it returns a new verification key v that is used to label a table. Two methods are provided for computing VRF values. The first provides just the VRF output and does not interact with the adversary. In the second, whenever invoked on an input m that is not asked before by a stakeholder that is associated to a certain table labeled by v, the functionality will query the adversary for the value of the proof π, and subsequently sample a random element ρ to associate with m. Verification is always consistent and will validate outputs that have already being inserted in a table. Unpredictability against malicious key generation is captured by imposing the same random selection of outputs even for the function tables that correspond to keys of corrupted stakeholders. Finally, the adversary is allowed to query all function tables maintained by the functionality for which either a proof has been computed, or they correspond to adversarial keys. In Appendix C, we show how to realize $\mathcal{F}_{\mathsf{VRF}}$ in the random oracle model under the CDH assumption based on the 2-Hash-DH verifiable oblivious PRF construction of [JKK14].

Review

Functionality $\mathcal{F}_{\mathsf{VRF}}$.

$\mathcal{F}_{\mathsf{VRF}}$ interacts with stakeholders $U_1, \ldots, U_n$ as follows:

- **Key Generation.** Upon receiving a message (KeyGen, sid) from a stakeholder U_i, hand $(\mathsf{KeyGen}, sid, U_i)$ to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, U_i, v)$ from the adversary, if U_i is honest, verify that v is unique, record the pair (U_i, v) and return $(\mathsf{VerificationKey}, sid, v)$ to U_i. Initialize the table $T(v, \cdot)$ to empty.
- **Malicious Key Generation.** Upon receiving a message $(\mathsf{KeyGen}, sid, v)$ from $\mathcal{S}$, verify that v has not being recorded before; in this case initialize table $T(v, \cdot)$ to empty and record the pair $(\mathcal{S}, v)$.
- **VRF Evaluation.** Upon receiving a message (Eval, sid, m) from U_i, verify that some pair (U_i, v) is recorded. If not, then ignore the request. Then, if the value $T(v, m)$ is undefined, pick a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(v, m) = (y, \emptyset)$. Then output $(\mathsf{Evaluated}, sid, y)$ to P, where y is such that $T(v, m) = (y, S)$ for some S.
- **VRF Evaluation and Proof.** Upon receiving a message $(\mathsf{EvalProve}, sid, m)$ from U_i, verify that some pair (U_i, v) is recorded. If not, then ignore the request. Else, send $(\mathsf{EvalProve}, sid, U_i, m)$ to the adversary. Upon receiving $(\mathsf{Eval}, sid, m, \pi)$ from the adversary, if value $T(v, m)$ is undefined, verify that π is unique, pick a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(v, m) = (y, \{\pi\})$. Else, if $T(v, m) = (y, S)$, set $T(v, m) = (y, S \cup \{\pi\})$. In any case, output $(\mathsf{Evaluated}, sid, y, \pi)$ to P.
- **Malicious VRF Evaluation.** Upon receiving a message $(\mathsf{Eval}, sid, v, m)$ from $\mathcal{S}$ for some v, do the following. First, if $(\mathcal{S}, v)$ is recorded and $T(v, m)$ is undefined, then choose a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(v, m) = (y, \emptyset)$. Then, if $T(v, m) = (y, S)$ for some $S \neq \emptyset$, output $(\mathsf{Evaluated}, sid, y)$ to $\mathcal{S}$, else ignore the request.
- **Verification.** Upon receiving a message $(\mathsf{Verify}, sid, m, y, \pi, v')$ from some party P, send $(\mathsf{Verify}, sid, m, y, \pi, v')$ to the adversary. Upon receiving $(\mathsf{Verified}, sid, m, y, \pi, v')$ from the adversary do:
  1. If $v' = v$ for some (U_i, v) and the entry $T(U_i, m)$ equals (y, S) with $\pi \in S$, then set $f = 1$.
  2. Else, if $v' = v$ for some (U_i, v), but no entry $T(U_i, m)$ of the form $(y, \{\ldots, \pi, \ldots\})$ is recorded, then set $f = 0$.
  3. Else, initialize the table $T(v', \cdot)$ to empty, and set $f = 0$.

  Output $(\mathsf{Verified}, sid, m, y, \pi, f)$ to P.

Fig. 2: Functionality $\mathcal{F}_{\mathsf{VRF}}$.

