Proof-of-Stake Protocols for Privacy-Aware Blockchains

Chaya Ganesh[1⋆], Claudio Orlandi[1⋆], and Daniel Tschudi[1,2⋆]

{ganesh,orlandi,tschudi}@cs.au.dk

[1] Department of Computer Science, DIGIT, Aarhus University
[2] Concordium

Abstract. *Proof-of-stake (PoS)* protocols are emerging as one of the most promising alternative to the wasteful *proof-of-work (PoW)* protocols for consensus in Blockchains (or distributed ledgers). However, current PoS protocols inherently disclose both the *identity* and the *wealth* of the stake-holders, and thus seem incompatible with privacy-preserving cryptocurrencies (such as ZCash, Monero, etc.). In this paper we initiate the formal study for PoS protocols with privacy properties. Our results include:

1. A (theoretical) feasibility result showing that it is possible to construct a general class of *private PoS (PPoS)* protocols; and to add privacy to a wide class of PoS protocols,
2. A privacy-preserving version of a popular PoS protocol, Ouroboros Praos.

Towards our result, we define the notion of *anonymous verifiable random function*, which we believe is of independent interest.

1 Introduction

Popular decentralized cryptocurrencies like Bitcoin [Nak08] crucially rely on the existence of a distributed ledger, known as the *Blockchain*. The original protocols used to build and maintain the Blockchain were based on *proof-of-work consensus protocols (PoW)*. While blockchain protocols mark a significant breakthrough in distributed consensus, reliance on expensive PoW components result in enormous waste of energy [OM14, CDE$^+$16], therefore it is an important open problem to find alternative consensus mechanisms which are less wasteful than PoW but at the same time maintain the positive features offered by PoW. *Proof-of-stake consensus protocols (PoS)* are one of the most promising technology to replace PoW and still preserve similar robustness properties: while PoW provides robustness assuming that a (qualified) majority of the *computing power* is honest, PoS instead relies on the assumption that a majority of the *wealth* in the system is controlled by honest participants. The rationale behind PoS is that users who have significant *stakes* in the system have an economic incentive in keeping the system running according to the protocol specification, as they risk that their stakes will become worthless if trust in the cryptocurrency vanishes. As is usual in this research space, the initial idea of proof-of-stake appeared informally in an online Bitcoin forum [bit11], and since then, there have been a series of candidates for such protocols [KN12, BLMR14, BGM16]. Recently, there have been works on formal models for proof-of-stake and protocols with provable security guarantees [BPS16, KRDO17, GHM$^+$17, DGKR18, BGK$^+$18].

Consensus based on lottery. Very informally, a lottery-based consensus protocol works in the following way: some publicly verifiable "lottery" mechanism is implemented to elect the next committee or a block "leader" who is then allowed to add the next block to the blockchain. The probability of a user being elected is proportional to the amount of some "scarce resource" owned by the user. In PoW, the probability is proportional to the computing power of the user, while in PoS it is proportional to the amount of coins the user owns. Since the resource is scarce and cannot be replicated, *Sybil attacks* are prevented (e.g., a user cannot inflate its probability of become the block leader). This, combined with the assumption that some majority of the resource is controlled by honest parties guarantees that the honest participants are in charge of the blockchain, thus guaranteeing integrity of the stored information.

Proof-of-work. In PoW, such as Bitcoin, for every block, all users try to solve some computationally challenging puzzle. The first user to have solved the puzzle publishes the solution together with the new

⋆ This work was supported by the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC), the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation program under grant agreement No 669255 (MPCPRO) and No 803096 (SPEC), and the Concordium Blockchain Research Center, Aarhus University, Denmark.

block, and often together with a new address that allows to collect the transaction fees and the block reward, which act as economic incentives for users to participate in the consensus protocol. All other participants can verify that the received block is valid (e.g., contains a valid solution to the puzzle) and, if so, append it to their local view of the Blockchain. Note that, if we assume that users are connected to each other using anonymous communication channels (e.g., Tor), then PoW provides full anonymity i.e., given two blocks it is hard to tell whether they came from the same user or not.

Proof-of-stake. On the other hand, PoS systems follow a different approach: here we must assume that the Blockchain contains information about the wealth owned by the users in the system. Then, for every block, each user has a way to locally compute (using a pseudorandom process) whether they won the lottery. The lottery has the property that the higher the stakes in the system, the higher the probability of becoming the block leader or committee member. If a user wins, then, in the case of block leadership, they publish a block together with a proof of winning the lottery, and in the case of committee membership publish a message along with proof of lottery win. As in PoW, it is important that the other users in the system can efficiently verify the correctness of this claim, that is, the user claiming to be the block leader has in fact won the lottery. Unfortunately, in existing PoS protocols, this requires the users to be able to link the newly generated block with some account in the system. Thus, everyone in the system will learn the identity of the block producer (and their wealth).

Privacy in PoS? As privacy-preserving cryptocurrencies (such as ZCash, Monero, Dash, etc.) increase in popularity, it is natural to ask the following question:

Is it possible to design consensus protocols which
are as energy-efficient as PoS, but as private as PoW?

In this paper we address this problem and provide the first positive results in this direction. In particular we offer two contributions: 1) we provide a feasibility result showing that it is possible to construct a *Private PoS (PPoS)* protocol, that is one where the identity of the lottery winner (and their wealth) is kept secret by the protocol; and 2) we show how to adapt a popular PoS protocol, Ouroboros Praos [DGKR18], to satisfy our anonymity requirement. In doing so, we introduce a novel cryptographic primitive – *anonymous verifiable random function (AVRF)* which might be of independent interest.

Related Work. In a recent independent and concurrent work [KKKZ18], the authors present a privacy-preserving proof-of-stake protocol. While our work is more modular and treats privacy of proof-of-stake consensus independently of the cryptocurrency layer, the work of [KKKZ18] builds an overall private transaction ledger system. Additionally, our construction guarantees full privacy (at the cost of assuming anonymous channels), while [KKKZ18] allows for the *leakage* of a function of the stake.

1.1 Technical Overview

All current proof-of-stake proposals rely on stake distribution being available in the clear. As a first step, let us consider how to hide the wealth of the lottery winner: a simple idea that might come to mind is to encrypt the stakes on the blockchain, and to replace the proof of winning the lottery (e.g. in form of a correct block) with a proof that uses the encrypted stakes instead. However, this falls short of our goal in at least two ways. For instance, in case of block leaders: 1) it is still possible to distinguish (for instance) whether two blocks were generated by the same block leader or not and, crucially 2) since the probability of being elected as block leader is proportional to one's wealth, the frequency with which a user wins the lottery indirectly leaks information about their stakes (i.e., a user who is observed to win t out of n blocks has relative stakes in the system close to t/n). Thus, we conclude that even if block leaders are only interested in hiding their wealth, a PPoS must necessarily hide their identity as well. Similar concerns are true for committee based protocols as well, where the number of times a user becomes a committee member reveals information about their wealth.

In Section 4 we provide a framework for VRF-based *private stake lottery.* Our framework is parametrized by a lottery mechanism, and allows therefore to construct PPoS protocols for some of the most popular lottery mechanisms used in current (non-private) PoS protocols (e.g., Algorand [GHM$^+$17], Ouroboros Praos [DGKR18], etc.).

After having established the first feasibility result in this area, in Section 5 we investigate how to efficiently implement PPoS. Our starting point is one of the main PoS candidates which comes with a rigorous proof of security, namely Ouroboros Praos.

In a nutshell, Ouroboros Praos works as follows: Every user in the system registers a verification key for a *verifiable random function* e.g., a PRF for which it is possible to prove that a given output is in the image of the function relative to the verification key. Then, at every round (or slots), users can apply the VRF to the slot number and thus receive a (pseudorandom) value. If the value is less than (a function of) their wealth, then that user has won the election process and can generate a new block. Thanks to the VRF property, all other users can verify that the VRF has been correctly computed, and since the wealth of the user is public as well, every other user can compare the output of the VRF with the (function of) the user's wealth.

Using the private stake lottery of Section 4, it would be possible for the elected leader to prove correctness of all steps above (without revealing any further information) using the necessary zero-knowledge proofs. However, this would result in a very inefficient solution. To see why, we need to say a few more words about the winning condition of Ouroboros Praos: one of the goals of Ouroboros Praos is to ensure that a user cannot artificially increase their probability of winning the lottery, therefore Ouroboros Praos compares the output of the VRF with a function of the wealth that satisfies the "independent aggregation" property i.e., a function such that the probability that two users win the lottery is the same as the probability of winning for a single user who owns the same wealth as the two users combined. In particular, the function used by Ouroboros Praos has the form

$$\phi_f(x) = 1 - (1-f)^{\mathsf{stk}/\mathsf{Stake}}$$

where stk is wealth of the user, Stake is the total amount of stakes in the system and f is a difficulty parameter. Implementing such a function using the circuit representation required by zero-knowledge proofs would be very cumbersome, due to the non-integer division and the exponentiation necessary to evaluate ϕ. Finally, the variable difficulty level would require to update the circuit in the ZK-proof as the difficulty changes.

One of our insights is to exploit the "independent aggregation" property of the function for efficiency purpose, and in fact our solution uses the function ϕ in a completely *black-box* way, and thus allows to replace the specific function above with any other function that satisfies the "independent aggregation" property. Thanks to the independent aggregation property, we can let the users commit to their wealth in a bit-by-bit fashion, thus effectively splitting their account into a number of "virtual parties" such that party i has wealth 0 or 2^i. Then, the values $V_i = \phi(2^i)$ can be publicly computed (outside of the ZK-proof) and what is left to do for the user is to prove that the output of the VRF (for at least one of the virtual parties i), is less than than the corresponding public value V_i (without revealing which one).

The solution as described so far allows to prove that one has won the election for a "committed" stake but, as described above, the frequency with which an account wins the election reveals information about the user's wealth as well. Therefore we need to replace the VRF with an "anonymous VRF" or AVRF. In a nutshell, an AVRF is a VRF in which there exist multiple verification keys for the same secret key, and where it is hard, given two valid proofs for different inputs under different verification keys, to tell whether they were generated by the same secret key or not. We show that it is possible to turn existing efficient VRF constructions into anonymous VRF with an very small efficiency loss, and we believe that AVRF is a natural cryptographic primitive which might have further applications.

