## 2016-889.pdf

916.77 KB • 7748 extracted lines

This PDF was converted to text due to its large size.

Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol

Aggelos Kiayias

∗

Alexander Russell

†

Bernardo David

‡

Roman Oliynykov

§

July 20, 2019

Abstract

We present “Ouroboros,” the first blockchain protocol based on proof of stake with rig-

orous security guarantees. We establish security properties for the protocol comparable to

those achieved by the bitcoin blockchain protocol. As the protocol provides a “proof of stake”

blockchain discipline, it offers qualitative efficiency advantages over blockchains based on proof

of physical resources (e.g., proof of work). We also present a novel reward mechanism for in-

centivizing proof of stake protocols and we prove that, given this mechanism, honest behavior

is an approximate Nash equilibrium, thus neutralizing attacks such as selfish mining. We also

present initial evidence of the practicality of our protocol in real world settings by providing

experimental results on transaction confirmation and processing.

1 Introduction

A primary consideration regarding the operation of blockchain protocols based on proof of work

(PoW)—such as bitcoin [34]—is the energy required for their execution. At the time of this writ-

ing, generating a single block on the bitcoin blockchain requires a number of hashing operations

exceeding 2

60

, which results in striking energy demands. Indeed, early calculations indicated that

the energy requirements of the protocol were comparable to that of a small country [36].

This state of affairs has motivated the investigation of alternative blockchain protocols that

would obviate the need for proof of work by substituting it with another, more energy efficient,

mechanism that can provide similar guarantees. Fundamentally, the proof of work mechanism of

bitcoin facilitates a type of robust randomized “leader election” process that elects one of the miners

to issue the next block. Furthermore—provided that all miners follow the protocol—this selection

is performed in a randomized fashion proportionally to the computational power of each miner.

(Deviations from the protocol may distort this proportionality as exemplified by “selfish mining”

strategies [25, 43].)

A natural alternative mechanism relies on the notion of “proof of stake” (PoS). Rather than

miners investing computational resources in order to participate in the leader election process, they

instead run a process that randomly selects one of them proportionally to the stake that each

possesses according to the current blockchain ledger.

∗

University of Edinburgh and IOHK. akiayias@inf.ed.ac.uk. Work partly performed while at the National and

Kapodistrian University of Athens, supported by ERC project CODAMODA #259152. Work partly supported by

H2020 Project #653497, PANORAMIX.

†

University of Connecticut and IOHK. acr@cse.uconn.edu.

‡

Aarhus University and IOHK, bernardo@bmdavid.com. Work partly supported by European Research Council

Starting Grant 279447.

§

IOHK, roman.oliynykov@iohk.io.

1

In effect, this yields a self-referential blockchain discipline: maintaining the blockchain relies on

the stakeholders themselves and assigns work to them (as well as rewards) based on the amount

of stake that each possesses as reported in the ledger. Aside from this, the protocol should make

no further “artificial” computational demands on the stakeholders. In some sense, this sounds

ideal; however, realizing such a proof of stake protocol appears to involve a number of definitional,

technical, and analytic challenges.

Previous work. The concept of PoS has been discussed extensively in the bitcoin forum.

1

Proof

of stake based blockchain design has been more formally studied by Bentov et al., both in conjunc-

tion with PoW [7] as well as the sole mechanism for a blockchain protocol [6]. Although Bentov

et al. showed that their protocols are secure against some classes of attacks, they do not provide

a formal model for analyzing PoS based protocols or security proofs relying on precise definitions.

Heuristic proof of stake based blockchain protocols have been proposed (and implemented) for a

number of cryptocurrencies.

2

Being based on heuristic security arguments, these cryptocurrencies

have been frequently found to be deficient from the point of view of security. See [6] for a discussion

of various attacks.

It is also interesting to contrast a PoS-based blockchain protocol with a classical consensus

blockchain that relies on a fixed set of authorities (see, e.g., [21]). What distinguishes a PoS-based

blockchain from those which assume static authorities is that stake changes over time and hence

the trust assumption evolves with the system as well as the fact that the complexity of system

maintenance should be sublinear in the total number of users/stakeholders.

Another alternative to PoW is the concept of proof of space [2, 24], which has been specifically

investigated in the context of blockchain protocols [37]. In a proof of space setting, a “prover”

wishes to demonstrate the utilization of space (storage/memory); as in the case of a PoW, this

utilizes a physical resource but can demand less energy. A related concept is proof of space-time

(PoST) [32]. In all these cases, however, the protocol relies on an expensive physical resource (either

storage or computational power).

The PoS Design challenge. A fundamental problem for PoS-based blockchain protocols is to

simulate the leader election process. In order to achieve a fair randomized election among stake-

holders, entropy must be introduced into the system, and mechanisms to introduce entropy may be

prone to manipulation by the adversary. For instance, an adversary controlling a set of stakeholders

may attempt to simulate the protocol execution trying different sequences of stakeholder partici-

pants so that it finds a protocol continuation that favors the adversarial stakeholders. This leads

to a so called “grinding” vulnerability, where adversarial parties may use computational resources

to bias the leader election.

Our Results. We present “Ouroboros,” a provably secure proof of stake system. To the best of

our knowledge this is the first blockchain protocol of its kind with a rigorous security analysis. In

more detail, our results are as follows.

First, we provide a model that formalizes the problem of realizing a PoS-based blockchain proto-

col. The model we introduce is in the spirit of [28], focusing on persistence and liveness, two formal

properties of a robust transaction ledger. Persistence states that once a node of the system pro-

claims a certain transaction as “stable,” the remaining nodes, if queried and responding honestly,

1

See “Proof of stake instead of proof of work”, Bitcoin forum thread. Posts by user “QuantumMechanic” and

others. (https://bitcointalk.org/index.php?topic=27787.0.).

2

A non-exhaustive list includes NXT, Neucoin, Blackcoin, Tendermint, Bitshares.

2

will also report it as stable. Here, stability is to be understood as a predicate that will be parame-

terized by some security parameter k that will affect the certainty with which the property holds.

(E.g., “more than k blocks deep.”) Liveness ensures that once an honestly generated transaction

has been made available for a sufficient amount of time to the network nodes, say u time steps,

it will become stable. The conjunction of liveness and persistence provides a robust transaction

ledger in the sense that honestly generated transactions are adopted and become immutable. Our

model is suitably adapted to reflect PoS-based dynamics.

Second, we describe a novel blockchain protocol based on PoS. Our protocol assumes that

parties can freely create accounts and receive and make payments, and that stake shifts over time.

We utilize a (simple) secure multiparty implementation of a coin-flipping protocol to produce the

randomness for the leader election process. This distinguishes our approach (and prevents so called

“grinding attacks”) from other previous solutions that either defined such values deterministically

based on the current state of the blockchain or used collective coin flipping as a way to introduce

entropy [6]. Also, unique to our approach is the fact that the system ignores round-to-round

stake modifications. Instead, a snapshot of the current set of stakeholders is taken in regular

intervals called epochs; in each such interval a secure multiparty computation takes place utilizing

the blockchain itself as the broadcast channel. Specifically, in each epoch a set of randomly selected

stakeholders form a committee which is then responsible for executing the coin-flipping protocol.

The outcome of the protocol determines the set of elected stakeholders that will execute the protocol

in the subsequent epoch, as well as the outcomes of all leader elections for the epoch.

Third, we provide a set of formal arguments establishing that no adversary can break persistence

and liveness. Our protocol is secure under a number of plausible assumptions: (1) the network

is synchronous in the sense that an upper bound can be determined during which any honest

stakeholder is able to communicate with any other stakeholder, (2) a number of stakeholders drawn

from the honest majority is available as needed to participate in each epoch, (3) the stakeholders

do not remain offline for long periods of time, (4) the adaptivity of corruptions is subject to a small

delay that is measured in rounds linear in the security parameter (or alternatively, the players

have access to a sender-anonymous broadcast channel). At the core of our security arguments is a

probabilistic argument regarding a combinatorial notion of “forkable strings” which we formulate,

prove and also investigate experimentally. In our analysis we also distinguish covert attacks, a

special class of general forking attacks. “Covertness” here is interpreted in the spirit of covert

adversaries against secure multiparty computation protocols, cf. [3], where the adversary wishes to

break the protocol but prefers not to be caught doing so. We show that covertly forkable strings are

a subclass of the forkable strings with much smaller density; this permits us to provide two distinct

security arguments that achieve different trade-offs in terms of efficiency and security guarantees.

Our forkable string technique is a natural and general tool that may have applications beyond

analysis of our specific PoS protocol.

Fourth, we turn our attention to the incentive structure of the protocol. We present a novel

reward mechanism for incentivizing the participants to the system which we prove to be an (ap-

proximate) Nash equilibrium. In this way, attacks like block withholding and selfish-mining [25, 43]

are mitigated by our design. The core idea behind the reward mechanism is to provide positive

payoff for those protocol actions that cannot be stifled by a coalition of parties that diverges from

the protocol. In this way, it is possible to show that under plausible assumptions—namely that

certain protocol execution costs are small—following the protocol faithfully is an equilibrium when

all players are rational.

Fifth, we introduce a stake delegation mechanism that can be seamlessly added to our blockchain

protocol. Delegation is particularly useful in our context as we would like to allow our protocol

to scale even in a setting where the set of stakeholders is highly fragmented. In such cases, the

3

delegation mechanism can enable stakeholders to delegate their “voting rights,” i.e., the right of

participating in the committees running the leader selection protocol in each epoch. As in liquid

democracy (a.k.a. delegative democracy [27]), stakeholders have the ability to revoke their delegative

appointment when they wish, independently of each other.

Given our model and protocol description we also explore how various attacks considered in

practice can be addressed within our framework. Specifically, we discuss double spending attacks,

transaction denial attacks, 51% attacks, nothing-at-stake, desynchronization attacks and others.

Finally, we present evidence for the practical efficiency of our design. First we consider double

spending attacks. For illustrative purposes, we perform a comparison with Nakamoto’s analysis for

bitcoin regarding transaction confirmation time with assurance 99.9%. Against covert adversaries,

the transaction confirmation time is from 10 to 16 times faster than that of bitcoin, depending on

the adversarial hashing power; for general adversaries confirmation time is from 5 to 10 times faster.

