsf{NIZK}}(1^\lambda)$

1) Receive a common reference string crs from a NIZK challenger $\mathcal{C}_{\mathsf{NIZK}}$
2) Forward crs to the adversary $\mathcal{A}_{12}$ internally.
3) Execute the Initialization phase and ELIGIBLE phase for all honest parties as in $\mathbf{H}_1$.
4) In the CreateProof phase:
   - $\mathsf{stmt} = (\mathsf{rt}_{\vec{V}}, \mathsf{rt}_{\mathsf{pk}}, \mathtt{tag}, msg, C_i^v)$
   - $w = (i, \mathsf{PRF.sk}_i, v_i, \sigma_i, s_{prf}, \mathsf{pk}_i, \mathsf{path}_{\mathsf{pk}}, \mathsf{path}_{\vec{V}})$
   - Forward (stmt, w) to the NIZK challenger C_{NIZK}
5) Receive back a NIZK proof π_{NIZK}
6) Send $(msg, \mathtt{tag}, \pi = (\mathsf{rt}_{\vec{V}}, \mathsf{rt}_{\mathsf{pk}}, C_i^v, \pi_{\mathsf{NIZK}}))$ to $\mathcal{A}_{12}$
7) Output the bit b' received from $\mathcal{A}_{12}$

Note that when $b = 0$, the view of the adversary $\mathcal{A}_{12}$ is exactly the same as in hybrid $\mathbf{H}_1$. When $b = 1$, then the view of the adversary $\mathcal{A}_{12}$ is exactly the same as in hybrid $\mathbf{H}_2$.

From our hypothesis, $\mathcal{A}_{12}$ can distinguish the transcripts with non-negligible probability p, thus the reduction can use $\mathcal{A}_{12}$ to break the zero-knowledge property of NIZK with non-negligible probability. We thus we have a contradiction since we assumed a secure NIZK proof system. ■

Lemma 3. *If* NIZK *has simulation sound extractability, then "*ExtractionFailure*" happens with negligible probability.*

Proof: If $\mathsf{Extract}(\mathsf{crs}, \mathsf{stmt}, \pi)$ outputs a $w = \perp$, this immediately breaks the simulation sound extractability property of NIZK, since :

$$\Pr[(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \mathcal{S}_1(1^\lambda); (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(\mathsf{crs}, \tau, \cdot)}(\mathsf{crs}, \mathsf{ek}); w \leftarrow \mathsf{Extract}(\mathsf{crs}, \mathsf{ek}, \mathsf{stmt}, \pi) : \mathsf{stmt} \notin Q \wedge (\mathsf{stmt}, w) \notin \mathcal{L} \wedge \mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1] \approx$$

Therefore with only very negligible probability do we get the case " ExtractionFailure". ■

Lemma 4. *Assuming the security of trapdoor permutation* TRP*, EUF-CMA unforgeability property of signature scheme* SIG *and collision resistance of hash function H,* SoundnessFailure *happens with negligible probability.*

Simulator $\mathcal{S}_{\mathsf{Anon\text{-}Selection}}$

Setup : Run $\mathcal{S}_1$ to generate a simulated crs, trapdoor τ and extraction key ek.

Initialization : For each honest party P_i :
- Generate a valid $\mathsf{TRP.pk}_i$ by running $\mathsf{TRP.KeyGen}(1^\lambda)$.
- Generate a valid $\mathsf{SIG.vk}_i$ by running $\mathsf{SIG.KeyGen}(1^\lambda)$
- Sample a randomness r and commit to zero - $C^{prf} = \mathsf{Com}(0^\lambda; r)$.
- Publish $\mathsf{pk}_i = (\mathsf{TRP.pk}_i, \mathsf{SIG.vk}_i, C^{prf})$.

Generate a Merkle tree, $\mathsf{MTree}(\mathsf{pk})$ with all pk_i as leaves and obtain the root $\mathsf{rt}_{\mathsf{pk}}$.