2 Preliminaries

Notation. We use $[1, n]$ to represent the set of numbers $\{1, 2, \ldots, n\}$. If A is a randomized algorithm, we use $y \leftarrow \mathsf{A}(x)$ to denote that y is the output of A on x. We write $x \overset{R}{\leftarrow} \mathcal{X}$ to mean sampling a value x uniformly from the set $\mathcal{X}$. We write PPT to denote a probabilistic polynomial-time algorithm. Throughout the paper, we use κ to denote the security parameter or level. A function is negligible if for all large enough values of the input, it is smaller than the inverse of any polynomial. We use negl to denote a negligible function. We denote by H a cryptographic hash function.

2.1 Zero-knowledge proofs

Let R be an efficiently computable binary relation which consists of pairs of the form (x, w) where x is a statement and w is a witness. Let L be the language associated with the relation R, i.e., $L =$

$\{x \mid \exists w \text{ s.t. } R(x, w) = 1\}$. L is an NP language if there is a polynomial p such that every w in $R(x)$ has length at most $p(x)$ for all x.

A zero-knowledge proof for L lets a prover P convince a verifier V that $x \in L$ for a common input x without revealing w. A proof of knowledge captures not only the truth of a statement $x \in L$, but also that the prover "possesses" a witness w to this fact. This is captured by requiring that if P can convince V with reasonably high probability, then a w can be efficiently extracted from P given x.

Non-interactive Zero-knowledge Proofs. A model that assumes a trusted setup phase, where a string of a certain structure, also called the public parameters of the system is generated, is called the common reference string (CRS) model. Non-interactive zero-knowledge proofs (NIZKs) in the CRS model were introduced in [BFM88]. We give a formal definition of NIZKs in AppendixB.1. In this paper, we will be concerned with non-interactive proofs.

2.2 Commitment schemes

A commitment scheme for a message space is a triple of algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ such that $\mathsf{Setup}(1^\kappa)$ outputs a public commitment key; Com given the public key and a message outputs a commitment along with opening information. Open given a commitment and opening information outputs a message or $\perp$ if the commitment is not valid. We require a commitment scheme to satisfy correctness, hiding and binding properties. Informally, the hiding property guarantees that no PPT adversary can generate two messages such that it can distinguish between their commitments. The binding property guarantees that, informally, no PPT adversary can open a commitment to two different messages.

2.3 Sigma Protocols

A sigma protocol for a language L is a three round public-coin protocol between a prover P and a verifier V. P's first message in a sigma protocol is denoted by $a \leftarrow P(x; R)$. V's message is a random string $r \in \{0, 1\}^\kappa$. P's second message is $e = P(w, a, r, R)$. (a, r, e) is called a transcript, and if the verifier accepts, that is $V(x, a, r, e) = 1$, then the transcript is $\mathsf{accepting}$ for s.

Definition 1 (Sigma protocol). *An interactive protocol $\langle P, V\rangle$ between prover P and verifier V is a Σ protocol for a relation R if the following properties are satisfied:*

1. *It is a three move public coin protocol.*
2. *Completeness: If P and V follow the protocol then* $\Pr[\langle P(w), V\rangle(x) = 1] = 1$ *whenever $(x, w) \in R$.*
3. *Special soundness: There exists a polynomial time algorithm called the extractor which when given s and two transcripts (a, r, e) and (a, r', e') that are accepting for s, with $r \neq r'$, outputs w' such that $(x, w') \in R$.*
4. *Special honest verifier zero knowledge: There exists a polynomial time simulator which on input s and a random r outputs a transcript (a, r, e) with the same probability distribution as that generated by an honest interaction between P and V on (common) input s.*

Sigma protocols and NIZK. The Fiat-Shamir transform [FS87] may be used to compile a Σ protocol into a non-interactive zero-knowledge proof of knowledge in the random oracle model. In this paper, we will be concerned with transformations in the CRS model [Dam00, Lin15]. The transformation of [Dam00] gives a 3-round concurrent zero-knowledge protocol, while [Lin15] is non-interactive.

OR composition of Σ-protocols. In [CDS94], the authors devise a composition technique for using sigma protocols to prove compound OR statements. Essentially, a prover can efficiently show $((x_0 \in \mathcal{L}) \vee (x_1 \in \mathcal{L}))$ without revealing which x_i is in the language. More generally, the OR transform can handle two different relations R_0 and R_1. If Π_0 is a Σ-protocol for R_0 and Π_1 a Σ-protocol for R_1, then there is a Σ-protocol Π_{OR} for the relation R_{OR} given by $\{((x_0, x_1), w) : ((x_0, w) \in R_0) \vee ((x_1, w) \in R_1)\}$.

Pedersen Commitment. Throughout the paper, we use an algebraic commitment scheme that allows proving polynomial relationships among committed values. The Pedersen commitment scheme [Ped92] is one such example that provides computational binding and unconditional hiding properties based on the discrete logarithm problem. It works in a group of prime order q. Given two random generators g and h such that $\log_g h$ is unknown, a value $x \in \mathbb{Z}_q$ is committed to by choosing r randomly from $\mathbb{Z}_q$, and computing $C_x = g^x h^r$. We write $\mathsf{Com}_q(x)$ to denote a Pedersen commitment to x in a group of order q. There are Sigma protocols in literature to prove knowledge of a committed value, equality of two committed values, and so on, and these protocols can be combined in natural ways. In particular, Pedersen commitments allow proving polynomial relationships among committed values: Given $\mathsf{Com}(x)$ and $\mathsf{Com}(y)$, prove that $y = ax + b$ for some public values a and b. In our constructions, we make use of existing sigma protocols for proving statements about discrete logarithms, and polynomial relationships among committed values [Sch91, FO97, CS97, CM99]. Throughout, we use the following notation:

$$\mathsf{PK}\{(x, y, \ldots) : \textit{ statements} \text{ about } x, y, \ldots\}$$

In the above, $x, y, \ldots$ are secrets (discrete logarithms), the prover claims knowledge of $x, y, \ldots$ such that they satisfy *statements.* The other values in the protocol are public.

2.4 Merkle Tree

A Merkle tree is a hash based data structure that is used both to generically extend the domain of a hash function and as a succinct commitment to a large string. To construct a Merkle tree from a string $m \in \{0,1\}^n$, we split the string into blocks $b_i \in \{0,1\}^k$. Each block is then a leaf of the tree, and we use a hash function H to compress two leaves into an internal node. Again, at the next level, each pair of siblings is compressed into a node using the hash function, and this is continued until a single node is obtained which is the root of the Merkle tree. In order to verify the membership of a block in a string represented by a Merkle tree root, it is sufficient to provide a path from the leaf node corresponding to the claimed block all the way up to the public root node. This is easily verified given the hash values along the path together with the hash values of sibling nodes.

2.5 Decisional Diffie-Hellman Assumption

Let G be the description of cyclic group of prime order q for $q = \Theta(2^\kappa)$ output by a PPT group generator algorithm $\mathcal{G}$ on input 1^κ. Let g be a generator of G. The *decisional Diffie–Hellman* (DDH) problem for G is the following: given group elements (α, β, γ), distinguish whether they are independent and uniformly random in G or whether $\alpha = g^a$ and $\beta = g^b$ are independent and uniformly random and $\gamma = g^{ab}$.

The DDH assumption is said to hold in G if there exists a function negl such that no PPT algorithm $\mathcal{A}$ can win in the above distinguishing game with probability more that $1/2 + \mathsf{negl}(\kappa)$.

3 Model

In this section, we define certain functionalities that are used in our later protocols.

Stake Distribution. We assume that parties have access to a list of (static) stakeholder accounts. Each such account consists of the committed stake or voting-power and a signature verification key. Each stakeholder additionally has access to his own stake value, the signing key, and the randomness used for the stake commitment. In our protocols the functionality $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com},\mathsf{SIG}}$ is used to provide the static stake information. In practice, this information could for instance be stored in the genesis block of a blockchain.

More generally, the stake distribution is dynamic and can be read from the the blockchain. We discuss the extension to dynamic stake in Section 4.4.

Functionality $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com},\mathsf{SIG}}$

The functionality is parametrized by a commitment scheme Com and a signature scheme $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$.

Initialization

The functionality initially contains a list of stakeholder id's pid and their relative stake α_{pid}. For each stakeholder pid, the functionality does:

1. Compute commitment $\mathsf{Com}(\alpha_{\mathsf{pid}}; r_{\mathsf{pid}})$ with fresh randomness r_{pid};

2. Pick a random secret key $\mathsf{sk}_{\mathsf{pid}}$ and compute $\mathsf{vk}_{\mathsf{pid}} = \mathsf{KeyGen}(\mathsf{sk}_{\mathsf{pid}})$.

Information

- Upon receiving input $(\textsc{GetPrivateData}, \mathrm{sid})$ from a stakeholder pid (or the adversary in the name of corrupted stakeholder) output $(\textsc{GetPrivateData}, \mathrm{sid}, \alpha_{\mathsf{pid}}, r_{\mathsf{pid}}, \mathsf{sk}_{\mathsf{pid}})$.
- Upon receiving $(\textsc{GetList}, \mathrm{sid})$ from a party (or the adversary in the name of corrupted party) output the list $\mathcal{L} = \{(\mathsf{Com}(\alpha_{\mathsf{pid}}), \mathsf{vk}_{\mathsf{pid}})_{\mathsf{pid}}\}$.

Common reference string. In our protocols stakeholders use zero-knowledge proofs to show that they won the stake lottery. The functionality $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$ provides the common reference string required for those zero-knowledge proofs.

Functionality $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$

The functionality is parametrized by a distribution $\mathcal{D}$.

- Sample a CRS, $\mathsf{crs} \leftarrow \mathcal{D}$
- Upon receiving $(\mathsf{Setup}, \mathrm{sid})$ from a party, output $(\mathsf{Setup}, \mathrm{sid}, \mathsf{crs})$.

Verifiable pseudorandom function. In our protocols, stakeholders use the VRF functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ to get the randomness in the stake lottery. The functionality allows a stakeholder to generate a key and then evaluate the VRF under that key. The evaluation returns a value and a commitment of that value. The commitment can then used by parties to verify the claimed $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ evaluation. The functionality also offers Verify queries, where anyone can check if a given output of the VRF was computed correctly. Note that the Verify queries do not disclose the identity of the party who have generated the output. In other words, Verify checks if a given output is in the combined image of all the registered VRF keys.

The VRF functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ is defined as follows.

Functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$

The functionality maintains a table $T(\cdot, \cdot)$ which is initially empty.

Key Generation

Upon input $(\textsc{KeyGen}, \mathrm{sid})$ from a stakeholder pid generate a unique "ideal" key vid, record $(\mathsf{pid}, \mathsf{vid})$. Return $(\textsc{KeyGen}, \mathrm{sid}, \mathsf{vid})$ to pid.