Moreover, our concrete analysis of double-spending attacks relies on our combinatorial analysis of

forkable and covertly forkable strings and applies to a much broader class of adversarial behavior

than Nakamoto’s more simplified analysis.

3

We then survey our prototype implementation and

report on benchmark experiments run in the Amazon cloud that showcase the power of our proof

of stake blockchain protocol in terms of performance.

Related and Follow-up Work. In parallel to the development of Ouroboros, a number of other

protocols were developed targeting various positions in the design space of distributed ledgers based

on PoS. Sleepy consensus [8] considers a fixed stakeholder distribution (i.e., stake does not evolve

over time) and targets a “mixed” corruption setting, where the adversary is allowed to be adaptive

as well as perform fail-stop and recover corruptions in addition to Byzantine faults. It is straight-

forward to extend our analysis in this mixed corruption setting, cf. Remark 2; nevertheless, the

resulting security can be argued only in the “corruptions with delay” setting, and thus is not fully

adaptive; in follow up work it is shown how our analysis can be extended to incorporate both

adaptive corruptions [22] as well as a more refined model of availability [4] that covers fail-stop and

recover/sleepy corruptions. Snow White [9] addresses an evolving stakeholder distribution and uses

a corruption delay mechanism similar to ours for arguing security. Contrary to our approach here

the leader election of [9] enables a degree of “grinding”, i.e., making it feasible to significantly bias

any adversarially chosen high probability event but affords a much more efficient randomness gen-

eration process. In follow-up work [22] it is shown how it is possible to obtain full adaptive security

and similarly efficient randomness generation extending the forkable strings analysis introduced in

the present paper. Algorand [31] provides a distributed ledger following a Byzantine agreement

per block approach that can withstand adaptive corruptions. Given that agreement needs to be

reached for each block, such protocols will produce blocks at a rate substantially slower than a PoS

blockchain (where the slow down matches the expected length of the execution of the Byzantine

agreement protocol); on the other hand, Algorand can be parameterised to be free of forks and

then blocks will “settle” immediately after the conclusion of each consensus sub-protocol. In par-

ticular full settlement will happen at expected constant number of rounds, which, asymptotically,

is superior to any blockchain protocol (it is worth noting that blockchain protocols somewhat mit-

igate this inherent slow-down by letting ledger users choose a different level of settlement security

depending on the value of transactions that are at risk or by using an overlay protocol such as

lightning [41]). This benefit comes at the expense of imposing specific participation bounds for

3

Nakamoto’s simplifications are pointed out in [28]: the analysis considers only the setting where a block with-

holding attacker acts without interaction as opposed to a more general attacker that, for instance, tries strategically

to split the honest parties in more than one chains during the course of the double spending attack.

4

each sub-protocol instance which are unnecessary in the case of a blockchain protocol, see the fol-

low up work on Ouroboros with dynamic availability [4] for further discussion on this topic. The

Algorand approach also does not require an agreed concept of time between the participants and

merely relies on the assumption that time passes with roughly the same speed. While the protocol

presented herein requires access to global time, in follow-up work, [5], it is shown how it is pos-

sible for the Ouroboros protocol to be enhanced with a time synchronization sub-protocol while

retaining the core security argument we present here. Fruitchain [40] provides a reward mechanism

and an approximate Nash equilibrium proof for a PoW-based blockchain. We use a similar reward

mechanism at the blockchain level, nevertheless our underlying mechanics are different since we

operate in a PoS setting. The core of the idea is to provide a PoS analogue of “endorsing” inputs in

a fair proportion using the same logic as the PoW-based byzantine agreement protocol for honest

majority from [28]. Finally, in the present paper we prove that the error bound for the common-

prefix property (and thus persistence of transactions) has an error rate that drops sub-exponentially

with the security parameter, while experimentally we demonstrate an exponential drop. Follow up

work [42] shows how this gap can be closed by proving an exponentially decreasing error bound.

Paper overview. We lay out the basic model in Sec. 2. To simplify the analysis of our protocol,

we present it in four stages that are outlined in Sec. 3. In short, in Sec. 4 we describe and analyze

the protocol in the static setting; we then transition to the dynamic setting in Sec. 5. We then

present the protocol enhancement with anonymous channels in Sec. 7. Our incentive mechanism

and the equilibrium argument are presented in Sec. 8. Following this, we discuss delegation in

Sec. 9 and in Sec. 10, the resilience of the protocol under various particular attacks of interest.

Finally, in Sec. 11 we discuss transaction confirmation times as well as general performance results

obtained from a prototype implementation running in the Amazon cloud.

2 Model

Time, slots, and synchrony. We consider a setting where time is divided into discrete units

called slots. A ledger, described in more detail below, associates with each time slot (at most) one

ledger block. Players are equipped with (roughly synchronized) clocks that indicate the current slot.

This will permit them to carry out a distributed protocol intending to collectively assign a block

to this current slot. In general, each slot is indexed by an integer r ∈ {1, 2, . . .}, and we assume

that the real time window that corresponds to each slot has the following properties.

• The current slot is determined by a publicly-known and monotonically increasing function of

current time.

• Each player has access to the current time. Any discrepancies between parties’ local time are

insignificant in comparison with the length of time represented by a slot.

• The length of the time window that corresponds to a slot is sufficient to guarantee that

any message transmitted by an honest party at the beginning of the time window will be

received by any other honest party by the end of that time window (even accounting for

small inconsistencies in parties’ local clocks). In particular, while network delays may occur,

they never exceed the slot time window.

Transaction Ledger Properties. A protocol Π implements a robust transaction ledger provided

that the ledger that Π maintains is divided into “blocks” (assigned to time slots) that determine the

5

order with which transactions are incorporated in the ledger. It should also satisfy the following

two properties.

• Persistence, with parameter k ∈ N. Once a node of the system proclaims a certain

transaction tx as stable, the remaining nodes, if queried, will report tx at the same position

of the ledger and agree on the entire prefix of the ledger (prior to tx). Stability is defined in

terms of the blockchain: a transaction is declared stable if and only if it is in a block that is

more than k blocks deep in the ledger.

• Liveness, with parameter u ∈ N. If all honest nodes in the system attempt to include

a certain transaction, then after the passing of time corresponding to u slots (called the

transaction confirmation time), all nodes, if queried and responding honestly, will report the

transaction as stable.

Persistence and liveness can be derived from the following three elementary properties [28, 30, 39]

provided that protocol Π derives the ledger from a data structure in the form of a blockchain (or

simply chain) that each party maintains locally and updates at the onset of each slot. The properties

are as follows.

• Common Prefix (CP); with parameter k ∈ N. The chains C

1

, C

2

adopted by two honest

parties at the onset of the slots sl

1

≤ sl

2

are such that C

dk

1

C

2

, where C

dk

1

denotes the chain

obtained by removing the last k blocks from C

1

, and denotes the prefix relation.

• Honest Chain Growth (HCG); with parameters τ ∈ (0, 1] and s ∈ N. Consider the

chain C adopted by an honest party. Let sl

2

be the slot associated with the last block of C

and let sl

1

be a prior slot in which C has an honestly-generated block. If sl

2

≥ sl

1

+ s, then

the number of blocks appearing in C after sl

1

is at least τ s. The parameter τ is called the

speed coefficient.

• Existential Chain Quality (∃CQ); with parameter s ∈ N. Consider the chain C adopted

by an honest party at the onset of a slot and any portion of C spanning s prior slots; then at

least one honestly-generated block appears in this portion.

Some remarks are in place. This definition of common prefix reflects a strong variant of that

appearing in [30]. As for chain quality, we work with this simple “existential” formulation for

convenience. Previous work focused on a more general notion that bounds the density of honestly-

generated blocks in a sufficiently long portion of a chain. In fact, one can establish such a density

bound directly from existential chain quality along the lines of the proof of chain growth in Sec-

tion 4.5.

As for chain growth, we focus on a version pertaining to the suffix of the chain following an

honest block. In fact, as we note in Section 4.5, combining ∃CQ and HCG one can directly infer the

following stronger version.

• Chain Growth (CG); with parameters τ ∈ (0, 1] and s ∈ N. Consider the chain C

adopted by an honest party at the onset of a slot and any portion of C spanning s prior slots;

then the number of blocks appearing in this portion of the chain is at least τ s.

As in the case of bitcoin, the longest chain plays a preferred role in our protocol; this provides

a straightforward guarantee of honest chain growth.

6

Security Model. We adopt the model introduced by [28] for analyzing security of blockchain

protocols enhanced with an ideal functionality F. We denote by VIEW

P,F

Π,A,Z

(κ) the view of party

P after the execution of protocol Π with adversary A, environment Z, security parameter κ and

access to ideal functionality F. Similarly we denote by EXEC

F

Π,A,Z

(κ) the output of Z.

We note that multiple different “functionalities” will be encompassed by F. Contrary to [28],

our analysis is in the “standard model,” and without a random oracle functionality. The first inter-

faces we incorporate in the ideal functionality used in the protocol are the “diffuse” and “key and

transaction” functionality, denoted F

D+KT

and described below. Note that the diffuse functionality

is also the mechanism via which we will obtain protocol synchronization.

Diffuse functionality. The diffuse functionality maintains an incoming string for each party U

i

that participates. A party, if activated, is allowed at any moment to fetch the contents of its

incoming string; one may think of this as a mailbox. Additionally, parties can instruct the

functionality to diffuse a message, in which case the message will be appended to each party’s

incoming string. The functionality maintains rounds (slots) and all parties activated in a

round are allowed to diffuse once. Rounds do not advance unless all activated parties have

diffused a message. The adversary, when activated, may also interact with the functionality;

it may read all inboxes and all diffuse requests and deliver messages to the inboxes in any

order it prefers. At the end of the round, the functionality will ensure that all inboxes contain

all messages that have been diffused (but not necessarily in the same order they have been

requested to be diffused). The current slot index may be requested at any time by any party.

If a party does not fetch in a certain slot the messages written to its incoming string, they

are flushed.

