and σ_j, a signature on $(st, d, sl_j, B_{\pi j})$ under the signing key for the time period of slot sl_j of the stakeholder U_i generating the block.

A blockchain *(or simply* chain*) relative to the genesis block B_0 is a sequence of blocks $B_1, \ldots, B_n$ associated with a strictly increasing sequence of slots for which the state st_i of B_i is equal to $H(B_{i-1})$, where H is a prescribed collision-resistant hash function. The* length *of a chain $\mathrm{len}(\mathcal{C}) = n$ is its number of blocks. The block B_n is the* head *of the chain, denoted $\mathrm{head}(\mathcal{C})$. We treat the empty string ε as a legal chain and by convention set $\mathrm{head}(\varepsilon) = \varepsilon$. Let $\mathcal{C}$ be a chain of length n and k be any non-negative integer. We denote by $\mathcal{C}^{\lceil k}$ the chain resulting from removal of the k rightmost blocks of $\mathcal{C}$. If $k > \mathrm{len}(\mathcal{C})$ we define $\mathcal{C}^{\lceil k} = \varepsilon$. We let $\mathcal{C}_1 \prec \mathcal{C}_2$ indicate that the chain $\mathcal{C}_1$ is a prefix of the chain $\mathcal{C}_2$.*

An epoch *is a set of R adjacent slots $S = \{sl_1, \ldots, sl_R\}$. (The value R is a parameter of the protocol we analyze in this section.)*

We consider as valid blocks that are generated by a stakeholder in the slot leader set of the slot to which the block is attributed. Later in Section 3.3 we discuss slot leader sets and how they are selected.

Definition 3 (Absolute and Relative Stake). *Let $U_{\mathcal{P}}$, $U_{\mathcal{A}}$ and $U_{\mathcal{H}}$ denote the sets of all stakeholders, the set of stakeholders controlled by an adversary $\mathcal{A}$, and the remaining (honest) stakeholders, respectively. For any party (resp. set of parties) X we denote by s_X^+ (resp. s_X^-) the maximum (resp. minimum)* absolute stake *controlled by X in the view of all honest stakeholders at a given slot, and by $\alpha_X^+ \triangleq s_X^+/s_{\mathcal{P}}$ and $\alpha_X \triangleq s_X/s_{\mathcal{P}}$ its* relative stake *taken as maximum and minimum respectively across of the view of all honest stakeholders. For simplicity, we use s_X^s, α_X^s instead of s_{U_X}, α_{U_X} for all $X \in \{\mathcal{P}, \mathcal{A}, \mathcal{H}\}, s \in \{+, -\}$. We also call $\alpha_{\mathcal{A}} \triangleq \alpha_{\mathcal{A}}^+$ and $\alpha_{\mathcal{H}} \triangleq \alpha_{\mathcal{H}}^-$ the* adversarial stake ratio *and* honest stake ratio*, respectively.*

3.1 Forward Secure Signatures and $\mathcal{F}_{\mathsf{KES}}$

In regular digital signature schemes, an adversary who compromises the signing key of a user can generate signatures for any messages it wishes, including messages that were (or should have been) generated in the past. Forward secure signature schemes [BM99] prevent such an adversary from generating signatures for messages that were issued in the past, or rather allows honest users to verify that a given signature was generated at a certain point in time. Basically, such security guarantees are achieved by "evolving" the signing key after each signature is generated and erasing the previous key in such a way that the actual signing key used for signing a message in the past cannot be recovered but a fresh signing key can still be linked to the previous one. This notion is formalized through *key evolving signature schemes*, which allow signing keys to be evolved into fresh keys for a number of time periods. We remark that efficient constructions of key evolving signature schemes with forward security exist [IR01] but no previous work has fully specified them in the UC setting. Previous (game-based) definitions are presented in Appendix A.3.

We present a UC definition of the type of key-evolving signatures that we will take advantage of in our constructions. $\mathcal{F}_{\mathsf{KES}}$ allows us to achieve forward security with erasures (*i.e.*, assuming that parties securely delete old signing keys as the protocol proceeds). This functionality embodies ideal key evolving signature schemes allowing an adversary that corrupts the signer to forge signatures only under the current and future signing keys, but not under a previous signing key that has been updated. Our starting point for $\mathcal{F}_{\mathsf{KES}}$ is the standard digital signature functionality defined in [Can04] with the difference that packs together with the signing operation a key-evolving operation. During verification, $\mathcal{F}_{\mathsf{KES}}$ lets the adversary set the response to a verification query (taking as input a given time period) only if no key update has been performed since that time period and no entry exists in its internal table for the specific message, signature and time period specified in the query. We present $\mathcal{F}_{\mathsf{KES}}$ in Figure 1. In Appendix B, we will show that $\mathcal{F}_{\mathsf{KES}}$ can be realized by a construction based on key evolving signature schemes as defined in Appendix A.3.