VRF Evaluation

Upon receiving a message $(\textsc{Eval}, \mathrm{sid}, \mathsf{vid}, m)$ from stakeholder pid, verify that pair $(\mathsf{pid}, \mathsf{vid})$ has been recorded. If not, ignore the request.

1. If $T(\mathsf{vid}, m)$ is undefined, pick random values y, r from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(\mathsf{vid}, m) = (y, \mathsf{Com}(y; r), r)$.
2. Return $(\textsc{Evaluated}, \mathrm{sid}, T(\mathsf{vid}, m))$ to pid.

VRF Verification

Upon receiving a message $(\textsc{Verify}, \mathrm{sid}, m, c)$ from some party, do:

1. If there exists a vid such that $T(\mathsf{vid}, m) = (y, c, r)$ for some y, r then set $f = 1$.
2. Else, set $f = 0$.

Output $(\textsc{Verified}, \mathrm{sid}, m, c, f)$ to the party.

Anonymous Broadcast. Stakeholders cannot publish their messages over a regular network as this would reveal their identity. We therefore assume that stakeholders use an anonymous broadcast channel. The functionality $\mathcal{F}_{\mathsf{ABC}}$ allows a party to send messages anonymously to all parties. The adversary is allowed to send anonymous messages to specific parties.

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

Any party can register (or deregister). For each registered party the functionality maintains a message buffer.

Send Message

Upon receiving $(\textsc{Send}, \text{sid}, m)$ from registered party P add m to the message buffers of all registered parties. Output $(\textsc{Sent}, \text{sid}, m)$ to the adversary.

Receive Message

Upon receiving $(\textsc{Receive}, \text{sid})$ from registered party P remove all message from P's message buffer and output them to P.

Adversarial Influence

Upon receiving $(\textsc{Send}, \text{sid}, m, P')$ from $\mathcal{A}$ on behalf some *corrupted* registered party add m to the message buffer of registered party P'. Output $(\textsc{Sent}, \text{sid}, m, P')$ to the adversary.

4 Feasibility of Private Proof-of-Stake

In order to make a proof-of-stake protocol private, a first solution that comes to mind is to have the parties prove in zero-knowledge that they indeed won the lottery (either for a slot or committee membership). This does hide the identity, but it reveals the stake of the winning account. It might seem like one can hide the stake too by having the parties commit to their stakes and give a zero-knowledge proof of winning on committed stake. While this indeed hides the stake in a single proof, it leaks how often a given account wins. One can infer information about the stake in a given account from the frequency with which an account participates in a committee or wins a slot. Therefore, the actual statement that one needs to prove in a private lottery needs to take the list of all accounts as input. Now, a party proves knowledge of corresponding secret key of some public key in a list, and the stake in that account won the lottery. We employ this idea to give a general framework for constructing a private proof-of-stake protocol. The framework applies to proof-of-stake protocols that work with lottery functions which are locally verifiable, that is, a party can locally determine whether it wins or not. The lottery is a function of the party's stake and may depend on other parameters like slot, role etc that we call *entry* parameters. The set $\mathcal{E}$ of entry parameters for the lottery depends on the type of proof of stake. In a slot-based proof-of-stake, for instance, the lottery elects a leader for a particular slot that allows the leader to publish a block for that slot. Ouroboros Praos [KRDO17] is an example of such a slot-based proof-of-stake. In protocols such as Algorand [GHM$^+$17], where the protocol is committee-based, the lottery is for determining a certain role in a committee, and our framework applies to both type of protocols.

4.1 Private Lottery Functionality

The private lottery functionality is an abstraction that we introduce to capture the privacy requirements discussed above. The functionality $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ is parametrized by the set $\mathcal{E}$ of allowed entry parameters, and a predicate function LE. The predicate LE takes as input the relative stake and randomness. It allows stakeholder pid to locally check whether they won the lottery for entry e. If yes, they can publish pairs of the form (e, m) where m is an allowed message as determined and verified by the proof-of-stake protocol that uses the lottery; for instance, when slot-based, m is a block, when committee-based, m is a committee message.

Functionality $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$

The functionality is parametrized by a set $\mathcal{E}$ of entries, and a predicate function LE which takes as input the relative stake (represented by a bit string in $\{0,1\}^{\ell_\alpha}$) and randomness (in $\{0,1\}^{\ell_{\mathsf{VRF}}}$).

Registered Parties. The functionality maintains a list $\mathcal{P}$ of registered parties. For each registered party the functionality maintains a message buffer.

Stakeholders. The functionality maintains a list of (the initial) stakeholders $\mathsf{pid}_1, \ldots, \mathsf{pid}_k$ and their relative stakes $\mathsf{pid}_1.\alpha, \ldots, \mathsf{pid}_k.\alpha$. Finally, the functionality also manages a table $T(,)$ where the entry $T(\mathsf{pid}, \mathsf{e})$ defines whether pid is allowed to publish a message relative to entry e. Initially, the table T is empty.

- Upon receiving $(\text{LOTTERY}, \text{sid}, \mathsf{e})$ from a stakeholder pid (or the adversary in the name of corrupted participant pid) do the following:
  1. If $T(\mathsf{pid}, \mathsf{e})$ is undefined sample a random value $r \in \{0,1\}^{\ell}$ and set $T(\mathsf{pid}, \mathsf{e}) = \mathsf{LE}(\mathsf{pid}.\alpha; r)$.
  2. Output $(\text{LOTTERY}, \text{sid}, \mathsf{e}, T(\mathsf{pid}, \mathsf{e}))$ to pid (or the adversary).
- Upon receiving $(\text{SEND}, \text{sid}, \mathsf{e}, m)$ from a stakeholder pid (or the adversary in the name of corrupted stakeholder pid) do the following:
  1. If $T(\mathsf{pid}, \mathsf{e}) = 1$ add (e, m) to the message buffers of all registered parties and output $(\text{SEND}, \text{sid}, \mathsf{e}, m)$ to the adversary.
- Upon receiving $(\text{SEND}, \text{sid}, \mathsf{e}, m, P')$ from the adversary in the name of corrupted stakeholder pid) do the following:
  1. If $T(\mathsf{pid}, \mathsf{e}) = 1$ add (e, m) to the message buffer of P' and output $(\text{SEND}, \text{sid}, \mathsf{e}, m, P')$ to the adversary.
- Upon receiving $(\text{FETCH-NEW}, \text{sid})$ from a party P (or the adversary on behalf of P) do the following:
  1. Output the content of P's message buffer and empty it.

Get Information

- Upon receiving $(\text{GET-STAKE}, \text{sid})$ from a stakeholder pid (or the adversary on behalf of pid) return $(\text{GET-STAKE}, \text{sid}, \mathsf{pid}.\alpha)$.

4.2 Private Lottery Protocol

The high level idea to implement $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ is as follows. Parties collect information available on the blockchain about the public keys and the corresponding stake of stakeholders. A list $\mathcal{L} = \{(C_{\mathsf{stk}_1}, \mathsf{vk}_1), \cdots, (C_{\mathsf{stk}_n}, \mathsf{vk}_n)\}$ is compiled with tuples of the form $(C_{\mathsf{stk}}, \mathsf{vk})$ where vk is a verification key for a signature scheme (KeyGen, Sign, Ver), and C_{stk} is a commitment to the stake.

The lottery is defined relative to a lottery predicate LE. A stakeholder pid wins the lottery for entry e, if $\mathsf{LE}(\mathsf{stk}, r(\mathsf{e}, \mathsf{pid})) = 1$, where r is randomness that depends on the entry e and stakeholder identity, ensuring that the lottery for different stakeholders is independent. The randomness for the lottery is generated by the VRF functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$. Winning the lottery for e allows a stakeholder to publish messages for e.

To ensure privacy, the stakeholder proves in zero-knowledge that he indeed won the lottery, and as part of this proof it is necessary to prove ownership of his stake. We can do this by proving that the tuple containing the same committed stake and a signing key is in the public list $\mathcal{L}$ (without revealing which one it is), and ownership of the key by proving knowledge of the corresponding secret key. The statement to prove is of the form "I know $\mathsf{sk}, \mathsf{vk}, \mathsf{stk}$ such that $(\mathsf{vk}, \mathsf{sk})$ is a valid signature key pair, $(C_{\mathsf{stk}}, \mathsf{vk}) \in \mathcal{L}$, and I won the lottery with stake stk for entry e". In addition, there needs to be a signature σ on (e, m) to ensure that no other message can be published with this proof, and this signature is also verified inside the zero-knowledge proof with respect to the verification key in the same tuple. Note that since the proof is used to verify the correctness of the signature, the proof itself (and public values for the statement) are not included in the information that is signed. More formally, the proof is of the following form.

$$\begin{aligned}\mathsf{PK}\{(C_{\mathsf{stk}}, \mathsf{stk}, \sigma, \mathsf{vk}, \mathsf{sk}, r) : \mathsf{LE}(\mathsf{stk}; r) = 1 \wedge C_{\mathsf{stk}} = \mathsf{Com}(\mathsf{stk})\\ \wedge \mathsf{Ver}_{\mathsf{vk}}((\mathsf{e}, m), \sigma) = 1 \wedge \mathsf{vk} = \mathsf{KeyGen}(\mathsf{sk}) \wedge (C_{\mathsf{stk}}, \mathsf{vk}) \in \mathcal{L}\}\end{aligned}$$

The published information now consists of entry e, the message m, zero-knowledge proof for the above statement, and certain public values that form the statement. We assume that the zero-knowledge proof requires a CRS which is given by the functionality $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$. The actual publication of the message is done via anonymous broadcast $\mathcal{F}_{\mathsf{ABC}}$ to protect the identity of the stakeholder.

The detailed construction of the private lottery $\mathsf{Lottery\ Protocol}^{\mathcal{E},\mathsf{LE}}$ is given below. The protocol $\mathsf{Lottery\ Protocol}^{\mathcal{E},\mathsf{LE}}$ is run by parties interacting with ideal functionalities $\mathcal{F}_{\mathsf{ABC}}, \mathcal{F}_{\mathsf{Init}}^{\mathsf{Com},\mathsf{SIG}}, \mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}, \mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ and among themselves. Let the algorithms (Setup, Prove, Verify) be a non-interactive zero-knowledge argument system. $\mathsf{Lottery\ Protocol}^{\mathcal{E},\mathsf{LE}}$ proceeds as follows.

Protocol $\mathsf{Lottery\ Protocol}^{\mathcal{E},\mathsf{LE}}$

This describes the protocol from the viewpoint of a party P. If the party is a stakeholder, it additionally has stakeholder-id pid.