Key and Transaction functionality. The key registration functionality is initialized with n

users, U

1

, . . . , U

n

, and their respective stakes, s

1

, . . . , s

n

; given such initialization, the func-

tionality will consult with the adversary and will accept a (possibly empty) sequence of

(Corrupt, U ) messages and mark the corresponding users U as corrupt. For the corrupt users

without a registered public-key the functionality will allow the adversary to set their public-

keys. For honest users the functionality will sample public/secret-key pairs and record them

based on a digital signature algorithm; we will consider two modes for key generation. In the

ideal mode, key generation will be outsourced to an instance of an ideal signature function-

ality; in the real mode, an actual digital signature algorithm will be utilized; (cf. Figure 1

where we make these two variants explicit). Public-keys of corrupt users will be marked as

such. Subsequently, any sequence of the following actions may take place: (i) A user may

request to sign or verify a message whereupon the functionality will respond accordingly, re-

stricting signing requests to only the owners of the respective keys. (ii) The entire directory

of public-keys may be requested, whereupon the functionality will return it to the requesting

user. (iii) A new user may be requested to be created by a message (Create, U, C) from the

environment, in which case the functionality will follow the same procedure as above: it will

consult the adversary regarding the corruption status of U and will set its public and possibly

secret-key depending on the corruption status; additionally, it will store C as the suggested

initial state. The functionality will return the public-key to the environment upon successful

completion of this interaction. (iv) An existing user may be requested to be corrupted by the

adversary via a message (Corrupt, U ). A user can only be corrupted after a delay of D slots;

specifically, after a corruption request is registered, a timer is maintained and the corruption

will take place after D slots have passed according to the round counter maintained in the

Diffuse component of the functionality. Note that when running in real mode, a corruption

7

will have the effect of revealing the secret-key to the adversary. In any case, the adversary

will be able to access the signing interface of any corrupted party.

We assume throughout that the execution of the protocol is with respect to a functionality

F that incorporates, at least, the above two functionalities. Further functionalities are explained

below. Note that a corrupted stakeholder U will relinquish its entire state to A; from this point on,

the adversary will be activated in place of the stakeholder U . Beyond any restrictions imposed by F,

the adversary can only corrupt a stakeholder if it is given permission by the environment Z running

the protocol execution. The permission is in the form of a message (Corrupt, U ) which is provided

to the adversary by the environment. The environment can give arbitrary permissions, however

with respect to an initial stakeholder set and respective stake, the adversary will be restricted to

controlling only a percentage of that stake, say α. In such case we refer to this adversary as an

α-initially-bounded adversary. In summary, regarding activations we have the following.

• At each slot sl

j

, the environment Z is allowed to activate any subset of stakeholders it

wishes. Each one of them will possibly produce messages that are to be transmitted to other

stakeholders.

• The adversary is activated, at least, as the last entity in each sl

j

(as well as during all

adversarial party activations).

It is easy to see that the model above confers such sweeping power on the adversary that one

cannot establish any significant guarantees on protocols of interest. It is thus important to restrict

the environment suitably (taking into account the details of the protocol) so that we may be able

to argue security. With foresight, the restrictions we will impose on the environment are as follows.

Restrictions imposed on the environment. The environment, which is responsible for acti-

vating the honest parties in each round, will be subject to the following constraints regarding the

activation of the honest parties running the protocol.

• In each slot there will be at least one honest activated party.

• There will be a parameter k ∈ Z that will signify the maximum number of slots that an honest

shareholder can be offline. In case an honest stakeholder is spawned after the beginning of

the protocol via (Create, U, C) its initialization chain C provided by the environment should

match an honest parties’ chain which was active in the previous slot.

• The environment will distribute the transaction data d ∈ {0, 1}

∗

to all parties, emulating the

effect of diffusing transactions in a peer-to-peer network. This transaction data, including

the required signatures by each stakeholder, is obtained by the environment as specified in

the protocol.

• In each slot sl

r

, and for each active stakeholder U

j

, the view of the stakeholder will include a

set S

j

(r) of public-keys and stake pairs of the form (vk

i

, s

i

) ∈ {0, 1}

∗

×N, for j = 1, . . . , n

r

; here

n

r

is the number of users introduced up to that slot according to the view of U

j

. Public-keys

will be marked as “corrupted” if the corresponding stakeholder has been corrupted (or marked

for corrupted by a delayed adversary). We will say the adversary is (1/2 − δ)-bounded in a

particular execution for a parameter δ > 0 if it holds that the total stake of the corrupted keys

divided by the total stake

∑

i

s

i

is less than 1/2 − δ in all possible S

j

(r). (Note the distinction

between this notion—a property of the execution—and that of a (1/2 − δ)-initially-bounded

8

adversary, which refers to the initial distribution appearing in the genesis block.) If the above

is violated, we say the event Bad

1

/2−δ

occurs for the given execution.

• Stake shift is bounded over short periods. Specifically, for any chain adopted by an honest

party, the observed shift in stake distribution between “nearby” prefixes of the chain are

bounded from above. See Definition 5.2 for a precise definition.

We note that the availability assumption (restricting honest parties from long periods of dis-

connection) stated above is very conservative and our protocol can tolerate much longer offline

times depending on the course of the execution; nevertheless, for the sake of simplicity, we use

the above restriction. Finally, we note that in all of our proofs, whenever we say that a property

Q holds with high probability over all executions, we will in fact argue that Q ∨ Bad

1

/2−δ

holds

with high probability over all executions. This captures the fact that we exclude environments and

adversaries that trigger Bad

1

/2−δ

with non-negligible probability.

3 The Ouroboros Protocol: Overview of Design and Analysis

We first provide a general overview of the approach to design and analyze our protocol. The

protocol’s analysis will depend on a number of parameters: (i.) k is the number of blocks a certain

message should have “on top of it” in order to be treated as part of the immutable history of the

ledger, (ii.) , σ are parameters that bound adversarial stake and stake shift in the sense that the

adversary is (1/2 − )-initially bounded as well as (1/2 − − σ)-bounded throughout the execution

and moreover stake shifts no more than σ occur over short enough time periods, (iii.) D is the

corruption delay that is imposed on the adversary, i.e., an honest stakeholder will be corrupted D

slots after the corresponding “corrupt” message is generated during an execution; (iv.) L is the

lifetime of the system, measured in slots; (v.) R is the length of an epoch, measured in slots (see

below for a discussion).

We present our protocol description in four stages, successively improving the adversarial model

it can withstand. In all stages an “ideal functionality” F

LS

is available to the participants. The

functionality captures the resources that are available to the parties as preconditions for the secure

operation of the protocol (e.g., the genesis block will be specified by F

LS

).

Stage 1: Static stake; D = L = R. In the first stage, the trust assumption is static and remains

with the initial set of stakeholders; the execution is not further divided into epochs. There is an

initial stake distribution which is hardcoded into the genesis block that includes the public-keys of

the stakeholders, {(vk

i

, s

i

)}

n

i=1

. Based on our restrictions on the environment, adversarial stake of

no more than 1/2 − is assumed among those initial stakeholders. Specifically, the environment ini-

tially will allow the corruption of a number of stakeholders whose relative stake represents 1/2 − .

The environment allows party corruption by providing tokens of the form (Corrupt, U ) to the adver-

sary; note that due to the corruption delay imposed in this first stage any further corruptions will

be against parties that have no stake initially and hence the corruption model is effectively “static.”

F

LS

will subsequently sample ρ which will seed a “weighted by stake” stakeholder sampling and in

this way lead to the election of a subset of m keys vk

i

1

, . . . , vk

i

R

to form the committee that will

possess honest majority with overwhelming probability in R, (this uses the fact that the relative

stake possessed by malicious parties is 1/2 − ; a linear dependency of R to

−2

will be imposed at

this stage to guarantee applicability of strong concentration bounds). In more detail, the committee

will be selected implicitly by appointing a stakeholder with probability proportional to its stake to

each one of the L = R slots. Subsequently, stakeholders will issue blocks following the schedule that

9

is determined by the slot assignment. The longest chain rule will be applied and it will be possible

for the adversary to fork the blockchain views of the honest parties. Nevertheless, we will prove

with a novel combinatorial argument that the probability of a k-common prefix violation drops

exponentially in

√

k, cf. Theorem 4.24. Similar reasoning will yield favorable guarantees of chain

growth and chain quality. An even more favorable analysis can be made against covert adversaries,

i.e., adversaries that prefer to remain “under the radar” (cf. Section 6).

Stage 2: Dynamic stake with a beacon, adversarial look-ahead E, epoch period of

R slots, and delay D ≈ 2R L. The central idea for the extension of the lifetime of the

above protocol is to consider the sequential composition of several invocations. We first describe a

protocol which relies on a trusted beacon that emits a uniformly random string at regular intervals.

Specifically, the beacon, during slots {(j − 1) · R + 1, . . . , jR}, reveals the j-th random string that

seeds the leader election function for the following epoch. To simplify the relationship between

this stage and the next, which adopts a secure multiparty computation for randomness generation,

we expose the randomness beacon to the adversary E slots prior to exposure to honest parties;

this is the “adversarial look-ahead” parameter. The critical difference compared to the static stake

protocol is that the stake distribution is allowed to change and is drawn from the blockchain itself.

This means that the stake distribution adopted during the j-th epoch (with j ≥ 2) is determined

by the most recent block with time stamp less than (j − 1) · R − k

′

for an appropriate parameter

k

′

. (The generic parameter k

′

appearing here is given a precise formula in the description of the

protocol.)

Regarding the evolving stake distribution, transactions will be continuously generated and trans-

ferred between stakeholders via the environment; players will incorporate posted transactions in the

blockchain-based ledgers that they maintain. In order to accommodate the new accounts that are

being created, the F

LS

functionality enables a new (vk, sk) to be created on demand and assigned to

a new party U

i

. Specifically, the environment can create new parties who will interact with F

LS

for

their public/secret-key in this way treating it as a trusted component that maintains the secret of

their wallet. Note that the adversary can interfere with the creation of a new party, corrupt it, and

supply its own (adversarially created) public-key instead. As before, the environment may request

transactions between stakeholder accounts and can also generate transactions in collaboration with

the adversary on behalf of the corrupted accounts. Recall that our assumption is that at any slot,

in the view of any honest player, the stakeholder distribution places (1/2 + + σ) stake majority

with the honest players (note that different honest players might perceive a different stakeholder

distribution in a certain slot). Furthermore, the stake distribution is guaranteed to shift by at most

σ statistical distance over a certain number of slots—this permits honest players to obtain reliable

estimates on past stake distributions by examining portions of their blockchains which may have

adversarial blocks. The security proof can be seen as an induction in the number of epochs L/R

with the base case supplied by the proof of the static stake protocol. In the end we will argue that

in this setting, a (1/2 − − σ) bound in adversarial stake is sufficient for security of a single draw

(and observe that the size of committee, m, now should be selected to overcome also an additive

term of size ln(L/R) given that the lifetime of the system includes such a number of successive

epochs). The corruption delay is set to D ≈ 2R (in fact, it can be set more precisely as a function

of E and k

′

), thus enabling the adversary to perform adaptive corruptions as long as these are not

instantaneous.