Theorem 1. *The π_{KES} construction presented in Appendix B, realizes $\mathcal{F}_{\mathsf{KES}}$ with erasures assuming $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ is a key evolving signature scheme with forward security as per Definition 15 and Definition 17.*

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

$\mathcal{F}_{\mathsf{KES}}$ is parameterized by the total number of signature updates T, interacting with a signer U_S and stakeholders U_i as follows:

- **Key Generation.** Upon receiving a message $(\mathsf{KeyGen}, sid, U_S)$ from a stakeholder U_S , send $(\mathsf{KeyGen}, sid, U_S)$ to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, U_S, v)$ from the adversary, send $(\mathsf{VerificationKey}, sid, v)$ to U_S, record the triple (sid, U_S, v) and set counter $\mathsf{k}_{\mathrm{ctr}} = 1$.
- **Sign and Update.** Upon receiving a message $(\mathsf{USign}, sid, U_S, m, j)$ from U_S, verify that (sid, U_S, v) is recorded for some sid and that $\mathsf{k}_{\mathrm{ctr}} \leq j \leq T$. If not, then ignore the request. Else, set $\mathsf{k}_{\mathrm{ctr}} = j + 1$ and send $(\mathsf{Sign}, sid, U_S, m, j)$ to the adversary. Upon receiving $(\mathsf{Signature}, sid, U_S, m, j, \sigma)$ from the adversary, verify that no entry $(m, j, \sigma, v, 0)$ is recorded. If it is, then output an error message to U_S and halt. Else, send $(\mathsf{Signature}, sid, m, j, \sigma)$ to U_S, and record the entry $(m, j, \sigma, v, 1)$.
- **Signature Verification.** Upon receiving a message $(\mathsf{Verify}, sid, m, j, \sigma, v')$ from some stakeholder U_i do:
  1. If $v' = v$ and the entry $(m, j, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
  2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, j, \sigma', v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, j, \sigma, v, 0)$. (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
  3. Else, if there is an entry (m, j, σ, v', f') recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
  4. Else, if $j < \mathsf{k}_{\mathrm{ctr}}$, let $f = 0$ and record the entry $(m, j, \sigma, v, 0)$. Otherwise, if $j = \mathsf{k}_{\mathrm{ctr}}$, hand $(\mathsf{Verify}, sid, m, j, \sigma, v')$ to the adversary. Upon receiving $(\mathsf{Verified}, sid, m, j, \phi)$ from the adversary let $f = \phi$ and record the entry (m, j, σ, v', ϕ). (This condition guarantees that the adversary is only able to forge signatures under keys belonging to corrupted parties for time periods corresponding to the current or future slots.)

  Output $(\mathsf{Verified}, sid, m, j, f)$ to U_i.

Fig. 1: Functionality $\mathcal{F}_{\mathsf{KES}}$.

3.2 UC-VRFs with Unpredictability Under Malicious Key Generation

The usual pseudorandomness definition for VRFs (as stated in Appendix A.1, Definition 14) captures the fact that an attacker, seeing a number of VRF outputs and proofs for adversarially chosen inputs under a key pair that is correctly generated by a challenger, cannot distinguish the output of the VRF on a new (also adversarially chosen) input from a truly random string. This definition is too weak for our purposes for two reasons: first, we need a simulation-based definition so that the VRF can be composed directly within our protocol; second, we need the primitive to provide some level of unpredictability even under malicious key generation, *i.e.*, against adversaries who are allowed to generate the secret and pubic key pair.

Our UC formulation of VRFs cannot be implied by the standard VRF security definition or even the simulatable VRF notion of [CL07]. For instance, the VRF proofs in our setting have to be simulatable without knowlege of the VRF output (which is critical as we would like to ensure that the VRF output is not leaked to the adversary prematurely); it is easy to construct a VRF that is secure in the standard definition, but it is impossible to simulate its proofs without knowledge of the VRF output. Furthermore, if the adversary is allowed to generate its own key pair it is easy to see that the distribution of the VRF outputs cannot be guaranteed. Indeed, even for known constructions (*e.g.* [DY05]), an adversary that maliciously generates keys can easily and significantly skew the output distribution.

We call the latter property *unpredictability under malicious key generation* and we present, in Figure 2, a UC definition for VRF's that captures this stronger security requirement.[4] The functionality operates as follows. Given a key generation request from one of the stakeholders,

[4] In fact our UC formulation captures a stronger notion: even for adversarial keys the VRF function will act as a random oracle. We note that while we can achieve this notion in the random oracle model, a weaker condition of mere unpredictability can be sufficient for the security of our protocol. A UC version