CreateProof: Upon receiving $(\mathsf{Prove}, sid, msg, \mathtt{tag})$ from ideal functionality $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$.
- Compute a zero knowledge proof π_{ZK} for the message $(msg, \mathtt{tag})$, by calling the simulator $\mathcal{S}_2$ on $(\mathsf{crs}, \tau, \mathsf{stmt})$, where $\mathsf{stmt} = (\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{\mathsf{pk}}, \mathtt{tag}, msg, C)$, where C is sampled uniformly at random.
- Set $\pi = (\mathsf{rt}_{\vec{V}_{\mathtt{tag}}}, \mathsf{rt}_{\mathsf{pk}}, C, \pi_{\mathsf{ZK}})$ and store $(\pi, msg, \mathtt{tag})$ in a list $\mathcal{L}$.
- Send $(\mathsf{Done}, \pi, \mathtt{tag}, msg)$ to $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$.

Verify: Upon receiving $(\mathsf{Verify}, sid, \mathtt{tag}, msg, \pi)$ from a corrupted party.
- If $(\pi, msg, \mathtt{tag}) \in \mathcal{L}$, then send $(\mathsf{Verified}, sid, \pi, \mathtt{tag}, msg, 1)$.
- Else, compute $\vec{V}^* = \mathsf{ProcessRO}(\mathtt{tag})$ and create a Merkle tree $\mathsf{MTree}(\vec{V}_{\mathtt{tag}})$, with all $\vec{V}^*[i]$ as leaves and set the root of the Merkle tree as $\mathsf{rt}_{\vec{V}^*}$.
- Set $\mathsf{stmt} = (\mathtt{tag}, msg, C, \mathsf{rt}_{\vec{V}^*}, \mathsf{rt}_{pk})$
- If $\mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 0$, ignore the message.
- Else run $\mathsf{Extract}(\mathsf{crs}, \mathsf{stmt}, \pi, \mathsf{ek})$ to get w.
  - If $w = \perp$, output "ExtractionFailure"
  - Else let $w = (i, \mathsf{pk}_i, \mathsf{sk}_i, v_i, \sigma_i, s_{prf}, C_i^{prf}, \mathsf{path}_{\mathsf{pk}}, \mathsf{path}_{\vec{V}^*})$ be the extracted witness. Obtain identity i. If P_i is honest and π was not in $\mathcal{L}$, output "SoundnessFailure".
- Send $(\mathsf{CreateProof}, sid, \mathtt{tag}, msg)$ to $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ on behalf of P_i.
- If $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ replies with $(\mathsf{Prove}, sid, \mathtt{tag}, msg)$, then send $(\mathsf{Done}, sid, \pi, \mathtt{tag}, msg)$
- Else if $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ replies $(\mathsf{Declined}, sid, \mathtt{tag}, msg)$, output "GetProofFailure"

Figure 12. Simulator for $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

Simulating $\mathcal{F}_{\mathsf{RO}}$

- Initialize list $\mathcal{Q} = \emptyset$
- Upon receiving query x to $\mathcal{F}_{\mathsf{RO}}$ from some party P: if there exists $(x, y) \in \mathcal{Q}$, output y, Else parse $x = (\mathtt{tag}, i)$ and does the following.
  - if i such that $P_i \notin$ set of malicious parties, sample y randomly and store $\mathcal{Q} = \mathcal{Q} \cup (x, y)$.
  - else send $(\mathsf{EligibilityCheck}, sid, \mathtt{tag})$ to $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}^{\mathsf{Eligible}}$ on behalf P_i and obtain b.
    - find r such that $\mathsf{Eligible}(r, (P_i, \mathsf{stake}_i), \mathtt{tag}) = b$.
    - Compute $V = f(\mathsf{TRP.pk}_i, r)$.
    - If there exists $(x, V) \in \mathcal{Q}$, abort and output "ROFailure".
    - Set $y = V$, and store $(x, y) \in \mathcal{Q}$.
    - Output y.

Figure 13. Simulating random oracle queries

Proof: Recall that the difference between $\mathbf{H}_3$ and $\mathbf{H}_4$ is that in $\mathbf{H}_4$ when the simulator extracts a witness

$$w = (i, \mathsf{pk}_i, \mathsf{PRF.sk}_i, v_i, \sigma_i, s_{prf}, C_i^{prf}, \mathsf{path}_{\mathsf{pk}}, \mathsf{path}_{\vec{V}})$$

it outputs SoundnessFailure in the case of a bad event. The bad event being: the witness belongs to some honest party i and the proof π is not in the list $\mathcal{L}$. For this bad event to occur, there must exist an adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$, that can forge a valid proof for some honest party.