Stage 3: Dynamic stake without a beacon, epoch period of R slots, and delay D ≈ 2R

L. In the third stage, we remove dependence on the beacon by introducing a secure multiparty

10

protocol with “guaranteed output delivery” that effectively simulates it. In this way, we can obtain

the long-livedness of the protocol as described in the stage 2 design but under the weaker assump-

tions of the stage 1 design, i.e., the mere availability of an initial random string and stakeholder

distributions with honest majority. The core idea is the following: given that we guarantee that an

honest majority among elected stakeholders will hold with very high probability, we can further use

this elected set as participants in an instance of a (simple) secure multiparty computation (MPC)

protocol. Naturally, this will require the choice of the length of the epoch to be sufficient so that

it can accommodate a run of the MPC protocol. From a security point of view, the main challenge

is to demonstrate that the MPC suitably simulates a beacon with the relaxation that the output

may become known to the adversary before it is known to the honest parties. A feature of this

stage—from a cryptographic design perspective—is the use of the ledger itself for the simulation of

a reliable broadcast that supports the MPC protocol.

Stage 4: Input endorsers, stakeholder delegates, anonymous communication. In the

final stage of our design, we augment the protocol with two new roles for the entities that are

maintaining the ledger and consider the benefits of anonymous communication. Input-endorsers

create a second layer of transaction endorsing prior to block inclusion. This mechanism enables

the protocol to withstand deviations such as selfish mining and enables us to show that honest

behaviour is an approximate Nash equilibrium under reasonable assumptions regarding the costs

of running the protocol. Note that input-endorsers are assigned to slots in the same way that

slot leaders are, and inputs included in blocks are only acceptable if they are endorsed by an

eligible input-endorser. Second, the delegation feature allows stakeholders to transfer committee

participation to selected delegates that assume the responsibility of the stakeholders in running

the protocol (including participation to the MPC and issuance of blocks). Delegation naturally

gives rise to “stake pools” that can act in the same way as mining pools in bitcoin. Finally, we

observe that by including an anonymous communication layer we can remove the corruption delay

requirement that is imposed in our analysis. This is done at the expense of increasing the online

time requirements for the honest parties.

4

4 Analysis of the Static Stake Protocol

4.1 Basic Concepts and Protocol Description

We begin by describing the blockchain protocol π

SPoS

in the “static stake” setting, where leaders

are assigned to blockchain slots with probability proportional to their (fixed) initial stake which

will be the effective stake distribution throughout the execution. To simplify our presentation, we

abstract this leader selection process, treating it simply as an “ideal functionality” that faithfully

carries out the process of randomly assigning stakeholders to slots. In the following section, we

explain how to instantiate this functionality with a specific secure computation.

We remark that—even with an ideal leader assignment process—analyzing the standard “longest

chain” preference rule in our PoS setting appears to require significant new ideas. The challenge

arises because large collections of slots (epochs, as described above) are assigned to stakeholders at

once; while this has favorable properties from an efficiency (and incentive) perspective, it furnishes

the adversary a novel means of attack. Specifically, an adversary in control of a certain population

of stakeholders can, at the beginning of an epoch, choose when standard “chain update” broadcast

messages are delivered to honest parties with full knowledge of future assignments of slots to

4

In follow-up work we show how the same can be achieved efficiently, see [22].

11

stakeholders. Additionally, an adversary in control of a particular slot may freely generate multiple

blocks associated with the slot, perhaps committing to distinct prior blockchains, and strategically

advertise them to honest players. In contrast, adversaries in typical PoW settings are constrained

to make decisions in an online fashion and cannot freely generate multiple blocks. We remark that

this can have a dramatic effect on the ability of an adversary to produce alternate chains; see the

discussion on “forkable strings” below for detailed discussion.

In the static stake case, we assume that a fixed collection of n stakeholders U

1

, . . . , U

n

interact

throughout the protocol. Stakeholder U

i

possesses s

i

stake before the protocol starts. For each

stakeholder U

i

a verification and signing key pair (vk

i

, sk

i

) for a prescribed signature scheme is

generated; we assume without loss of generality that the verification keys vk

1

, . . . are known by all

stakeholders. Before describing the protocol, we establish basic definitions following the notation

of [28].

Definition 4.1 (Genesis Block). The genesis block B

0

contains the list of stakeholders identified

by their public-keys, their respective stakes (vk

1

, s

1

), . . . , (vk

n

, s

n

) and auxiliary information ρ.

With foresight we note that the auxiliary information ρ will be used to seed the slot leader

election process.

Definition 4.2 (State). A state is a string st ∈ {0, 1}

λ

.

Definition 4.3 (Block). A block B generated at a slot sl ∈ {1, . . . , R} contains the current state

st ∈ {0, 1}

λ

, data d ∈ {0, 1}

∗

, the slot number sl and a signature σ = Sign

sk

(st, d, sl) computed

under sk corresponding to the stakeholder U generating the block at that slot.

Definition 4.4 (Blockchain). A blockchain (or simply chain) relative to the genesis block B

0

is a

sequence of blocks B

1

, . . . , 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 len(C) = n is its number of blocks. The block B

n

is the head of the chain, denoted

head(C). We treat the empty string ε as a legal chain and by convention set head(ε) = ε.

Let C be a chain of length n and k be any non-negative integer. We denote by C

dk

the chain

resulting from removal of the k rightmost blocks of C. If k ≥ len(C) we define C

dk

= ε. We let

C

1

C

2

indicate that the chain C

1

is a prefix of the chain C

2

. We let C[k] = B

k

and adopt interval

notation to reflect contiguous portions of a chain: specifically, C[k : `] = B

k

, . . . , B

`

and parentheses

are used to indicate removal of endpoint so that, for example, C[k : `) = B

k

, . . . , B

`−1

.

Definition 4.5 (Epoch). An epoch is a set of R adjacent slots S = {1, . . . , R}.

(The relevant value R is a parameter of the protocol we analyze in this section.)

Definition 4.6 (Adversarial Stake Ratio). Let U

A

be the set of stakeholders controlled by an

adversary A. Then the adversarial stake ratio is defined as

α =

∑

j∈U

A

s

j

∑

n

i=1

s

i

,

where n is the total number of stakeholders and s

i

is stakeholder U

i

’s stake.

Reflecting corruption delay. In general, we consider adversaries subject to a corruption delay,

which imparts a delay between the time when the adversary selects a party for corruption and the

time when the party actually passes into adversarial control. With such an adversary, parties which

12

have been identified for corruption but are not yet under adversarial control are counted in the

adversarial stake ratio.

Unambiguous stake distributions. In principle, it is possible for various honest parties to disagree

on the current (or a previous) stake distribution. We avoid these ambiguities by explicitly identifying

a stake distribution when considering adversarial stake ratio.

Slot Leader Selection. In the protocol described in this section, for each 0 < j ≤ R, a slot

leader E

j

is determined who has the (sole) right to generate a block at sl

j

. Specifically, for each

slot a stakeholder U

i

is selected as the slot leader with probability p

i

proportional to its stake

registered in the genesis block B

0

; these assignments are independent between slots. In this static

stake case, the genesis block as well as the procedure for selecting slot leaders are determined by an

ideal functionality F

LS

, defined in Figure 1. This functionality is parameterized by the set of initial

stakeholders {(U

1

, s

1

), . . . , (U

n

, s

n

)} assigning to each stakeholder its respective stake, a distribution

D that provides auxiliary information ρ and a leader selection function F defined below.

Definition 4.7 (Leader Selection Process). A leader selection process with respect to stakeholder

distribution S = {(vk

1

, s

1

), . . . , (vk

n

, s

n

)}, (D, F) is a pair consisting of a distribution and a deter-

ministic function such that, when ρ ← D it holds that for all sl

j

∈ {sl

1

, . . . , sl

R

}, F(S, ρ, sl

j

) outputs

U

i

∈ {U

1

, . . . , U

n

} with probability

p

i

=

s

i

∑

n

k=1

s

k

where s

i

is the stake held by stakeholder U

i

(we call this “weighting by stake”); furthermore the

family of random variables {F(S, ρ, sl

j

)}

R

j=1

are independent.

We note that sampling proportional to stake can be implemented in a straightforward manner.

For instance, a simple process operates as follows. Let ˜p

i

= s

i

/

∑

n

j=i

s

j

. For each i = 1, . . . , n − 1,

provided that no stakeholder has yet been selected, the process flips a ˜p

i

-biased coin; if the result

of the coin is 1, the party U

i

is selected for the slot and the process is complete. (Note that ˜p

n

= 1,

so the process is certain to complete with a unique leader.) When we implement this process as

a function F (·), sufficient randomness must be allocated to simulate the biased coin flips. If we

implement the above with λ precision for each individual coin flip, then selecting a stakeholder will

require ndlog λe random bits in total. Note that using a pseudorandom number generator (PRG)

one may use a shorter “seed” string and then stretch it using the PRG to the appropriate length.

A Protocol in the F

LS

[mode]-hybrid model. We start by describing a simple PoS based

blockchain protocol considering static stake in the F

LS

[SIG]-hybrid model, i.e., where the genesis

block B

0

(and consequently the slot leaders) are determined by the ideal functionality F

LS

[SIG].

F

LS

[SIG] provides the stakeholders with a genesis block containing a stake distribution indexed by

signature verification keys generated by a EUF-CMA signature scheme, while F

LS

[F

DSIG

] obtains

such keys from a signature ideal functionality F

DSIG

. This subtle difference comes into play when

describing an ideal version of π

SPoS

used in an intermediate hybrid argument of the security proof,

which will be discussed in Section 4.2. The stakeholders U

1

, . . . , U

n

interact among themselves and

with F

LS

through Protocol π

SPoS

described in Figure 2.