Initialization

- Send $(\textsc{GetList}, \mathrm{sid})$ to $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com},\mathsf{SIG}}$ to get the list $\mathcal{L}$ of stakeholders with committed stake and verification key.
- Send $(\mathsf{Setup}, \mathrm{sid})$ to $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$ and get the crs.
- If you are a stakeholder, send $(\textsc{GetPrivateData}, \mathrm{sid})$ to functionality $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com},\mathsf{SIG}}$ and get $\alpha_{\mathsf{pid}}, r_{\alpha,\mathsf{pid}}, \mathsf{sk}_{\mathsf{pid}}$ send $(\textsc{KeyGen}, \mathrm{sid})$ to functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ and get vid; and initialize an empty table $V(\cdot)$.

Lottery and Publishing

- As a stakeholder upon receiving $(\textsc{lottery}, \mathrm{sid}, \mathsf{e})$ from the environment do the following.
  1. If e is not in $\mathcal{E}$ ignore the request.
  2. If $V(\mathsf{e})$ is undefined:
     (a) Send $(\textsc{Eval}, \mathrm{sid}, \mathsf{vid}, \mathsf{e})$ to functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ and receive response $(\textsc{Evaluated}, \mathrm{sid}, (y, c, r))$.
     (b) Compute $b = \mathsf{LE}(\alpha_{\mathsf{pid}}, y)$, and set $V(\mathsf{e}) = (b, y, c, r)$.
  3. Return $(\textsc{lottery}, \mathrm{sid}, \mathsf{e}, b)$ where $V(\mathsf{e}) = (b, y, c, r)$.
- As a stakeholder upon receiving $(\textsc{send}, \mathrm{sid}, \mathsf{e}, m)$ from the environment do the following:
  1. Ignore the request if $V(\mathsf{e}) = (0, \cdots)$ or is undefined.
  2. Let $V(\mathsf{e}) = (1, y, c, r)$. Create the tuple $(\mathsf{e}, m, \pi_{\mathsf{zk}})$ in the following way:
     (a) Compute a signature σ on (e, m) under $\mathsf{sk}_{\mathsf{pid}}$.
     (b) π_{zk} is a non-interactive zero-knowledge proof of knowledge obtained by running Prove using crs for the following statement.

$$\begin{aligned}\mathsf{PK}\{(\alpha_{\mathsf{pid}}, r_{\alpha,\mathsf{pid}}, \mathsf{vk}_{\mathsf{pid}}, \mathsf{sk}_{\mathsf{pid}}, c_{\alpha,\mathsf{pid}}, \sigma, y, r) : \mathsf{Ver}_{\mathsf{vk}_{\mathsf{pid}}}((\mathsf{e}, m), \sigma) = 1\\ \wedge \mathsf{LE}(\alpha_{\mathsf{pid}}, y) = 1 \wedge \mathsf{vk}_{\mathsf{pid}} = \mathsf{KeyGen}(\mathsf{sk}_{\mathsf{pid}}) \wedge c = \mathsf{Com}(y; r)\\ \wedge c_{\alpha,\mathsf{pid}} = \mathsf{Com}(\alpha_{\mathsf{pid}}; r_{\alpha,\mathsf{pid}}) \wedge (c_{\alpha,\mathsf{pid}}, \mathsf{vk}_{\mathsf{pid}}) \in \mathcal{L}\}\end{aligned}$$

  3. Send $(\textsc{send}, \mathrm{sid}, (\mathsf{e}, m, c, \pi_{\mathsf{zk}}))$ to $\mathcal{F}_{\mathsf{ABC}}$.
- Upon receiving $(\textsc{fetch-new}, \mathrm{sid})$ from the environment do the following:
  1. Send $(\textsc{Receive}, \mathrm{sid})$ to $\mathcal{F}_{\mathsf{ABC}}$ and receive as message vector $\vec{m}$.
  2. For each $(\mathsf{e}, m, c, \pi_{\mathsf{zk}}) \in \vec{m}$ do:

(a) Check that $\mathsf{e} \in \mathcal{E}$.

(b) Send $(\textsc{Verify}, \mathrm{sid}, \mathsf{e}, c)$ to functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$. For response $(\textsc{Verified}, \mathrm{sid}, \mathsf{e}, c, b)$ from $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$, verify that $b = 1$.

(c) Verify the correctness of zero-knowledge proof π_{zk}, i.e., check that $\mathsf{Verify}(\mathsf{crs}, \pi_{\mathsf{zk}}) = 1$.

(d) If all check pass add $(\mathsf{e}, m, c, \pi_{\mathsf{zk}})$ to $\vec{o}$.

3. Output $(\textsc{fetch-new}, \mathrm{sid}, \vec{o})$.

Get Information

- As a stakeholder upon receiving $(\textsc{get-stake}, \mathrm{sid})$ from the environment output $(\textsc{get-stake}, \mathrm{sid}, \alpha_{\mathsf{pid}})$ where α is your lottery power.

Theorem 1. *The protocol* $\mathsf{Lottery\ Protocol}^{\mathcal{E},\mathsf{LE}}$ *realizes the* $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ *functionality in the* $(\mathcal{F}_{\mathsf{ABC}}, \mathcal{F}_{\mathsf{Init}}^{\mathsf{Com,SIG}}, \mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}, \mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}})$*-hybrid world in the presence of a PPT adversary.*

Proof. Let $\mathcal{S}_{\mathsf{zk}} = (\mathcal{S}_1, \mathcal{S}_2)$ be the simulator of the zero-knowledge proof system used in the protocol $\mathsf{Lottery\text{-}Protocol}^{\mathcal{E},\mathsf{LE}}$.

We construct a simulator $\mathcal{S}_{\mathsf{lottery}}$ and argue that the views of the adversary in the simulated execution and real protocol execution are computationally close. Here, we provide a high-level idea of simulation. The full description of the simulator $\mathcal{S}_{\mathsf{lottery}}$ can be found in Appendix A. At the beginning, the simulator gets the stake of dishonest stakeholders from $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ and internally emulates the $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com,SIG}}$. The simulator generates a CRS (with trapdoor) and emulates $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$. Similarly, the VRF functionality $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ is emulated by the simulator. If a dishonest stakeholder ask the VRF for an entry $\mathsf{e} \in \mathcal{E}$, the simulator first asks $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ if the stakeholder wins for this entry and then samples the output accordingly. The simulation of $\mathcal{F}_{\mathsf{ABC}}$ consists of two parts. First, if the adversary wants to send a message the simulator checks if the message is a valid tuple of the form $m' = (\mathsf{e}, m, c, \pi_{\mathsf{zk}})$. If this is the case, the simulator submits (e, m) to $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ for publication. Second, if an honest stakeholder publishes a tuple (e, m) the simulator creates a tuple $(\mathsf{e}, m, c, \pi_{\mathsf{zk}})$ which contains a simulated proof π_{zk} and adds it to the message buffers of dishonest parties. □

4.3 Flavors of Proof-of-Stake

As seen above any proof-of-stake lottery can be made private. In the following we discuss how this process applies to two widely-used types of proof-of-stake lotteries.

Slot-based PoS. In slot-based PoS protocols (e.g., the Ouroboros Praos protocol [KRDO17]), time is divided into slots and blocks are created relative to a slot. Parties with stake can participate in a slot lottery, and winning the lottery allows a stakeholder to create a block in a particular slot. Here, the set of lottery entries are slots, i.e. $\mathcal{E} = \mathbb{N}^+$. An (honest) lottery winner will publish one message in the form of a new block via $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$.

Committee-based PoS. In committee-based PoS protocols, such as Algorand [GHM+17], a stakeholder wins the right to take part in a committee which for example determines the next block. In such a protocol, the set of lottery entries could be of the form $(\mathsf{cid}, \mathsf{role})$ where cid is the id of the committee and role is the designated role of the winner. An (honest) lottery winner will then publish his messages for the committee protocol via $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$.

4.4 Dynamic Stake

Our protocol in Section 4.2 assumes that the stake distribution is fixed at the onset of the computation (in the form of $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com,SIG}}$), which is the *static stake* setting. In the following we give an intuition on how the protocol can be made to support the *dynamic stake* setting where the set of stakeholders and the distribution evolve over time.

Protocol idea. The idea is to collect information about the public keys and the corresponding stake of stakeholders on the blockchain instead of using $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com},\mathsf{SIG}}$. We assume that for each entry e the (honest) parties agree on the corresponding stake distribution $\mathcal{L}_\mathsf{e}$. This stake distribution might not be known from the beginning of the protocol[3]. We assume that (if defined) $\mathcal{L}_\mathsf{e}$ can be computed efficiently from the blockchain. The parties then use $\mathcal{L}_\mathsf{e}$ in the lottery protocol when dealing with e.

Observe that computation of $\mathcal{L}_\mathsf{e}$ is completely separated from the actual lottery protocol. The protocol therefore remains secure even in the dynamic stake setting.

4.5 Rewards

In many proof-of-stake based cryptocurrencies a stakeholder will include some sort of identification (e.g. his verification key) in his messages (e.g. in a new block) so that the rewards such as transaction fees are appropriately paid out. This, of course leaks the identity of the lottery winner and thus also information about his stake. This leakage can be prevented if the cryptocurrency allows for anonymous transactions and anonymous account creation. For instance, one could think of ZCash [BCG+14], which though not based on proof-of-stake allows for such mechanisms. Each stakeholder maintains a list of fresh accounts. Whenever the stakeholder needs to provide information for rewards, the stakeholder uses one of the accounts as the reward destination. Since the account was created anonymously it cannot be linked to the stakeholder. Later on, the stakeholder can anonymously transfer the money from that account to any of its other accounts.

5 Making Ouroboros Praos Private

In this section, we look at the Ouroboros Praos proof-of-stake protocol from [DGKR18], and apply the technique from our private lottery framework. In particular, we describe how the zero-knowledge proofs necessary for π_{zk} are instantiated for the Ouroboros Praos lottery.