To prove that such a bad event occurs with negligible probability we construct a reduction which executes $\mathcal{A}_{\mathsf{SoundnessFailure}}$ as a subroutine to break at least one of the following properties.

- *Security of trapdoor permutation*. That is, we construct a reduction $\mathcal{B}_{\mathsf{trp}}$ that takes as input $(\mathsf{TRP.pk}, y)$ from the trapdoor permutation challenger $\mathcal{C}_{\mathsf{trp}}$ and it uses the adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$ as a subroutine to find the inversion x corresponding to y.
- *EUF-CMA unforgeability property of signature scheme*. That is, we construct a reduction $\mathcal{B}_{\mathsf{sig}}$ that takes as input a signature verification key $\mathsf{SIG.vk}$ from the signature challenger $\mathcal{C}_{\mathsf{sig}}$ and it uses $\mathcal{A}_{\mathsf{SoundnessFailure}}$ as a subroutine to find a valid forgery (m, σ), where $m = msg \| \mathtt{tag}$.
- *Collision-resistance of hash function*. That is, we construct a reduction $\mathcal{B}_{\mathsf{hash}}$ that uses $\mathcal{A}_{\mathsf{SoundnessFailure}}$ as a subroutine to find a collision of collision-resistant hash function and send it to a hash function challenger $\mathcal{C}_{\mathsf{hash}}$.

Claim 1. *Assuming that there exists an adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$ that can forge a valid proof for some honest party with non-negligible probability, then we can build a reduction $\mathcal{B}_{\mathsf{trp}}$ that uses $\mathcal{A}_{\mathsf{SoundnessFailure}}$ to break the one-wayness of the trapdoor permutation with non-negligible probability.*

Proof: We describe how the reduction $\mathcal{B}_{\mathsf{trp}}$ works. The reduction $\mathcal{B}_{\mathsf{trp}}$ takes as input a trapdoor permutation public key $\mathsf{TRP.pk}$ and a random value y from the trapdoor permutation challenger $\mathcal{C}_{\mathsf{trp}}$. The goal of $\mathcal{B}_{\mathsf{trp}}$ is to output a preimage of y.

$\underline{\mathcal{B}_{\mathsf{trp}}}$

1) It computes the simulated common reference string crs by computing $(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \mathcal{S}_1(1^\lambda)$ and forwards crs to $\mathcal{A}_{\mathsf{SoundnessFailure}}$.
2) Guess at random which index $i^* \in \mathcal{H}$, $\mathcal{A}_{\mathsf{SoundnessFailure}}$ will try to forge in the CreateProof phase, where $\mathcal{H}$ is the set of honest parties.
   a) Set $\mathsf{TRP.pk}_{i^*} = \mathsf{TRP.pk}$.
   b) Generate each public key pk_i for each honest party, except that when it computes public key pk_{i^*} for the party P_{i^*}, it includes the trapdoor public key $\mathsf{TRP.pk}_{i^*}$.
   c) Execute EligibilityCheck phase and simulate $\mathcal{F}_{RO}$ as in the previous hybrid but respond with the random string $y_{i^*} = y$ when the honest party P_{i^*} queries the random oracle.
   d) It runs the adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$ to obtain $(msg, \mathtt{tag}, \pi)$ from the CreateProof phase, then it parses $\pi = (\mathsf{rt}_{\mathsf{pk}}, \mathsf{rt}_{\vec{V}}, C, \pi_{\mathsf{NIZK}})$ and runs the extraction algorithm Extract to extract the witness $w \leftarrow \mathsf{Extract}(\mathsf{crs}, \mathsf{ek}, \mathsf{stmt}, \pi_{\mathsf{NIZK}})$, where
   $$w = (i, \mathsf{pk}_i, v_i, \sigma_i, s_{prf}, \mathsf{PRF.sk}_i, \mathsf{path}_{\mathsf{pk}}, \mathsf{path}_{\vec{V}})$$
   and i is the index of some honest party that the adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$ is trying to forge.
3) If the guess $i^* \neq i$, the reduction simply aborts and outputs a random string x to the challenger $\mathcal{C}_{\mathsf{trp}}$. Otherwise, the reduction $\mathcal{B}_{\mathsf{TRP}}$ forwards $x = v_i$ to the challenger $\mathcal{C}_{\mathsf{trp}}$ as the preimage of y.