We are interested in applications where transactions are inserted in the ledger. In our analysis,

we will consider simple coin transfer transactions of the format “stakeholder vk

1

transfers to stake-

holders vk

2

an amount of x coins.” A transaction will consist of a transaction template tx of this

format accompanied by a signature of tx under the signing key corresponding to vk

1

. We define a

valid transaction as follows:

13

Functionality F

LS

[mode]

F

LS

[mode] incorporates the diffuse and key/transaction functionality F

D+KT

from Section 2 and is param-

eterized by the respective stakes of the initial stakeholders S

0

= {(U

1

, s

1

), . . . , (U

n

, s

n

)}, a distribution D

and a function F so that (D, F) is a leader selection process. In addition, F

LS

[mode] is parameterized by

mode, which determines how signature verification keys are generated. When F

LS

[mode] is instantiated

with mode = SIG (resp. mode = F

DSIG

) it is denoted F

LS

[SIG] (resp. F

LS

[F

DSIG

]). F

LS

interacts with

stakeholders as follows:

• Signature Key Pair Generation: F

LS

[SIG] generates signing and verification keys sk

i

, vk

i

for

stakeholder U

i

by executing KG(1

κ

) for i = 1, . . . , n. F

LS

[F

DSIG

] generates (sk

i

, vk

i

) by querying

F

DSIG

(Figure 3) with (KeyGen, sid

i

) on behalf of U

i

(with a unique session identifier sid

i

related

to U

i

) and setting (sk

i

= sid

i

, vk

i

= v

i

) (received from F

DSIG

as response) for i = 1, . . . , n. In

either case, F

LS

[mode] sets S

′

0

= {(vk

1

, s

1

), . . . , (vk

n

, s

n

)}.

• Genesis Block Generation Upon receiving (genblock req

, U

i

) from stakeholder U

i

, F

LS

proceeds

as follows. If ρ has not been set, F

LS

samples ρ ← D. In any case, F

LS

sends (genblock, S

′

0

, ρ, F)

to U

i

.

• Signatures and Verification. For signing and verification requests on behalf of user U

i

,

F

LS

[F

DSIG

] provides access to the corresponding F

DSIG

interface, while F

LS

[SIG] utilizes (sk

i

, vk

i

)

to respond restricting access of signing to the respective secret-key owners.

Figure 1: Functionality F

LS

[mode].

Definition 4.8 (Valid Transaction). A pair (tx, σ) is considered a valid transaction by a verifier

V if the following holds:

• The transaction template tx is of the format “stakeholder vk

1

transfers to stakeholder vk

2

an

amount of x coins with transaction serial number sn” where vk

1

is a verification key contained

in the current stake distribution S and x ∈ Z.

• Vrf

vk

1

(σ, tx) = 1.

• The ledger cannot contain two transactions issued from the same stakeholder with the same

serial number sn; thus a transaction is only valid with respect to a blockchain if no previous

transaction (from the same stakeholder) has the same serial number.

For simplicity we assume that all properly signed transactions are valid and are included in

the ledger; in particular this means that there is a way to parse the ledger and disambiguate any

recorded double/overspents (e.g., following the order that is imposed by the ledger).

Given Definitions 4.4 and 4.8, we define a valid chain as a blockchain (according to Defini-

tion 4.4) where all transactions contained in every block are valid (according to Definition 4.8).

The protocol relies on a maxvalid

S

(C, C) function that chooses a chain given the current chain C

and a set of valid chains C that are available in the network. In the static case we analyze the

simple “longest chain” rule. (In the dynamic case the rule is parameterized by a common chain

length; see Section 5.)

Function maxvalid(C, C): Returns the longest chain from C ∪ {C}. Ties are broken in

favor of C, if it has maximum length, or arbitrarily otherwise.

14

Protocol π

SPoS

π

SPoS

is a protocol run by stakeholders U

1

, . . . , U

n

interacting with F

LS

[SIG] over a sequence of slots

S = {1, . . . , R}. π

SPoS

proceeds as follows:

1. Initialization Stakeholder U

i

∈ {U

1

, . . . , U

n

}, receives from the key registration interface its

public and secret key. Then it receives the current slot from the diffuse interface and in case it

is sl

1

it sends (genblock req

, U

i

) to F

LS

[SIG], receiving (genblock, S

0

, ρ, F) as answer. U

i

sets the

local blockchain C = B

0

= (S

0

, ρ) and the initial internal state st = H(B

0

). Otherwise, it receives

from the key registration interface the initial chain C, sets the local blockchain to C and the initial

internal state st = H(head(C)).

2. Chain Extension For every slot sl

j

∈ S, every stakeholder U

i

performs the following steps:

(a) U

i

receives from the environment the transaction data d ∈ {0, 1}

∗

to be inserted into the

blockchain.

(b) Collect all valid chains received via broadcast into a set C, verifying that for every chain C

′

∈

C and every block B

′

= (st

′

, d

′

, sl

′

, σ

′

) ∈ C

′

it holds that Vrf

vk

′

(σ

′

, (st

′

, d

′

, sl

′

)) = 1, where vk

′

is the verification key of the stakeholder U

′

= F(S

0

, ρ, sl

′

). U

i

computes C

′

= maxvalid(C, C),

sets C

′

as the new local chain and sets state st = H(head(C

′

)).

(c) If U

i

is the slot leader determined by F(S

0

, ρ, sl

j

), it generates a new block B = (st, d, sl

j

, σ)

where st is its current state, d ∈ {0, 1}

∗

is the transaction data and σ = Sign

sk

i

(st, d, sl

j

) is a

signature on (st, d, sl

j

). U

i

computes C

′

= C|B, broadcasts C

′

, sets C

′

as the new local chain

and sets state st = H(head(C

′

)).

3. Transaction generation Upon receiving a transaction template tx from the environment, U

i

computes σ = Sign

sk

i

(tx) provided that tx is consistent with the state of the ledger in the view of

U

i

) and sends (tx, σ) to the environment.

Figure 2: Protocol π

SPoS

.

4.2 Transition to the Ideal Protocol

As a first step of the security analysis of π

SPoS

, we will introduce an idealized protocol π

iSPoS

and

present an intermediate hybrid argument that shows that it is computationally indistinguishable

from π

SPoS

. Instead of relying on F

LS

[SIG] and an EUF-CMA signature scheme, π

iSPoS

operates

with an ideal signature scheme. To that end, π

iSPoS

interacts with F

LS

[F

DSIG

] for obtaining signing

and verification keys for the ideal signature scheme employed in the protocol. In the next sec-

tions, we will prove that π

iSPoS

is secure through a series of combinatorial arguments. This hybrid

approach insulates these combinatorial arguments from the specific details of the underlying sig-

nature schemes used to instantiate π

SPoS

and the biases that these schemes might introduce in the

distributions of π

SPoS

.

First, in Figure 3, we present Functionality F

DSIG

as defined in [16], where it is also shown that

EUF-CMA signature schemes realize F

DSIG

. Notice that this fact will be used to show that our

idealized protocol can actually be realized based on practical digital signature schemes (such as

DSA and ECDSA) and ultimately that π

iSPoS

is indistinguishable from π

SPoS

.

The idealized protocol π

iSPoS

is run by the stakeholders interacting with F

LS

[F

DSIG

] and F

DSIG

.

Basically, π

iSPoS

behaves exactly as π

SPoS

except for calls to Vrf

vk

(σ) and Sign

sk

(m). Namely, instead

of locally computing Sign

sk

i

(m), U

i

sends (Sign, sid, m) to F

DSIG

, receiving (Signature, sid, m, σ)

and outputting σ as the signature. Moreover, instead of locally computing Vrf

vk

′

(σ, m), U

i

sends

(Verify, sid

i

, m, σ, v

′

) to F

DSIG

(where v

′

corresponds to verification key vk

′

), outputting the value

f received in message (Verified, sid

i

, m, f ). Protocol π

iSPoS

is described in Figure 4. This idealized

description will be further developed when arguing about the dynamic stake case, where additional

15

Functionality F

DSIG

F

DSIG

interacts with stakeholders as follows:

• Key Generation Upon receiving a message (KeyGen, sid) from a stakeholder U

i

, verify that sid =

(U

i

, sid

′

) for some sid

′

. If not, then ignore the request. Else, hand (KeyGen, sid) to the adversary.

Upon receiving (VerificationKey, sid, v) from the adversary, output (VerificationKey, sid, v) to U

i

,

and record the pair (U

i

, v).

• Signature Generation Upon receiving a message (Sign, sid, m) from U

i

, verify that sid =

(U

i

, sid

′

) for some sid

′

. If not, then ignore the request. Else, send (Sign, sid, m) to the ad-

versary. Upon receiving (Signature, sid, m, σ) from the adversary, verify that no entry (m, σ, v, 0)

is recorded. If it is, then output an error message to U

i

and halt. Else, output (Signature, sid, m, σ)

to U

i

, and record the entry (m, σ, v, 1).

• Signature Verification Upon receiving a message (Verify, sid, m, σ, v

′

) from some stakeholder

U

i

′

, hand (Verify, sid, m, σ, v

′

) to the adversary. Upon receiving (Verified, sid, m, φ) from the ad-

versary do:

1. If v

′

= v and the entry (m, σ, 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, σ

′

, v, 1) for any σ

′

is recorded,

then set f = 0 and record the entry (m, σ, 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, σ, v

′

, f

′

) recorded, then let f = f

′

. (This condition guaran-

tees consistency: All verification requests with identical parameters will result in the same

answer.)

4. Else, let f = φ and record the entry (m, σ, v

′

, φ).

Output (Verified, sid, m, f ) to U

i

′

.

Figure 3: Functionality F

DSIG

.

building blocks must be considered in the idealized protocol.

The following proposition is an immediate corollary of the results in [16] showing that EUF-CMA

signature schemes realize F

DSIG

.

Proposition 4.9. For each PPT A, Z it holds that there is a PPT S so that

EXEC

F

LS

[SIG]

π

SPoS

,A,Z

(λ) and EXEC

F

LS

[F

DSIG

]

π

iSPoS

,S,Z

(λ)

are computationally indistinguishable.