5.1 Ouroboros Praos Leader Election

Recall that the VRF leader election scheme in Ouroboros Praos works as follows. The probability p that a stakeholder pid is elected as leader in a slot sl is independent of other stakeholders. It depends only on pid's relative stake $\alpha = \mathsf{stk}/\mathsf{Stake}$ where Stake is the total stake in the system. More precisely, the probability p is given by,

$$p = \phi_f(\alpha) \triangleq 1 - (1-f)^\alpha$$

where f is the difficulty parameter. A stakeholder pid can evaluate the VRF using private key k along with a proof of evaluation that can be verified using a public key. To check if they are a leader in slot sl, the stakeholder computes their threshold $T = 2^{\ell_\alpha} p$ where ℓ_α is the output length of the VRF. The stakeholder wins if $y < T$ where $(y, \pi) = \mathsf{VRF}(\mathsf{k}, \mathsf{sl})$. The proof π allows any party to verify pid's claim given pid's verification key. In other words, the LE predicate function for Ouroboros Praos is given by:

$$\mathsf{LE}(\mathsf{stk}; y) = \begin{cases} 1, & \text{if } y < 2^{\ell_\alpha} \cdot \left(1 - (1-f)^{\frac{\mathsf{stk}}{\mathsf{Stake}}}\right) \\ 0, & \text{otherwise} \end{cases} .$$

Ouroboros Praos VRF. Ouroboros Praos uses the 2-Hash VRF of [JKK14] based on the hardness of the computational Diffie-Hellman problem. Let $\mathbb{G} = \langle g \rangle$ be a group of order q. VRF uses two hash functions H_1 and H_2 modeled as random oracles. H_1 has range $\{0,1\}^{\ell_\alpha}$ and H_2 has range $\mathbb{G}$. Given a key $\mathsf{k} \in \mathbb{Z}_q$, the public key is $v = g^\mathsf{k}$, and $(y, \pi) = \mathsf{VRF}(m)$ is given by $y = H_1(m, u)$ where $u = H_2(m)^\mathsf{k}$, and $\pi : \mathsf{PK}\{(k) : \log_{H_2(m)}(u) = \log_g(v)\}$.

[3] If, for example, the entries are slots (cf. Section 4.3) the stake distribution for a particular entry is only defined once the blockchain has grown far enough.

5.2 Anonymous Verifiable Random Function

We define a primitive that we call an *anonymous VRF* that captures a requirement necessary in the proof π_{zk}; which is roughly that verification should not reveal the public key. The high level idea is that there are many public keys associated with a secret key, and two different evaluations (on different messages) under the same secret key cannot be linked to a public key. The verifiability property is still preserved, that is, there is a public key, which allows to verify the correctness of output with respect to a proof. We now give a formal definition.

Definition 2. *A function family $F_{(\cdot)}(\cdot) : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ is a family of anonymous VRFs, if there is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Update}, \mathsf{VRFprove}, \mathsf{VRFverify})$ such that: $\mathsf{Gen}(1^k)$ generates a key pair $(\mathsf{pk}, \mathsf{k})$; Update takes the public key pk and outputs an updated public key pk'; $\mathsf{VRFprove}_{\mathsf{k}}(\mathsf{pk}', x)$ outputs a tuple $(F_{\mathsf{k}}(x), \pi_{\mathsf{k}}(x))$ where $\pi_{\mathsf{k}}(x)$ is the proof of correct evaluation; $\mathsf{VRFverify}_{\mathsf{pk}'}(x, y, \pi)$ verifies that $y = F_{\mathsf{k}}(x)$ using the proof π. We require that the following properties are satisfied.*

- *Pseudorandomness. For any pair of PPT (A_1, A_2), the following probability is $1/2 + \mathsf{negl}(k)$.*

$$\Pr\left(\begin{array}{c} b = b' \\ \wedge x \notin Q_1 \cup Q_2 \end{array}\middle|\begin{array}{c} (\mathsf{pk}, \mathsf{k}) \leftarrow \mathsf{Gen}(1^k); (Q_1, x, \mathsf{state}) \leftarrow A_1^{\mathsf{VRFprove}(\cdot)}(\mathsf{pk}); \\ y_0 = F_{\mathsf{k}}(x); y_1 \leftarrow \{0,1\}^{\ell}; \\ b \leftarrow \{0,1\}; (Q_2, b') \leftarrow A_2^{\mathsf{VRFprove}(\cdot)}(y_b, \mathsf{state}) \end{array}\right).$$

  The sets Q_1, Q_2 contain all the queries made to the Prove oracle. The random variable state stores information that A_1 can save and pass on to A_2.
- *Uniqueness[4]. No PPT adversary can output values $(\mathsf{pk}, x, y_1, y_2, \pi_1, \pi_2)$ such that $y_1 \neq y_2$ and*

$$\mathsf{VRFverify}_{\mathsf{pk}}(x, y_1, \pi_1) = \mathsf{VRFverify}_{\mathsf{pk}}(x, y_2, \pi_2) = 1$$

- *Provability. $\mathsf{VRFverify}_{\mathsf{pk}'}(x, y, \pi) = 1$ for $(y, \pi) = \mathsf{VRFprove}_{\mathsf{k}}(\mathsf{pk}', x), \mathsf{pk}' \leftarrow \mathsf{Update}(\mathsf{pk})$*
- *Anonymity. For any PPT algorithm A, the following probability is $1/2 + \mathsf{negl}(k)$.*

$$\Pr\left(b = b' \middle| \begin{array}{c} (\mathsf{pk}_0, \mathsf{k}_0) \leftarrow \mathsf{Gen}(1^k); (\mathsf{pk}_1, \mathsf{k}_1) \leftarrow \mathsf{Gen}(1^k); \\ x \leftarrow A(\mathsf{pk}_0, \mathsf{pk}_1); \mathsf{pk}_0' \leftarrow \mathsf{Update}(\mathsf{pk}_0); \\ (y_0, \pi_0) = \mathsf{VRFprove}_{\mathsf{k}_0}(\mathsf{pk}_0', x); \mathsf{pk}_1' \leftarrow \mathsf{Update}(\mathsf{pk}_1); \\ (y_1, \pi_1) = \mathsf{VRFprove}_{\mathsf{k}_1}(\mathsf{pk}_1', x); b \leftarrow \{0,1\}; b' \leftarrow A(\mathsf{pk}_b', y_b, \pi_b) \end{array}\right).$$

Intuitively, the above definition says that no adversary can tell which key an output came from, given two public keys.

Anonymous VRF construction. We show how to instantiate the AVRF primitive. Our starting point is a PRF adapted from the 2-hash VRF construction used in Ouroboros Praos. To use this primitive in making Ouroboros Praos private, the additional property required is that it is secure against malicious key generation. The work of [KKKZ18] shows that this PRF indeed satisfies unpredictability under malicious key generation. Let AVRF be the tuple of algorithms $(\mathsf{Gen}, \mathsf{VRFprove}, \mathsf{VRFverify})$ which are defined as follows.

- $\mathsf{Gen}(1^k)$: Choose a generator g of a group of prime order q such that $q = \Theta(2^k)$, and sample a random $\mathsf{k} \in \mathbb{Z}_q$ and output $(\mathsf{pk}, \mathsf{k})$, where $\mathsf{pk} = (g, g^{\mathsf{k}})$.
- $\mathsf{Update}(\mathsf{pk})$: Let pk be (g, v). Choose a random $r \in \mathbb{Z}_q$, let $g' = g^r, v' = v^r$, set $\mathsf{pk}' = (g', v')$, output pk'.
- $\mathsf{VRFprove}_{\mathsf{k}}(\mathsf{pk}', x)$: Let pk' be (g, v). Compute $u = H(x), y = u^{\mathsf{k}}$, and $\pi' : \mathsf{PK}\{(\mathsf{k}) : \log_u(y) = \log_g(v)\}$. Set $\pi = (u, \pi')$. Output (pk', y, π)
- $\mathsf{VRFverify}_{\mathsf{pk}'}(x, y, \pi)$: Output 1 if $u = H(x)$ and π verifies, and 0 otherwise.

[4] Note that this is a weakening of the uniqueness property required from VRF in literature, where, uniqueness says there does not exist values $(\mathsf{pk}, x, y_1, y_2, \pi_1, \pi_2)$ such that $y_1 \neq y_2$ and $\mathsf{VRFverify}_{\mathsf{pk}}(x, y_1, \pi_1) = \mathsf{VRFverify}_{\mathsf{pk}}(x, y_2, \pi_2) = 1$.

It is clear that the above construction satisfies the above properties of a AVRF. For anonymity, we reduce to DDH; we show that any adversary who breaks anonymity can be used to break DDH. Let $\mathcal{A}$ be the adversary who wins the anonymity game in Definition 2. We now show how to use $\mathcal{A}$ to break DDH. Let $\mathcal{B}$ be an adversary who receives a challenge (g, g^a, g^b, g^c) and has to determine whether it is a DDH tuple or not. $\mathcal{B}$ works as follows: it chooses random $k_0, k_1 \in \mathbb{Z}_q$, sets $\mathsf{k}_0 = k_0, \mathsf{pk}_0 = (g^a, g^{k_0 a}), \mathsf{k}_1 = k_1, \mathsf{pk}_1 = (g^b, g^{k_1 b})$ and return $\mathsf{pk}_0, \mathsf{pk}_1$ to $\mathcal{A}$. On receiving a x, $\mathcal{B}$ chooses $\beta \in \{0,1\}$ at random and returns $(\mathsf{pk}'_\beta, y_\beta, \pi_\beta)$ to $\mathcal{A}$, where $\mathsf{pk}'_\beta = (g^c, g^{k_\beta c})$ and y_β, π_β are evaluated with key k_β. Let β' be the output of $\mathcal{A}$. If $\beta = \beta'$, $\mathcal{B}$ returns DDH tuple, otherwise $\mathcal{B}$ decides not a DDH tuple.

Note that while AVRF gives the anonymous verifiability property, it does not guarantee that the key used to evaluate comes from one of the two keys that the adversary sees at the onset of the game. In applications, it is desirable to satisfy this "key membership" property. Indeed, the $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ functionality that was defined in Section 3 has the property that verification does not leak a public key and also guarantees that it is one of the registered keys. The $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ functionality also allows verifiability of y while keeping y secret. We use other techniques on top of the AVRF primitive to realize the $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$ functionality; in general, proving membership of the corresponding AVRF secret key in a list of committed secret keys will suffice for membership and we preserve privacy by committing to the output and proving correct evaluation in zero-knowledge. We elaborate on this in the next section.

"Approval Voting" via AVRF. To demonstrate the usefulness of AVRF outside of the context of PPoS, here is a simple example application, namely approval voting. In approval voting, a group of users can vote (e.g., approve) any number of candidates, and the winner of the election is the candidate who is approved by the highest number of voters. To implement such voting with cryptographic techniques, one needs to ensure anonymity of the voters and, at the same time, that each voter can approve each candidate at most once. This can be easily done using our AVRF abstraction: Each user registers an AVRF public key $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$. To vote on option x, user i publishes $\mathsf{pk}' = \mathsf{Update}(\mathsf{pk}_i)$ and gives a ZK-proof that $\exists i : (\mathsf{pk}', \mathsf{pk}_i) \in L$ (in our AVRF the language L is simply the language of DDH tuple). Then the user computes and publishes $(y, \pi) = \mathsf{VRFprove}_{\mathsf{k}}(\mathsf{pk}', x)$. If the proof π does not verify or if the value y has already appeared in this poll, then the other users discard this vote. Otherwise, they register a new vote for option x. Now, due to the anonymity and indistinguishability properties of the VRF, it is unfeasible to link any two casted votes, except if the same user tries to approve the same candidate more than once, since the value y is only a function of k and x.