When $\mathcal{A}_{\mathsf{SoundnessFailure}}$ can forge a valid proof for some honest party P_i with some non-negligible probability p, then the reduction $\mathcal{B}_{\mathsf{TRP}}$ can break the security of trapdoor permutation with non-negligible probability $p/|\mathcal{H}|$, which contradicts our assumption. ■

Claim 2. *Assuming that there exists an adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$ that can forge a valid proof for some honest party with non-negligible probability, then we can build a reduction $\mathcal{B}_{\mathsf{sig}}$ that uses $\mathcal{A}_{\mathsf{SoundnessFailure}}$ to break the EU-CMA unforgeability property of signature scheme with non-negligible probability.*

Proof: This proof is using standard techniques and for completeness we describe it in our full version [20]

■

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.

Claim 3. *Assuming that there exists an adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$ that can forge a valid proof for some honest party with non-negligible probability, then we can build a reduction $\mathcal{B}_{\mathsf{hash}}$ that uses the adversary $\mathcal{A}_{\mathsf{SoundnessFailure}}$ to find a collision of the collision-resistant hash function used in the Merkle tree with non-negligible probability.*

Proof: This proof is using standard techniques and for completeness we describe it in our full version [20] ■

Therefore, assuming $\mathcal{A}_{\mathsf{SoundnessFailure}}$ can forge a valid proof for some honest party with non-negligible probability, then we can construct a reduction to break at least one of the following properties: security of trapdoor permutation, unforgeability of signature and collision resistance of hash function. However, this contradicts to the assumption we made. Therefore this completes the proof of the lemma. ■

Lemma 5. *If the simulation of F_{RO} is correctly programmed, "*$\mathsf{GetProofFailure}$*" occurs with negligible probability.*

Proof: "GetProofFailure" is output only when $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ replies with $(\mathsf{Declined}, \mathtt{tag}, msg)$, when queried with the message $(\mathsf{CreateProof}, \mathtt{tag}, msg)$.
Note that $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ sends the Declined command only when $T(P_i, \mathtt{tag}) \neq 1$.
The ideal functionality $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$ sets $T(P_i, \mathtt{tag}) = 1$ only if $\mathsf{Eligible}((P_i, \mathsf{stake}_i), \mathtt{tag}, r) = 1$. This implies for P_i, the predicate Eligible returned 0 if it received Declined from the ideal functionality.
In $\mathbf{H}_1$ we show that the the simulator creates a y for malicious parties such that they are eligible or ineligible according to the predicate Eligible. Therefore if an extracted witness has v_i such that P_i is eligible to speak for $\mathtt{tag}$, then $\mathsf{Eligible}(y, (P_i, \mathsf{stake}_i), \mathtt{tag}) = 1$ and therefore the ideal functionality sets $T(P_i, \mathtt{tag}) = 1$. This implies $\mathcal{F}_{\mathsf{Anon\text{-}Selection}}$ will not send back Declined. We thus arrive at a contradiction.
Therefore the " GetProofFailure" occurs only with negligible probability. ■

Lemma 6. *Assuming pseudorandomness property of PRFs, the view of the adversary in hybrid $\mathbf{H}_5$ is indistinguishable from the view of the adversary in hybrid $\mathbf{H}_6$.*

Proof:
This proof is using standard techniques and for completeness we describe it in our full version [20] ■

Lemma 7. *Assuming the hiding property of commitment scheme* Com*, the view of the adversary in hybrid $\mathbf{H}_6$ is indistinguishable from the view of the adversary in hybrid $\mathbf{H}_7$.*

Proof:
This proof is using standard techniques and for completeness we describe it in our full version [20] ■

This completes the proof of the theorem. ■

B. The multi-stake ideal functionality

In this section we present an ideal functionality for anonymous lottery where parties are associated with multiple units of stake and not necessarily a single unit of stake.

$\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection\text{-}MS}}$

The ideal functionality is parameterized by an Eligible predicate and maintains the following elements: (1) A global set of registered parties $\mathcal{P} = ((P_1, \mathsf{stake}_1), \ldots, (P_n, \mathsf{stake}_n))$. (2) A table T, which has one row per party and a column for each tag given by parties when checking eligibility. The table stores a tuple (wt_i, wt'_i) of each party in each tag. (3) A list $\mathcal{L}$, to store a proof π corresponding to a message msg in some tag.