In light of the above proposition in the remaining of the analysis we will focus on the properties

of the protocol π

iSPoS

(note that this implication does not apply to any

5

possible property one

might consider in an execution for π

iSPoS

; nevertheless the properties we will prove for π

iSPoS

are all verifiable by the environment Z and as a result they can be inherited by π

SPoS

due to

proposition 4.9).

4.3 The Fork Abstraction

In our security arguments we routinely use elements of {0, 1}

n

to indicate which slots—among

a particular window of slots of length n—have been assigned to adversarial stakeholders. When

5

An example of such a property would be a property testing a non-trivial fact about the parties’ private states.

16

Protocol π

iSPoS

π

iSPoS

is a protocol run by stakeholders U

1

, . . . , U

n

interacting with F

LS

[F

DSIG

] over a sequence of slots

S = {1, . . . , R}. π

iSPoS

proceeds as follows:

1. Initialization Stakeholder U

i

∈ {U

1

, . . . , U

n

}, receives from the key registration interface its

public and secret key. Then it receives the current slot from the diffuse interface and in case it

is sl

1

it sends (genblock req

, U

i

) to F

LS

[F

DSIG

], receiving (genblock, S

0

, ρ, F) as answer. U

i

sets the

local blockchain C = B

0

= (S

0

, ρ) and the initial internal state st = H(B

0

). Otherwise, it receives

from the key registration interface the initial chain C, sets the local blockchain to C and the initial

internal state st = H(head(C)).

2. Chain Extension For every slot sl

j

∈ S, every stakeholder U

i

performs the following steps:

(a) Collect all valid chains received via broadcast into a set C, verifying that for every

chain C

′

∈ C and every block B

′

= (st

′

, d

′

, sl

′

, σ

′

) ∈ C

′

it holds that F

DSIG

answers

with (Verified, sid, (st

′

, d

′

, sl

′

), 1) upon being queried with (Verify, sid, (st

′

, d

′

, sl

′

), σ

′

, vk

′

),

where vk

′

is the verification key of the stakeholder U

′

= F(S

0

, ρ, sl

′

). U

i

computes

C

′

= maxvalid(C, C), sets C

′

as the new local chain and sets state st = H(head(C

′

)).

(b) If U

i

is the slot leader determined by F(S

0

, ρ, sl

j

), it generates a new block B = (st, d, sl

j

, σ)

where st is its current state, d ∈ {0, 1}

∗

is the transaction data and σ is obtained from

F

DSIG

’s answer (Signature, sid, (st, d, sl

j

), σ) upon being queried with (Sign, sid

i

, (st, d, sl

j

)).

U

i

computes C

′

= C|B, broadcasts C

′

, sets C

′

as the new local chain and sets state st =

H(head(C

′

)).

3. Transaction generation Given a transaction template tx, U

i

returns σ obtained from F

DSIG

’s

answer (Signature, sid

i

, tx, σ) upon being queried with (Sign, sid

i

, tx), provided that tx is consistent

with the state of the ledger in the view of U

i

.

Figure 4: Protocol π

iSPoS

.

strings have this interpretation we refer to them as characteristic strings.

Definition 4.10 (Characteristic String). Fix an execution E with genesis block B

0

, adversary A,

and environment Z. Let S = {i + 1, . . . , i + n} denote a sequence of slots of length |S| = n. The

characteristic string w ∈ {0, 1}

n

of S is defined so that w

k

= 1 if and only if the adversary controls

the slot leader of slot i + k. For such a characteristic string w ∈ {0, 1}

∗

we say that the index i is

adversarial if w

i

= 1 and honest otherwise.

We start with some intuition for our approach to analyze the protocol. Let w ∈ {0, 1}

n

be

a characteristic string for a sequence of slots S. Among the fundamental properties we wish to

ensure of our protocol (CP, ∃CQ, and HCG), common prefix (CP) will require the most technical

effort. In this section, we develop a graph-theoretic abstraction to facilitate reasoning about these

properties, principally motivated by the task of establishing CP.

To motivate this, consider two observers that (i.) go offline immediately prior to the com-

mencement of a sequence of slots S, (ii.) have adopted the same current chain C

0

prior to the

commencement of S, and (iii.) come back online at the last slot of S and request an update of their

chain. A fundamental concern in our analysis is the possibility that such observers can be presented

with a “diverging” view over the sequence S: specifically, the possibility that the adversary can

force the two observers to adopt two different chains C

1

, C

2

whose common prefix is exactly C

0

. We

observe that not all characteristic strings permit this. For instance the (entirely honest) string 0

n

ensures that the two observers will adopt the same chain C which will consist of n new blocks on top

of the common prefix C

0

. On the other hand, other strings do not guarantee such common extension

of C

0

; in the case of 1

n

, it is possible for the adversary to produce two completely different histories

17

during the sequence of slots S and thus furnish to the two observers two distinct chains C

1

, C

2

that

only share the common prefix C

0

. The bulk of the proof that the Ouroboros protocol achieves CP

relies on the fact that characteristic strings permitting such “forkings” are quite rare—indeed, we

show that they have density 2

−Ω(

√

n)

so long as the fraction of adversarial slots is 1/2 − .

To reason about the protocol at a more abstract level, we define below a formal notion of

“fork” that captures the relationship between the chains adopted by honest slot leaders during an

execution of the protocol π

iSPoS

. In preparation for the definition, we recall that honest players

always choose to extend a maximum length chain among those available to the player on the

network. Furthermore, if such a maximal chain C includes a block B previously broadcast by an

honest player, the prefix of C prior to B must entirely agree with the chain (terminating at B)

broadcast by this previous honest player. This “confluence” property follows immediately from the

fact that the state of any honest block effectively commits to a unique chain beginning at the genesis

block. To conclude, any chain C diffused by an honest player must begin with a chain produced

by a previously honest player (or, alternatively, the genesis block), continue with a possibly empty

sequence of adversarial blocks and, finally, terminate with an honest block. It follows that the

chains broadcast by honest players form a natural directed tree. The fact that honest players

reliably broadcast their chains and always build on the longest available chain introduces a second

important property of this tree: the “depths” of the various honest blocks added by honest players

during the protocol must all be distinct.

Of course, the actual chains induced by an execution of π

iSPoS

are comprised of blocks containing

a variety of data that are immaterial for reasoning about forking or the other elementary chain

properties. For this reason the formal notion of fork below merely reflects the directed tree formed

by the relevant chains and the identities of the players—expressed as indices in the string w—

responsible for generating the blocks in these chains.

Forks and forkable strings. We define, below, the basic combinatorial structures we use to

reason about the possible views observed by honest players during a protocol execution with this

characteristic string.

w =

0

1

1

2

2

0

3

1

4

4

0

5

0

6

1 1

8

0

90

ˆ

t

t

Figure 5: A fork F for the string w = 010100110; vertices appear with their labels and honest

vertices are highlighted with double borders. Note that the depths of the (honest) vertices associated

with the honest indices of w are strictly increasing. Two tines are distinguished in the figure: one,

labeled ˆt, terminates at the vertex labeled 9 and is the longest tine in the fork; a second tine t

terminates at the vertex labeled 3. The quantity gap(t) indicates the difference in length between t

and ˆt; in this case gap(t) = 4. The quantity reserve(t) = |{i | `(v) < i ≤ |w| and w

i

= 1}| indicates

the number of adversarial indices appearing after the label of the last honest vertex v of the tine;

in this case reserve(t) = 3. As each leaf of F is honest, F is closed.

18

Definition 4.11 (Fork). Let w ∈ {0, 1}

n

and let H = {i | w

i

= 0} denote the set of honest indices.

A fork for the string w is a directed, rooted tree F = (V, E) with a labeling ` : V → {0, 1, . . . , n} so

that

• each edge of F is directed away from the root;

• the root r ∈ V is given the label `(r) = 0;

• the labels along any directed path in the tree are strictly increasing;

• each honest index i ∈ H is the label of exactly one vertex of F ;

• the function d : H → {1, . . . , n}, defined so that d(i) is the depth in F of the unique vertex v

for which `(v) = i, is strictly increasing. (Specifically, if i, j ∈ H and i < j, then d(i) < d(j).)

As a matter of notation, we write F ` w to indicate that F is a fork for the string w. We say that

a fork is trivial if it contains a single vertex, the root.

Definition 4.12 (Tines, depth, and height). A path in a fork F originating at the root is called a

tine. For a tine t we let length(t) denote its length, equal to the number of edges on the path. For

a vertex v, we let depth(v) denote the length of the (unique) tine terminating at v. The height of

a fork (as usual for a tree) is defined to be the length of the longest tine.

We overload the notation `() so that it applies to tines, by defining `(t) , `(v), where v is

the terminal vertex on the tine t. We borrow the “truncation operator,” described earlier in the

paper for chains: for a tine t we let t

dk

denote the tine obtained by removing the last k edges; if

length(t) ≤ k, we define t

dk

to consist solely of the root.

If a vertex v of a fork is labeled with an adversarial index (i.e., w

`(v)

= 1) we say that the vertex

is adversarial; otherwise, we say that the vertex is honest. For convenience, we declare the root

vertex to be honest. We extend this terminology to tines: a tine is honest if it terminates with an

honest vertex and adversarial otherwise. By this convention the empty tine t

is honest.

The fork of honestly constructed chains. As discussed above, with an execution E of π

iSPoS

one can naturally associate a characteristic string w and a fork F

D

` w corresponding to the chains

constructed and diffused by honest participants during the protocol. See Figure 5 for an example,

which also demonstrates some of the quantities defined above and in the remainder of this section.

The fork shown in the figure reflects an execution in which (i.) the honest player associated with

the first slot builds directly on the genesis block (as it must), (ii.) the honest player associated with

the third slot is shown a chain of length 1 produced by the adversarial player of slot 2 (in addition

to the honestly generated chain of step (i.)), which it elects to extend, (iii.) the honest player

associated with slot 5 is shown a chain of length 2 building on the chain of step (i.) augmented

with a further adversarial block produced by the player of slot 4, etc.

We remark that the tight correspondence described above between forks and executions requires

that a slot is marked as adversarial if the owner of that slot ever fell under adversarial control

during the protocol; this is one direct indication of the challenge of “long-range attacks” which

involve corruption of slot leaders long after they have been awarded leadership. In our long-lived

protocols, such attacks are mitigated by a bounded-depth longest-chain rule which permits analysis

of forks whose characteristic strings are determined by the corruption schedule of an adversary over

a bounded period of time.