5.3 Private Ouroboros Praos

Recall that our private lottery protocol now needs to prove that $\mathsf{LE}(\mathsf{stk}; y) = 1$ in zero-knowledge. For this, we need to prove $y < T$ in zero knowledge, that is, without revealing y or T[5]. Note that, in addition, we need to prove the correct computation of T which involves evaluating ϕ on a secret α involving floating-point arithmetic. Using generic zero-knowledge proofs for a statement like above would be expensive. We show how to avoid this and exploit the specific properties of the statement. In particular, we take advantage of the "independent aggregation" property that is satisfied by the above function ϕ to construct a zero-knowledge proof for leader election i.e., that the function ϕ satisfies the following property:

$$1 - \phi\left(\sum_i \alpha_i\right) = \prod_i (1 - \phi(\alpha_i))$$

The above implies that if a party were to split its stake among virtual parties, the probability that the party is elected for a particular slot is identical to the probability that one of the virtual parties is elected for that slot.

Remark 1. Due to rounding performed when evaluating the predicate LE, the probability of winning is not identical under redistribution of stakes. However, by setting the precision ℓ_α appropriately we can always ensure that the difference between the winning probabilities above is at most negligible.

[5] Since T is a direct function of stk, it should be clear why T should stay private. At the same time, revealing the value y and the fact that LE output 1 allows to rule out that $\mathsf{stk} = s$ for any value s such that $\mathsf{LE}(s; y) = 0$.

Proof of correct evaluation of LE predicate π_{LE}. The idea behind our proof is to split the stake among virtual parties and prove that one of the virtual parties wins without revealing which one of them won. We also use the hash-based AVRF instantiation in the LE since we want to achieve verifiability of correct evaluation without disclosing a public key. More precisely, each stakeholder has a key pair $(\mathsf{vk}, \mathsf{sk})$ of a signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$, and a key pair $(\mathsf{pk}, \mathsf{k})$ for an AVRF family F. To realize the key membership property for the AVRF, we now include the public key for the AVRF in a stakeholder's tuple. Thus, the list $\mathcal{L}$ now consists of tuples $(C_{\mathsf{stk}}, \mathsf{vk}, \mathsf{pk})$.

Let pid be a stakeholder with (absolute) stake stk, and wants to prove that it won the election, that is $\mathsf{LE}(\mathsf{stk}; F_{\mathsf{k}}(\mathsf{sl})) = 1$, where F is the AVRF function. Let $b_i, \forall i \in [0, s-1]$ be the bits of the stake of stakeholder pid, where $\mathsf{stk} = \sum_{i=0}^{s-1} 2^i b_i$ and the maximum stake in the system is represented in s-bits. Now, the stake is split among s virtual parties "in the head" where the stake of the ith virtual party is $2^i b_i$. We now have by the aggregation property, that the probability of winning with stake stk is equal to the probability of winning with one of the above s stakes. Let the probability of winning with stake $2^i b_i$ be p_i, let $(y_i, \pi_i) = \mathsf{VRFprove}_{\mathsf{k}}(\mathsf{pk}_i, i||\mathsf{sl})$, for $\mathsf{pk}_i \leftarrow \mathsf{Update}(\mathsf{pk})$. Let T_i be the threshold corresponding to the ith divided stake, $T_i = 2^{\ell_\alpha} p_i$. We use the AVRF key of the stakeholder to evaluate y_i corresponding to the ith stake by including the index i along with the slot number in the evaluation, and prove that $y_i < T_i$ for at least one i. Note that now, the thresholds T_i in the statement are public values, in contrast to private threshold prior to the the stake being split among virtual parties. In addition, the statement only uses the function ϕ in a blackbox way and is independent of the difficulty parameter f. The zero-knowledge proofs, therefore do not have to change with tuning of the difficulty parameter of the leader election function. The proof is for the statement that there exists at least one bit such that the bit is one, the corresponding virtual party won the lottery, bits combine to yield the committed stake and correct evaluation of the AVRF. The following is a proof that LE was evaluated correctly on stk.

$$\mathsf{PK}\{(y_1, \cdots y_s, b_1, \cdots, b_s, i^*, \mathsf{k}, \mathsf{stk}) : \\ \left(\bigwedge_{i=1}^{s} (b_i \in \{0,1\})\right) \wedge (b_{i^*} = 1 \wedge y_{i^*} < T_{i^*} \wedge y_{i^*} = F_{\mathsf{k}}(i^*||\mathsf{sl})) \wedge \mathsf{stk} = \sum 2^j b_j\}$$

Proof π_{LE} is about the correct evaluation of the predicate LE on private stake and randomness, and correct evaluation of the y_i's. The above proof convinces that a committed stake wins the lottery. It is still necessary to prove ownership of this stake. We can do this by proving that the tuple containing the same committed stake, signature verification key, and an APRF key is in the list $\mathcal{L}$, and ownership of the signature key by proving knowledge of the corresponding signing key.

Proof of ownership π_{own}. We represent the list $\mathcal{L}$ as a Merkle tree, where the leaf are the tuples $(C_{\mathsf{stk}_{\mathsf{pid}}}, \mathsf{vk}_{\mathsf{pid}}, \mathsf{pk}_{\mathsf{pid}})_{\mathsf{pid}} \in \mathcal{L}$. We can now prove membership by proving a valid path to the public root given a commitment to a leaf. Let $\mathcal{L}(\mathsf{root})$ denote the Merkle tree representation of the list $\mathcal{L}$. Given the root of a Merkle tree, an AVRF public key, and commitments to the signature verification key, and stake, we want a party proposing a new block to prove that the stake used in the proof of winning lottery corresponds to the signature key and AVRF key it "owns". That is, prove knowledge of $(\mathsf{vk}, \mathsf{stk}, \mathsf{pk})$ such that $C_{\mathsf{stk}}||\mathsf{vk}||\mathsf{pk}$ is a leaf of the Merkle tree with root root. To prove membership, one can reveal the path along with the values of the sibling nodes up to the root. We want to prove membership without disclosing the leaf node and therefore use a zero knowledge proof π_{path} to prove a valid path from a committed leaf to a public root. Let l_i be $C_{\mathsf{stk}}||\mathsf{vk}||\mathsf{pk}$, H be the hash function function used to construct the Merkle tree, and let $\mathsf{sib}_1, \ldots, \mathsf{sib}_t$ be the sibling nodes of the nodes on the path from l_i to the root of a tree with depth t. π_{path} proves $l_i \in \mathcal{L}(\mathsf{root})$.

$$\mathsf{PK}\{(l_i, \mathsf{sib}_1, \ldots, \mathsf{sib}_t) : \mathsf{H}(\cdots \mathsf{H}(\mathsf{H}(l_i||\mathsf{sib}_1)||\mathsf{sib}_2) \cdots) = \mathsf{root}\}$$

Using the above proof π_{path}, we can prove ownership. Given root, we denote by π_{own} the following proof.

$$\mathsf{PK}\{(\mathsf{vk}, \mathsf{stk}, \mathsf{k}, \mathsf{pk}, C_{\mathsf{stk}}) : (C_{\mathsf{stk}}||\mathsf{vk}||\mathsf{pk}) \in \mathcal{L}(\mathsf{root}) \wedge \mathsf{pk} = g^{\mathsf{k}}\}$$

Proof of signature on a block under the winning key π_{sig}. π_{zk} also consists of a proof that a block signature verifies under the winning key. π_{sig} denotes the following proof, where M is the public block information that is signed.

$$\mathsf{PK}\{(\mathsf{vk}, \mathsf{sk}, \sigma) : \mathsf{vk} = \mathsf{KeyGen}(\mathsf{sk}) \wedge \mathsf{Ver}_{\mathsf{vk}}(\sigma, M) = 1\}$$

Overall proof. The detailed construction of proof π_{zk} is given below. If the commitment to stake C_{stk} is an extended Pedersen commitment (e.g., $h^r \cdot \Pi_{i=1}^{s}(g_i)^{b_i}$) where the stakes are already committed bit by bit, the proof π_{LE} is a standard sigma protocol. If instead, it is a Pedersen commitment to the entire stake, one can publish fresh commitments to bits and prove correct recombination. The range proofs that are used in π_{LE} allow one to prove that $x \in [0, R]$ for a public R and committed x. Range proofs may be instantiated using several known techniques [CCs08, Bou00]. More recently, the technique of bulletproof [BBB+18] results in very efficient range proofs when the interval is $[0, 2^n - 1]$ for some n. Since we use SNARKs for other statements, we also implement the range check inside a SNARK resulting in short proofs. The proof π_{LE} also relies on the OR composition of sigma protocols. π_{own} may be realized efficiently using SNARKs when the Merkle tree hash function H is non-algebraic. While it might seems like such a statement would result in inefficient proofs, this can in fact be done efficiently in practice, and is implemented by ZCash's private-pool transactions [BCG+14]. The predicate Eq that tests if two public keys comes from the same key is the following predicate for the concrete AVRF: it outputs 1 if pk and pk' form a DDH tuple. For a public pk' and private pk as in our case, this can be implemented using double discrete logarithm sigma protocol proofs [CS97, MGGR13]. The proof for part of the statement represented as a circuit (the hash function) in the hash-based AVRF can be implemented using SNARKs, and we can use the construction of [AGM18] for SNARK on algebraically committed input and output so we can work with Pedersen commitments and sigma protocols for other parts of the proof. The rest of the proof components may be implemented using standard sigma protocol techniques.