Theorem 2. *The 2Hash-DH construction presented in Appendix C, realizes $\mathcal{F}_{\mathsf{VRF}}$ in the random oracle model assuming the CDH.*

3.3 Oblivious Leader Selection and $\mathcal{F}_{\mathsf{INIT}}$

As in (synchronous) Ouroboros, for each $0 < j \leq R$, a *slot leader* E_j is a stakeholder who is elected to generate a block at sl_j. However, our leader selection process differs from Ouroboros [KRDO17] in three points: (1) potentially, multiple slot leaders may be elected for a particular slot (forming a *slot leader set*); (2) frequently, slots will have *no leaders* assigned to them; and (3) a priori, only a slot leader is aware that it is indeed a leader for a given slot; this assignment is unknown to all the other stakeholders including other slot leaders of the same slot until the other stakeholders receive a valid block from this slot leader. The combinatorial analysis presented in Section 4 shows (with an honest stake majority) that (i.) blockchains generated according to these dynamics are well-behaved even if multiple slot leaders are selected for a slot and that (ii.) sequences of slots with no leader provide sufficient stability for honest stakeholders to effectively synchronize. As a matter of terminology, we call slots with an associated nonempty slot leader set *active slots* and slots that are not assigned a slot leader *empty slots.*

of the notion of verifiable pseudorandom permutations, cf. [DP07], could potentially be used towards a standard model instantiation of the primitive.

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

$\mathcal{F}_{\mathsf{INIT}}$ incorporates the delayed diffuse functionality from Section 2.2 and is parameterized by the number of initial stakeholders n and their respective stakes $s_1, \ldots, s_n$. $\mathcal{F}_{\mathsf{INIT}}$ interacts with stakeholders $U_1, \ldots, U_n$ as follows:

- In the first round, upon a request from some stakeholder U_i of the form $(\mathsf{ver\_keys}, sid, U_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{dsig}})$, it stores the verification keys tuple $(U_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{dsig}})$ and acknowledges its receipt. If any of the n stakeholders does not send a request of this form to $\mathcal{F}_{\mathsf{INIT}}$, it halts. Otherwise, it samples and stores a random value $\eta \xleftarrow{\$} \{0,1\}^{\lambda}$ and constructs a genesis block $(\mathbb{S}_0, \eta)$, where $\mathbb{S}_0 = \left((U_1, v_1^{\mathrm{vrf}}, v_1^{\mathrm{kes}}, v_1^{\mathrm{dsig}}, s_1), \ldots, (U_n, v_n^{\mathrm{vrf}}, v_n^{\mathrm{kes}}, v_n^{\mathrm{dsig}}, s_n)\right)$.
- In later rounds, upon a request of the form $(\mathsf{genblock\_req}, sid, U_i)$ from some stakeholder U_i, $\mathcal{F}_{\mathsf{INIT}}$ sends $(\mathsf{genblock}, sid, \mathbb{S}_0, \eta)$ to U_i.

Fig. 3: Functionality $\mathcal{F}_{\mathsf{INIT}}$.

The idealized slot leader assignment and the active slots coefficient. The fundamental leader assignment process calls for a stakeholder U_i to be independently selected as a leader for a particular slot sl_j with probability p_i depending only on its relative stake. (In this static-stake analysis, relative stake is simply determined by the genesis block B_0.) The exact relationship between p_i and the relative stake α_i is determined by a parameter f of the protocol which we refer to as the *active slots coefficient*. Specifically,

$$p_i = \phi_f(\alpha_i) \triangleq 1 - (1-f)^{\alpha_i} \,, \tag{1}$$

where α_i is the relative stake held by stakeholder U_i. We occasionally drop the subscript f and write $\phi(\alpha_i)$ when f can be inferred from context. As the events "U_i is a leader for sl_j" are independent, this process may indeed generate multiple (or zero) leaders for a given slot.

Remarks about $\phi_f(\cdot)$. Observe that $\phi_f(1) = f$; in particular, the parameter f is the probability that a party holding all the stake will be selected to be a leader for given slot. On the other hand, $\phi_f()$ is not linear, but slightly concave; see Figure 7. To motivate the choice of the function ϕ_f, we note that it satisfies the "independent aggregation" property:

$$1 - \phi\left(\sum_i \alpha_i\right) = \prod_i (1 - \phi(\alpha_i)) \,. \tag{2}$$

In particular, when leadership is determined according to ϕ_f, the probability of a stakeholder becoming a slot leader in a particular slot is independent of whether this stakeholder acts as a single party in the protocol, or splits its stake among several "virtual" parties. In particular, consider a party U with relative stake α who contrives to split its stake among two virtual subordinate parties with stakes α_1 and α_2 (so that $\alpha_1 + \alpha_2 = \alpha$). Then the probability that one of these virtual parties is elected for a particular slot is $1 - (1 - \phi(\alpha_1))(1 - \phi(\alpha_2))$, as these events are independent. Property (2) guarantees that this is identical to $\phi(\alpha)$. *Thus this selection rule is invariant under arbitrary reapportionment of a party's stake among virtual parties.*