We begin by defining a natural notion of inclusion for two forks:

19

Definition 4.13 (Fork prefixes). If w is a prefix of the string w

′

∈ {0, 1}

∗

, F ` w, and F

′

` w

′

,

we say that F is a prefix of F

′

, written F v F

′

, if F is a consistently-labeled subgraph of F

′

.

Specifically, every vertex and edge of F appears in F

′

and, furthermore, the labels given to any

vertex appearing in both F and F

′

are identical.

If F v F

′

, each tine of F appears as the prefix of a tine in F

′

. In particular, the labels appearing

on any tine terminating at a common vertex are identical and, moreover, the depth of any honest

vertex appearing in both F and F

′

is identical.

In many cases, it is convenient to work with forks that do not “commit” anything beyond final

honest indices.

Definition 4.14 (Closed forks). A fork is closed if each leaf is honest. By convention the trivial

fork, consisting solely of a root vertex, is closed.

Note that the fork F

D

discussed above of corresponding to honestly created chains is closed:

Every chain constructed by an honest player naturally terminates with block broadcast in an honest

slot. We remark that a closed fork has a unique longest tine (as all maximal tines terminate with

an honest vertex, and these must have distinct depths). Note, additionally, that if ˇw is a prefix of

w and F ` w, then there is a unique closed fork

ˇ

F ` ˇw for which

ˇ

F v F . In particular, taking

ˇw = w, we note that for any fork F ` w, there is a unique closed fork

F ` w for which F v F ; in

this case we say that

F is the closure of F .

The fork of adopted chains. Consider now the valid chains adopted by the honest participants

of the protocol. This set clearly includes all chains constructed (and diffused) by honest participants;

on the other hand, it may contain additional valid chains delivered by the adversary to honest

participants. In particular, we may associate a fork F

A

` w with these adopted chains and note

that F

D

v F

A

. The fork F

A

is not, in general, closed. Note, however, that maximal tines in this

fork—as they have been adopted by an honest player according to the longest chain rule—must

have length no less than any chain previously diffused by an honest player. We begin by precisely

defining this notion.

Definition 4.15 (Viability). Let F ` w be a fork for a string w ∈ {0, 1}

n

and let t be a tine of F .

We say that t is viable if, for all honest indices h ≤ `(t), we have

d(h) ≤ length(t) .

(Recall that `(t) is the label of the terminal vertex of t.)

If t is viable, an honest participant (or observer) witnessing the execution at time `(t)—if

provided the tine t along with all honest tines generated up to time `(t)—could conceivably select

t via the maxvalid() rule. Observe that any honest tine is viable: by definition, the depth of the

terminal vertex of an honest tine exceeds that of all prior honest vertices. As remarked above, all

maximal tines appearing in F

A

are viable.

4.3.1 The Abstract Chain Properties

Let w ∈ {0, 1}

n

be a characteristic string. We define the following properties of w which are direct

analogues of the general protocol properties defined above. Given a tine t and a sequence of slots S

such that `(t) ≥ max S, we refer to a “portion of t spanning S” as the maximum subgraph t

′

of t such

that `(v) ∈ S ∧ v ∈ t =⇒ v ∈ t

′

. We use the notation t(S) to denote this subgraph. We continue

20

to use interval notation for sequences of slots: that is, [sl

1

: sl

2

] = {sl

1

, . . . , sl

2

} and parentheses in

place of brackets indicate that the endpoint is left out. Thus [sl

1

: sl

2

) = {sl

1

, . . . , sl

2

− 1}. As a

final matter of notation, we often elide the parentheses in expressions such as t([sl

1

, sl

2

]), simply

writing t[sl

1

, sl

2

]. We will refer to a “portion of t spanning s slots” when the particular sequence of

slots is not specified.

• Common Prefix (cp); with parameter k ∈ N. A characteristic string w possesses k-cp if,

for every fork F ` w and every pair of viable tines t

1

and t

2

of F for which `(t

1

) ≤ `(t

2

), the

tine t

dk

1

is a prefix of t

2

. (Equivalently, length(t

1

) − length(t

1

∩ t

2

) ≤ k, where t

1

∩ t

2

denotes

the common prefix of the two tines.)

• Honest-Bounded Chain Growth (hcg); with parameters τ ∈ (0, 1] and s ∈ N. A

characteristic string w possesses hcg with parameters τ and s if, for every fork F ` w, every

viable tine t of F , and every honest vertex v on t for which `(v)+s ≤ `(t), the path t(`(v), `(t)]

contains at least τ s vertices.

• Chain Growth (cg); with parameters τ ∈ (0, 1] and s ∈ N. A characteristic string

w possesses (τ, s)-cg if, for every fork F ` w and every viable tine t of F , any portion of t

spanning s slots contains at least τ s vertices.

• Existential Chain Quality (∃cq); with parameter s ∈ N. A characteristic string w

possesses s-∃cq if, for every fork F ` w and every viable tine t of F , any portion of t spanning

s slots contains at least one honest vertex.

4.3.2 Probabilistic Preliminaries

Anticipating the proofs in the next few sections, we record a Chernoff–Hoeffding bound and an

elementary stochastic dominance argument.

Theorem 4.16 (Chernoff–Hoeffding bound; see, e.g., [33]). Let X

1

, . . . , X

T

be independent random

variables with X

i

∈ [0, 1]. Let X =

∑

T

i=1

X

i

and μ = E[X]. Then, for all δ ≥ 0,

Pr[X ≥ (1 + δ)μ] ≤ e

−

δ

2

2+δ

μ

and Pr[X ≤ (1 − δ)μ] ≤ e

−

δ

2

2

μ

.

Additionally, for any Λ > 0,

Pr[X ≥ μ + Λ] ≤ e

−2Λ

2

/T

and Pr[X ≤ μ − Λ] ≤ e

−2Λ

2

/T

.

Definition 4.17. For two elements x, y ∈ {0, 1}

n

, we write x ≤ y if, for each i, x

i

≤ y

i

. With this

partial order, we define a notion of monotone subsets of {0, 1}

n

: A subset E ⊆ {0, 1}

n

is monotone

if, for each pair x, y ∈ {0, 1}

n

, x ∈ E and x ≤ y implies that y ∈ E. Let X and Y be two random

variables taking values in {0, 1}

n

. We write X ≺ Y if Pr[X ∈ E] ≤ Pr[Y ∈ E] for any monotone

set E. In this case, we say that Y stochastically dominates X.

Lemma 4.18. Let X = (X

1

, . . . , X

n

) be a family of random variables taking values in {0, 1} with

the property that, for each i > 0, E[X

i

| X

1

, . . . , X

i−1

] ≤ p. Let B = (B

1

, . . . , B

n

) be a family of

independent random variables, taking values in {0, 1}, for which E[B

i

= 1] = p. Then X ≺ B.

Proof. We proceed by induction. The statement is clear for n = 1. In general, consider a random

variable X satisfying the conditions of the theorem and taking values in {0, 1}

n+1

; let E ⊂ {0, 1}

n+1

be a monotone event. We wish to prove that Pr[X ∈ E] ≤ Pr[B ∈ E].

21

We write X = (Y, Z), where Y takes values in {0, 1}

n

and Z in {0, 1}. By induction, we may

assume that Y ≺ (B

1

, . . . , B

n

). Consider the events

E

0

= {(y

1

, . . . , y

n

) | (y

1

, . . . , y

n

, 0) ∈ E} and E

1

= {(y

1

, . . . , y

n

) | (y

1

, . . . , y

n

, 1) ∈ E} ;

observe that the monotonicity of E implies that E

0

⊆ E

1

and that E

0

, E

1

are monotone. To study

Pr[X ∈ E], for an element y = (y

1

, . . . , y

n

) ∈ {0, 1}

n

define

q(y) = Pr[X ∈ E | Y = y] .

Observe that Pr[X ∈ E] = E[q(Y )] and, recalling that E

0

⊂ E

1

, that

y ∈ E

0

⇒ q(y) = 1,

y ∈ E

1

\ E

0

⇒ q(y) ≤ p by assumption, and

y 6 ∈ E

1

⇒ q(y) = 0 .

We conclude that

Pr[X ∈ E] ≤ Pr[Y ∈ E

0

] + p · Pr[Y ∈ E

1

\ E

0

] = p Pr[Y ∈ E

1

] + (1 − p) Pr[Y ∈ E

0

]

≤ p Pr[(B

1

, . . . , B

n

) ∈ E

1

] + (1 − p) Pr[(B

1

, . . . , B

n

) ∈ E

0

] (1)

= Pr[(B

1

, . . . , B

n

) ∈ E

0

] + p Pr[(B

1

, . . . , B

n

) ∈ E

1

\ E

0

] = Pr[B ∈ E] ,

as desired. The inequality of line (1) follows by the induction hypothesis.

4.4 Chain Quality

Lemma 4.19 (Abstract Existential Chain Quality). Consider a characteristic string w = w

1

. . . w

L

drawn in {0, 1}

L

so that each w

i

is independently assigned the value 1 with probability 1/2 − for

some ∈ (0, 1/2). Then, for s > 0,

Pr[w does not satisfy s-∃cq] ≤

∃cq

(s; L, ) ,

(

(

−2

+ 3)/2

)

L exp(−2

2

s) .

Proof. An s-∃cq violation for w consists of a fork F ` w and a viable tine t for which there is a

pair of indices sl

1

and sl

2

so that sl

1

+ s ≤ sl

2

≤ `(t) and t(sl

1

: sl

2

] contains no honest vertices.