- Upon receiving $(\mathsf{EligibilityCheck}, sid, \mathtt{tag})$ from a party $P_i \in \mathcal{P}$ do the following :
  1) If $P_i \in \mathcal{P}$ and $T(P_i, \mathtt{tag})$ is undefined, sample a random number $r \in \{0,1\}^\ell$ run $\mathsf{Eligible}(r, (P_i, \mathsf{stake}_i), \mathtt{tag})$ to get a weight value wt_i. Set $T(P_i, \mathtt{tag}) = (\mathsf{wt}_i, \mathsf{wt}'_i = \mathsf{wt}_i)$
  2) Output $(\mathsf{EligibilityCheck}, sid, T(P_i, \mathtt{tag}))$ to P_i.
- Upon receiving $(\mathsf{CreateProof}, \mathtt{tag}, msg, \mathsf{wt}_j)$ from $P_i \in \mathcal{P}$
  1) Get $(\mathsf{wt}_i, \mathsf{wt}'_i)$ from $T(P_i, \mathtt{tag})$. If $\mathsf{wt}_i > 0$ and $\mathsf{wt}'_i - \mathsf{wt}_j \geq 0$, send $(\mathsf{Prove}, \mathtt{tag}, msg)$ to $\mathcal{A}$ for each $k \in [1, \mathsf{wt}_j]$. Else, send $(\mathsf{Declined}, \mathtt{tag}, msg)$ to P_i.
  2) Upon receiving $(\mathsf{Done}, \psi_k, \mathtt{tag}, msg)$ from $\mathcal{A}$ for each k. Set $\pi_k \leftarrow \psi_k$ and record $(\pi_k, \mathtt{tag}, msg)$ in $\mathcal{L}$ for each $k \in [1, \mathsf{wt}_j]$. Send $(\mathsf{Proof}, \pi_k, \mathtt{tag}, msg)$ to P_i for each $k \in [1, \mathsf{wt}_j]$ and set $\mathsf{wt}'_i = \mathsf{wt}'_i - \mathsf{wt}_j$ in $T(P_i, \mathtt{tag})$
- Upon receiving $(\mathsf{Verify}, \pi, \mathtt{tag}, msg)$ from some party P':
  1) If $(\pi, \mathtt{tag}, msg) \in \mathcal{L}$ output $(\mathsf{Verified}, (\pi, \mathtt{tag}, msg), 1)$ to P'.
  2) If $(\pi, \mathtt{tag}, msg) \notin \mathcal{L}$, send $(\mathsf{Verify}, (\pi, \mathtt{tag}, msg)$ to $\mathcal{A}$ and wait for a reply w from the adversary $\mathcal{A}$. Check if w is valid If yes :
     - Extract $(P_i, \mathtt{tag}, msg)$ from w and check that $T(P_i, \mathtt{tag}) > 0$
     - If yes, store $(\pi, \mathtt{tag}, msg)$ in the list $\mathcal{L}$ and send $(\mathsf{Verified}, (\pi, \mathtt{tag}, msg), 1)$ to P'.

     If either of these checks are false output $(\mathsf{Verified}, (\pi, \mathtt{tag}, msg), 0)$ to P'.

Unlike the lottery functionality of $\mathcal{F}^{\mathsf{Eligible}}_{\mathsf{Anon\text{-}Selection}}$, here parties receive *a weight* wt when they check their eligibility. This wt is the number of messages they are allowed to propose with a proof. The ideal functionality maintains a variable wt'_i for each party P_i such that that for each message sent in a tag, the functionality decrements wt'_i by 1 (as long as $\mathsf{wt}' > 0$). This ensures that a party cannot create proofs for $> wt_i$ number of messages in one tag. The verification works like the single stake case, except that the functionality now checks if $T(P_i, \mathtt{tag}) > 0$ for a witness received from the adversary. We highlight the differences from the single stake ideal functionality in blue.

Authorized licensed use limited to: George Mason University. Downloaded on September 02,2020 at 13:47:53 UTC from IEEE Xplore. Restrictions apply.