Protocol Constructing π_{zk}

– Given a list $\mathcal{L} = \{(C_{\mathsf{stk}_{\mathsf{pid}}}, \mathsf{vk}_{\mathsf{pid}}, \mathsf{pk}_{\mathsf{pid}})_{\mathsf{pid}}\}$, construct a Merkle tree representation. Let root be the root of the tree.
– For stakeholder pid, let $b_1, \ldots, b_s$ represent the bits of the stake stk. Let the private information be $(\mathsf{stk}, C_{\mathsf{stk}}, C_{\mathsf{k}}, \mathsf{vk}, \mathsf{sk}, \mathsf{k})$. Let M be the part of the block that is signed. Compute signature $\sigma = \mathsf{Sign}(\mathsf{sk}, M)$. To construct a proof π_{zk} for submitting a new block:
  - Compute $\mathsf{pk}' \leftarrow \mathsf{Update}(\mathsf{pk})$ and $(y_i, \pi_i) = \mathsf{VRFprove}_{\mathsf{k}}(\mathsf{pk}', i||\mathsf{sl})$. Then publish pk'. Compute and publish $C_\sigma = \mathsf{Com}(\sigma), C_{\mathsf{vk}} = \mathsf{Com}(\mathsf{vk}), C_{\mathsf{sk}} = \mathsf{Com}(\mathsf{sk})$. There is a predicate $\mathsf{Eq}(\mathsf{pk}_i, \mathsf{pk}, \mathsf{k})$ which outputs 1 if $\mathsf{pk}_i, \mathsf{pk}$ have the same secret key k. Compute proof of correct evaluation of LE predicate π_{LE} :

$$\begin{aligned}&\mathsf{PK}\{(y_1, \cdots y_s, b_1, \cdots, b_s, \pi_j, \mathsf{stk}, C_{\mathsf{stk}}) :\\ &\quad (\forall i\, (b_i \in \{0,1\}))\\ &\quad \wedge \big(\exists j\, \big(b_j = 1 \wedge y_j < T_j \wedge \mathsf{VRFverify}_{\mathsf{pk}'}(j||\mathsf{sl}, y_j, \pi_j) = 1\big)\big)\\ &\quad \wedge \mathsf{stk} = \sum 2^j b_j \wedge C_{\mathsf{stk}} = \mathsf{Com}(\mathsf{stk})\}\end{aligned}$$

  - Compute proof of signature on a block under the winning key π_{sig}:

$$\begin{aligned}\mathsf{PK}\{(\mathsf{vk}, \mathsf{sk}, \sigma) : \mathsf{vk} = \mathsf{KeyGen}(\mathsf{sk}) \wedge C_{\mathsf{vk}} = \mathsf{Com}(\mathsf{vk}) \wedge C_{\mathsf{sk}} = \mathsf{Com}(\mathsf{sk})\\ \wedge C_\sigma = \mathsf{Com}(\sigma) \wedge \mathsf{Ver}_{\mathsf{vk}}(\sigma, M) = 1\}\end{aligned}$$

  - Compute proof of ownership of signature and AVRF key π_{own}:

$$\begin{aligned}\mathsf{PK}\{(\mathsf{vk}, \mathsf{stk}, \mathsf{k}, \mathsf{pk}, C_{\mathsf{stk}}) : (C_{\mathsf{stk}}||\mathsf{vk}||\mathsf{pk}) \in \mathcal{L}(\mathsf{root}) \wedge C_{\mathsf{vk}} = \mathsf{Com}(\mathsf{vk})\\ \wedge \mathsf{Eq}(\mathsf{pk}', \mathsf{pk}, \mathsf{k}) = 1 \wedge C_{\mathsf{stk}} = \mathsf{Com}(\mathsf{stk})\}\end{aligned}$$

Set π_{zk} to be $(\pi_{\mathsf{LE}}, \pi_{\mathsf{sig}}, \pi_{\mathsf{own}})$.

Usage of π_{zk} in Ouroboros Praos. If a stakeholder has won the lottery for slot sl, they will create a new block of the form $(\mathsf{pt}, \mathsf{sl}, \mathsf{st}, c, \pi_{\mathsf{zk}})$ where pt is a reference to a previous block, st the block payload, c

is a commitment to y, the output of the AVRF, and π_{zk} is the proof as described above. The stakeholder then publishes the block using an anonymous broadcast.

We give an estimate of the proof size that determines the overhead that is incurred by privacy preserving Ouroboros Praos compared to the non-private version. The size of π_{LE} is dominated by $O(s)$ group/field elements due to the sigma protocol OR composition, with the rest of the components resulting in succinct SNARK proofs. π_{sig} for the key-evolving signature scheme may be implemented by using SNARK on committed input together with sigma protocols with only a slight overhead in size over the SNARK proof. The size of π_{own} is dominated by the proof size for the predicate Eq which is $O(\kappa)$ elements for a statistical security parameter κ. The size of π_{zk} is therefore roughly (ignoring the size of proofs for statements that use SNARKs and standard sigma protocols), $O(s)+O(\kappa)$ group/field elements where s is the number of bits to represent the stake in the system, κ is the statistical security parameter. We remark that the actual complexity depends on the implementation of the signature scheme, and potentially the hash function of the AVRF.

Corollary 1. *Ouroboros Praos used with the private lottery protocol results in a private proof-of-stake protocol.*

Proof. The proof easily follows from the properties of the underlying building blocks. Note that overall protocol remains the same as in the original Ouroboros Praos, with only small differences: Instead of using a VRF, a stakeholder uses an AVRF to determine whether they win the slot-lottery. Then, a slot leader will publish a block with a zero-knowledge proof of the above form (instead of adding his verification key and a VRF-proof). Due to the soundness of the zero-knowledge protocol and the uniqueness property of the AVRF, the modified protocol still has the same security properties as Ouroboros Praos i.e., the the protocol still reaches consensus under the same security guarantees as the original protocol.

The proof that the resulting protocol is a *private* proof of stake follows directly from the proof of Theorem 1.

Acknowledgements. We would like the thank Alessandra Scafuro, Foteini Baldimtsi, Varun Madathil and Linfeng Zhou for pointing out an issue with instantiating the zero-knowledge proofs about random oracle pre-images in an earlier version of the paper.

References

[AGM18] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 643–673. Springer, Heidelberg, August 2018. `doi:10.1007/978-3-319-96878-0_22`.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. `doi:10.1109/SP.2018.00020`.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. `doi:10.1109/SP.2014.36`.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. `doi:10.1145/62212.62222`.

[BGK+18] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. Cryptology ePrint Archive, Report 2018/378, 2018. `https://eprint.iacr.org/2018/378`.

[BGM16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *International Conference on Financial Cryptography and Data Security*, pages 142–157. Springer, 2016.

[bit11] Proof of stake instead of proof of work. `https://bitcointalk.org/index.php?topic=27787.0`, July 2011.

[BLMR14] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 431–444. Springer, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_31.

[BPS16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. http://eprint.iacr.org/2016/919.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252. Springer, Heidelberg, December 2008. doi:10.1007/978-3-540-89255-7_15.

[CDE$^+$16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994. doi:10.1007/3-540-48658-5_19.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 107–122. Springer, Heidelberg, May 1999. doi:10.1007/3-540-48910-X_8.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Heidelberg, August 1997. doi:10.1007/BFb0052252.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_30.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78375-8_3.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 16–30. Springer, Heidelberg, August 1997. doi:10.1007/BFb0052225.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[GHM$^+$17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. http://eprint.iacr.org/2017/454.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Heidelberg, December 2014. doi:10.1007/978-3-662-45608-8_13.

[KKKZ18] Thomas Kerber, Markulf Kohlweiss, Aggelos Kiayias, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. Cryptology ePrint Archive, Report 2018/1132, 2018. To appear at IEEE Symposium on Security and Privacy - S&P 2019. URL: https://eprint.iacr.org/2018/1132.

[KN12] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_12.

[Lin15] Yehuda Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 93–109. Springer, Heidelberg, March 2015. doi:10.1007/978-3-662-46494-6_5.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press, May 2013. doi:10.1109/SP.2013.34.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[OM14] K. J. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. In *ISSC 2014/CIICT 2014*, pages 280–285, 2014. doi:10.1049/cp.2014.0699.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992. `doi:10.1007/3-540-46766-1_9`.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

Appendix

A Private Proof of Stake Lottery

Theorem 1. *The protocol* $\mathsf{Lottery\ Protocol}^{\mathcal{E},\mathsf{LE}}$ *realizes the* $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ *functionality in the* ($\mathcal{F}_{\mathsf{ABC}}, \mathcal{F}_{\mathsf{Init}}^{\mathsf{Com,SIG}}$, $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}, \mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$)*-hybrid world in the presence of a PPT adversary.*

Proof. Let $\mathcal{S}_{\mathsf{zk}} = (\mathcal{S}_1, \mathcal{S}_2)$ be the simulator of the zero-knowledge proof system used in $\mathsf{Lottery\text{-}Protocol}^{\mathcal{E},\mathsf{LE}}$. We construct a simulator $\mathcal{S}_{\mathsf{lottery}}$ and argue that the views of the adversary in the simulated execution and real protocol execution are computationally close. Consider the simulator $\mathcal{S}_{\mathsf{lottery}}$.

Simulator $\mathcal{S}_{\mathsf{lottery}}$

Initialization

Upon first activation the simulator does the following.

1. For each dishonest stakeholder pid the simulator queries functionality $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ (using (GET-STAKE, sid)) to get the lottery powers α. Then, the simulator creates ($c_{\mathsf{pid}} = \mathsf{Com}(\alpha_{\mathsf{pid}}; r_{\mathsf{pid}})$, $r_{\mathsf{pid}}, \mathsf{sk}_{\mathsf{pid}}, \mathsf{vk}_{\mathsf{pid}}$) the same way as $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com,SIG}}$ would do.
2. For each honest stakeholder pid the simulator creates the tuple ($c_{\mathsf{pid}} = \mathsf{Com}(0), r_{\mathsf{pid}}, \mathsf{sk}_{\mathsf{pid}}, \mathsf{vk}_{\mathsf{pid}}$) similar to $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com,SIG}}$ except that the relative stake is set to 0.

Simulation of $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$

- Call $\mathcal{S}_1$ to generate a simulated CRS scrs and a trapdoor τ.
- Upon receiving $(\mathsf{Setup}, \mathrm{sid})$, output $(\mathsf{Setup}, \mathrm{sid}, \mathsf{scrs})$.

Simulation of $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com,SIG}}$

- Upon receiving (GetPrivateData, sid) from the adversary in the name of the dishonest stakeholder pid output (GetPrivateData, sid, α_{pid}, r_{pid}, $\mathsf{sk}_{\mathsf{pid}}$).
- Upon receiving (GetList, sid) from the adversary in the name of a dishonest party output the list $\mathcal{L} = \{(c_{\mathsf{pid}}, \mathsf{vk}_{\mathsf{pid}})_{\mathsf{pid}}\}$.

Simulation of $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$

The simulator maintains a table $T(\cdot,\cdot)$ and a list of vids.

- Upon input (KeyGen, sid) from the adversary in the name of dishonest stakeholder pid generate a unique key vid, record $(\mathsf{pid}, \mathsf{vid})$, and initialise the table $T(\mathsf{vid}, \cdot)$ to be empty. Return (KeyGen, sid, vid) to the adversary.
- Upon input (Eval, sid, vid, x) from the adversary in the name of dishonest stakeholder pid do the following:

```
Abort and ignore the request if $(\mathsf{pid}, \mathsf{vid})$ is undefined.
**if** $T(\mathsf{vid}, x)$ is undefined **then**
    **if** If $x \in \mathcal{E}$ **then**
        Send (LOTTERY, sid, $x$) in the name of $\mathsf{pid}$ to $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$.
        Denote by (LOTTERY, sid, $x, b$) the answer from $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$.
        Pick random value $y$ from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ such that $b = \mathsf{LE}(\alpha_{\mathsf{pid}}, y)$.[a]
    **else**
        Pick random value $y$ from $\{0,1\}^{\ell_{\mathsf{VRF}}}$.
    **end if**
    Pick random value $r$ from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set table $T(\mathsf{vid}, x) = (y, \mathsf{Com}(y; r), r)$.
**end if**
Output (Evaluated, sid, $T(\mathsf{vid}, x)$).
```

- Upon receiving a message $(\textsc{Verify}, \mathrm{sid}, x, c)$ from the adversary in the name of a dishonest party do the following:
  1. If there exists a vid such that $T(\mathsf{vid}, x) = (y, c, r)$ for some y, r then set $f = 1$.
  2. Else, set $f = 0$.
  3. Output $(\textsc{Verified}, \mathrm{sid}, x, c, f)$ to the adversary.

Simulation of $\mathcal{F}_{\mathsf{ABC}}$

The simulator maintains for each dishonest party a message buffer.

- Upon receiving $(\textsc{send}, \mathrm{sid}, \mathsf{e}, m)$ from $\mathcal{F}^{\mathsf{LE},\mathcal{E}}_{\mathsf{Lottery}}$ do the following:
  1. Create an entry $(\perp, \mathsf{vid})$ with unique vid for the internal $\mathcal{F}^{\mathsf{Com}}_{\mathsf{VRF}}$.
  2. Pick random values y, r from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set table $T(\mathsf{vid}, \mathsf{e}) = (y, \mathsf{Com}(y; r), r)$.
  3. Create simulated proof π_{zk} by calling the simulator $\mathcal{S}_2$ on (scrs, τ).
  4. Add $(e, m, \mathsf{Com}(y; r), \pi_{\mathsf{zk}})$ to the message buffers of all dishonest parties.
  5. Output $(\textsc{send}, \mathrm{sid}, (\mathsf{e}, m, \mathsf{Com}(y; r), \pi_{\mathsf{zk}}))$ to the adversary.
- Upon receiving $(\textsc{send}, \mathrm{sid}, m')$ from the adversary do the following:
  1. Add m' to all message buffers of dishonest parties.
  2. If $m' = (\mathsf{e}, m, c, \pi_{\mathsf{zk}})$ do:
     (a) Check that $\mathsf{e} \in \mathcal{E}$.
     (b) Check that there is vid such that $T(\mathsf{vid}, \mathsf{e}) = (y, c, r)$ for some y, r.
     (c) Check that $\mathsf{Verify}(\mathsf{scrs}, \pi_{\mathsf{zk}}) = 1$.
     (d) If all checks pass send $(\textsc{send}, \mathrm{sid}, \mathsf{e}, m)$ to $\mathcal{F}^{\mathsf{LE},\mathcal{E}}_{\mathsf{Lottery}}$.
  3. Output $(\textsc{Sent}, \mathrm{sid}, m')$ to the adversary.
- Upon receiving $(\textsc{Send}, \mathrm{sid}, m', P')$ from the adversary do the following:
  1. Add m' to message buffers of dishonest party P'.
  2. If $m' = (\mathsf{e}, m, c, \pi_{\mathsf{zk}})$ do:
     (a) Check that $\mathsf{e} \in \mathcal{E}$
     (b) Check that there is vid such that $T(\mathsf{vid}, \mathsf{e}) = (y, c, r)$ for some y, r.
     (c) Check that $\mathsf{Verify}(\mathsf{scrs}, \pi_{\mathsf{zk}}) = 1$.
     (d) If all checks pass and P' is honest send $(\textsc{send}, \mathrm{sid}, \mathsf{e}, m, P')$ to $\mathcal{F}^{\mathsf{LE},\mathcal{E}}_{\mathsf{Lottery}}$.
  3. Output $(\textsc{Sent}, \mathrm{sid}, m', P')$ to the adversary.
- Upon receiving $(\textsc{Receive}, \mathrm{sid})$ from the adversary in the name of corrupted party P. Remove all message from P's message buffer and output them to P.

[a] This requires that it is possible to efficiently sample randomness r satisfying $\mathsf{LE}(\mathsf{stk}, r) = b$ for given stake stk.

Let HYB_0 be the (distribution) of the protocol execution (in the hybrid world where the auxiliary functionalities are available). We consider the world HYB_1 which is the same as the protocol execution except for the following: calls to $\mathcal{F}^{\mathsf{Com}}_{\mathsf{VRF}}$ are answered as is done by the simulator $\mathcal{S}_{\mathsf{lottery}}$ consistent with the outcome returned by $\mathcal{F}^{\mathsf{LE},\mathcal{E}}_{\mathsf{Lottery}}$. It follows that distributions of HYB_0 and HYB_1 are indistinguishable. We now argue that the world HYB_1 is computationally indistinguishable from the ideal world simulation.

Simulation of $\mathcal{F}^{\mathsf{Com,SIG}}_{\mathsf{Init}}$ The only difference between HYB_1 and the simulation is that the list $\mathcal{L}$ consists of commitments to honest stakes in the protocol, whereas the commitments are to 0 in the interaction with the simulator. By the hiding property of the commitment scheme Com, the two distributions are identical.

Simulation of $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$ The CRS in HYB_1 is distributed the same as in the simulation.

Simulation of $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$

The key-generation and evaluation queries by the adversary are distributed the same. The same holds for verification queries where the adversary verifies a commitment which was created by an evaluation query by the adversary. In HYB_1, any other commitment message pair will be verified as true only if the commitment was part of an honest tuple $(\mathsf{e}, m, c, \pi_{\mathsf{zk}})$ which was sent to the adversary via $\mathcal{F}_{\mathsf{ABC}}$. Similarly, in the simulation any other commitment message pair will only be evaluated as true if the commitment was part of a simulated honest tuple.

Simulation of $\mathcal{F}_{\mathsf{ABC}}$

If the adversary sends a tuple $(\mathsf{e}, m, c, \pi_{\mathsf{zk}})$ in HYB_1, parties will accept it only if it is valid with respect to the information of $\mathcal{F}_{\mathsf{Init}}^{\mathsf{Com},\mathsf{SIG}}$, $\mathcal{F}_{\mathsf{crs}}^{\mathcal{D}}$, and $\mathcal{F}_{\mathsf{VRF}}^{\mathsf{Com}}$. In the ideal world, the simulator does the same checks with respect to the simulated functionalities. The simulator will then submit (e, m) to $\mathcal{F}_{\mathsf{Lottery}}^{\mathsf{LE},\mathcal{E}}$ which will send it to honest parties. The soundness of the zero-knowledge proof system and the binding property of the commitment scheme guarantee that the adversary can only submit tuples $(\mathsf{e}, m, c, \pi_{\mathsf{zk}})$ where the dishonest stakeholder won the lottery for e. Thus the distribution of HYB_1 and the ideal world is indistinguishable.

If in HYB_1 an honest stakeholder wins the lottery for entry e and publishes a message m via $\mathcal{F}_{\mathsf{ABC}}$, the adversary will receive a tuple of the form $(\mathsf{e}, m, c, \pi_{\mathsf{zk}})$. In the ideal world, the simulator gets (e, m) and creates a simulated tuple. By the zero-knowledge property of the proof system the distribution of HYB_1 and the ideal-world is indistinguishable. □

B Extended Preliminaries

B.1 Non-interactive Zero-knowledge

Definition 3 (Non-interactive Zero-knowledge Argument). *A non-interactive zero-knowledge argument for an NP relation R consists of a triple of polynomial time algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *defined as follows.*

- $\mathsf{Setup}(1^\kappa)$ *takes a security parameter κ and outputs a common reference string σ.*
- $\mathsf{Prove}(\sigma, x, w)$ *takes as input the CRS σ, a statement x, and a witness w, and outputs an argument π.*
- $\mathsf{Verify}(\sigma, x, \pi)$ *takes as input the CRS σ, a statement x, and a proof π, and outputs either* 1 *accepting the argument or* 0 *rejecting it.*

The algorithms above should satisfy the following properties.

1. *Completeness. For all $\kappa \in \mathbb{N}$, $(x, w) \in R$,*
$$\Pr\left(\mathsf{Verify}(\sigma, x, \pi) = 1 \;:\; \begin{array}{c}\sigma \leftarrow \mathsf{Setup}(1^\kappa)\\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w)\end{array}\right) = 1.$$
2. *Computational soundness. For all PPT adversaries $\mathcal{A}$, the following probability is negligible in κ:*
$$\Pr\left(\begin{array}{c}\mathsf{Verify}(\sigma, \tilde{x}, \tilde{\pi}) = 1\\ \wedge\, \tilde{x} \notin L\end{array} \;:\; \begin{array}{c}\sigma \leftarrow \mathsf{Setup}(1^\kappa)\\ (\tilde{x}, \tilde{\pi}) \leftarrow \mathcal{A}(1^\kappa, \sigma)\end{array}\right).$$
3. *Zero-knowledge. There exists a PPT simulator $(\mathcal{S}_1, \mathcal{S}_2)$ such that $\mathcal{S}_1$ outputs a simulated CRS Σ and trapdoor τ; $\mathcal{S}_2$ takes as input σ, a statement s and τ, and outputs a simulated proof π; and, for all PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, the following probability is negligible in κ:*
$$\left|\Pr\left(\begin{array}{c}(x, w) \in R \,\wedge\\ \mathcal{A}_2(\pi, \mathsf{st}) = 1\end{array} \;:\; \begin{array}{c}\sigma \leftarrow \mathsf{Setup}(1^\kappa)\\ (x, w, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\kappa, \sigma)\\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w)\end{array}\right) - \Pr\left(\begin{array}{c}(x, w) \in R \,\wedge\\ \mathcal{A}_2(\pi, \mathsf{st}) = 1\end{array} \;:\; \begin{array}{c}(\sigma, \tau) \leftarrow \mathcal{S}_1(1^\kappa)\\ (x, w, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\kappa, \sigma)\\ \pi \leftarrow \mathcal{S}_2(\sigma, \tau, x)\end{array}\right)\right|.$$