For such a violation, let sl

∗

1

denote the largest index of an honest vertex in t[0 : sl

1

]—note that the

root of F is honest by fiat so that sl

∗

1

is well-defined. Similarly, let sl

∗

2

denote the smallest index

for which sl

2

≤ sl

∗

2

≤ `(t) and t[0 : sl

∗

2

] is viable. This is also well defined since t is viable. Observe

that no sl

2

≤ sl

∗

≤ sl

∗

2

can index an honest vertex of t; otherwise, sl

∗

> sl

2

(by assumption) but the

tine t[0 : sl

∗

− 1] would be viable—as it supports extension by an honest vertex—which contradicts

minimality of sl

∗

2

. As we assume that s > 0,

sl

∗

1

≤ sl

1

< sl

1

+ s ≤ sl

2

≤ sl

∗

2

and observe that t(sl

∗

1

, sl

∗

2

] contains no honest vertices. We define the rank of this violation to be

the quantity ` = sl

∗

2

− sl

∗

1

and refer to sl

∗

2

as the target of the violation; we say that the pair (sl

∗

2

; `)

is the signature of such a violation. Note that the rank ` of a s-∃cq violation is always at least s.

Consider now the sequence

sl

∗

1

= hsl

0

< · · · < hsl

g

≤ sl

∗

2

,

22

where hsl

0

, . . . , hsl

g

denote the honest indices in the region [sl

∗

1

, sl

∗

2

], and let t

0

, . . . , t

g

denote the

tines for which `(t

i

) = hsl

i

. Note that t[0 : sl

∗

1

] = t

0

and that length(t

i−1

) < length(t

i

) for each

0 < i ≤ g. As t[0 : sl

∗

2

] is viable and hsl

g

is honest, we note that length(t

g

) ≤ length(t[0 : sl

∗

2

]). To

conclude,

length(t[0 : sl

∗

2

]) ≥ length(t[0 : sl

∗

1

]) + g .

We remark that g = #

0

( ˆw), where ˆw = w

sl

∗

1

+1

, . . . , w

sl

∗

2

, and note that ` = | ˆw|. On the other hand,

as t(sl

∗

1

, sl

∗

2

] contains no honest vertex we have the immediate upper bound

length(t[0 : sl

∗

2

]) ≤ length(t[0 : sl

∗

1

]) + #

1

( ˆw) ,

where #

1

( ˆw) denotes the number of adversarial indices in ˆw. Thus #

1

( ˆw) ≥ #

0

( ˆw).

Fixing a particular signature (sl, `) it follows that

Pr[w admits an s-∃CQ violation with signature (sl; `)]

≤ Pr

w

[#

0

( ˆw) − #

1

( ˆw) ≤ 0]

= Pr

w

[#

1

( ˆw) ≥ `/2]

≤ Pr

w

[#

1

( ˆw) ≥ `(1/2 − ) + `] ≤ exp(−2

2

`)

where ˆw, as above, denotes w

sl−`+1

, . . . , w

sl

. It follows that for any sl ≤ L

Pr[w admits an s-∃CQ violation with signature (sl; `) for any ` ≥ s]

≤

∞

∑

`=s

exp(−2

2

`) ≤

∫

∞

s−1

exp(−2

2

`) d` =

1

2

2

exp(−2

2

(s − 1))

=

exp(2

2

)

2

2

exp(−2

2

s) .

The union bound, applied over all indices, yields

Pr[w admits an s-∃CQ violation over y] ≤

exp(2

2

)

2

2

L exp(−2

2

s) ≤

[

(

−2

+ 3)/2

]

L exp(−2

2

s) ,

where the final inequality follows from exp(x) ≤ 1 + (3/2)x for x ∈ (0, 1/2).

4.5 Chain Growth

We begin with the lemma below demonstrating that chain growth (cg) follows from existential

chain quality (∃cq) and honest-bounded chain growth (hcg).

Lemma 4.20. Consider a characteristic string w that satisfies ∃cq with parameter s

∃cq

and hcg

with parameters τ

hcg

and s

hcg

; then it satisfies cg with parameters

s = 2s

∃cq

+ s

hcg

and τ = τ

hcg

·

(

s

hcg

s

hcg

+ 2s

∃cq

)

.

In particular, assuming s

hcg

≥ 2s

∃cq

, the execution satisfies cg with parameter τ ≥ τ

hcg

/2.

Proof. Let F ` w be a fork and let t be a viable tine. Consider a portion of t spanning ˆs ≥ s =

2s

∃cq

+ s

hcg

slots. By ∃cq, there must be an honest vertex of t associated with the first s

∃cq

and

23

last s

∃cq

slots. As these two honest blocks are separated by at least s

hcg

slots, applying hcg to the

tine that terminates at the later honest block (which is necessarily viable) guarantees that at least

τ

hcg

· (ˆs − 2s

∃CQ

) = τ

hcg

·

(

ˆs − 2s

∃cq

ˆs

)

︸ ︷︷ ︸

(†)

ˆs ≥ τ

hcg

·

(

s

hcg

s

hcg

+ 2s

∃cq

)

ˆs

vertices appear in the region. (The last inequality follows because the function f

λ

(x) = (x − λ)/x,

for any λ > 0, is strictly increasing for x > 0—thus (†) is minimized when ˆs = s

HCG

+ 2s

∃CQ

.) The

statement of the lemma follows.

We now establish concrete bounds on hcg; coupled with the conclusions about ∃cq from Sec-

tion 4.4, this will yield our desired bounds on cg.

Lemma 4.21 (Abstract Honest Chain Growth). Consider w = w

1

. . . w

L

drawn in {0, 1}

L

so that

each w

i

independently takes the value 1 with probability 1/2 − . Then for any s > 0 and δ > 0,

Pr[w admits a (τ, s)-hcg violation] ≤

hcg

(τ, s; L, ) ,

(

(δ

−2

+ 3)/2

)

L exp(−2δ

2

s) ,

where τ = 1/2 + − δ. In particular, taking δ = , we remark that

Pr[w admits a (1/2, s)-hcg violation] ≤ (

2

/2 + 2)L exp(−2

2

s) .

Proof. In the event that w admits a (τ, s)-hcg violation there is a fork F ` w, a viable tine t of

F , and two indices sl

1

and sl

2

for which (i.) t has an honest vertex v

1

associated with sl

1

, (ii.)

sl

2

= `(t), (iii.) sl

1

+ s ≤ sl

2

, and (iv.) t(sl

1

, sl

2

] contains fewer than τ s vertices. Let

sl

1

= hsl

0

< hsl

1

< · · · < hsl

g

≤ sl

2

denote the increasing sequence of all honest indices in the interval {sl

1

, . . . , sl

2

} and, for each

0 ≤ i ≤ g, let t

i

denote the tine for which `(t

i

) = hsl

i

. Observe that t[0 : sl

1

] = t

0

and, in case

g > 0, length(t

i

) < length(t

i+1

) for i ∈ {0, . . . , g − 1}. Observe that,

length(t) = length(t[0 : sl

2

])

(∗)

≥ length(t

g

) ≥ length(t

0

) + g ≥ length(t[0 : sl

1

]) + g .

where

(∗)

≥ follows from viability of t. Thus length(t) ≥ length(t[0 : sl

1

]) + g.

Note that g is the number of zeros appearing in the string ˇw = w

sl

1

+1

, . . . , w

sl

2

and, in light of

the discussion above g < τ s. We say sl

2

is the target of this violation, and that sl

2

− sl

1

= | ˇw| is

the rank.

Observe now that

Pr[w admits a (τ, s)-hcg violation with target sl and rank `]

≤ Pr[#

0

( ˇw) < τ s] = Pr[#

0

( ˇw) < [(1 − α) − δ]`]

≤ Pr

[

wt( ˇw) < E[wt( ˇw)] − `δ

]

≤ exp(−2δ

2

`) ,

where the last inequality follows from the Chernoff-Hoeffding bounds (Theorem 4.16).

24

By the union bound, applied over ranks `, we conclude that

Pr[W admits a (τ, s)-HCG violation with target sl]

≤

∑

`≥s

exp(−2δ

2

`) ≤

∫

∞

`=s−1

exp(−2δ

2

`) d`

=

1

2δ

2

exp(−2δ

2

(s − 1)) ≤

exp(2δ

2

)

2δ

2

exp(−2δ

2

s) .

To complete the argument, a union bound over all targets yields the bound:

Pr[w does not possess (s, τ )-hcg] ≤ L

exp(2δ

2

)

2δ

2

exp(−2δ

2

s) ≤

(

(δ

−2

+ 3)/2

)

L exp(−2δ

2

s) ,

where the final inequality follows from exp(x) ≤ 1 + (3/2)x for x ∈ (0, 1/2).

Lemma 4.22 (Abstract Chain Growth). Consider w = w

1

, . . . , w

L

drawn in {0, 1}

L

so that each

w

i

independently takes the value 1 with probability 1/2 − . Then for s > 0, we have

Pr[w does not satisfy (1/4, s)-cg] ≤

cg

(1/4, s; L, ) , (

−2

+ 4)L exp(−

2

s/2) .

Proof. The corollary follows directly by combining Lemmas 4.19, 4.20, and 4.21 with the parameters

s

∃cq

= s/4, and s

hcg

= s/2, δ

hcg

= .

Specifically, applying Lemma 4.19 with s

∃cq

= s/4,

Pr[w does not satisfy s/4-∃cq] ≤ (

−2

/2 + 2)L exp(−

2

s/2) .

Likewise, applying Lemma 4.21 with s

hcg

= s/2, δ

hcg

= , and τ = 1/2 + − δ

hcg

= 1/2,

Pr[w does not satisfy (1/2, s/2)-hcg] ≤ (

−2

/2 + 2)L exp(−

2

s) .

In light of Lemma 4.20, and considering that s

∃cq

≤ s

hcg

/2,

Pr[w does not satisfy (1/4, s)-cg] ≤ (

−2

/2 + 2)L exp(−

2

s/2) + (

−2

/2 + 2)L exp(−

2

s)

≤ (

−2

+ 4)L exp(−

2

s/2) .

4.6 Common Prefix

Definition 4.23 (Flat forks; the ∼ relation). For two tines t

1

and t

2

of a fork F , we write t

1

∼ t

2

if they share an edge. Note that ∼ is an equivalence relation on the set of nontrivial tines; on the

other hand, if t

denotes the “empty” tine consisting solely of the root vertex then t

6

∼ t for any

tine t. We say that a fork is flat if it has two tines t

1

6

∼ t

2

of length equal to the height of the fork.

A string w ∈ {0, 1}

∗

is said to be forkable if there is a flat fork F ` w.

Note that in order for an execution of π

iSPoS

to yield two entirely disjoint chains of maximum

length (in the view of an observer), the characteristic string associated with the execution must be

forkable. Our goal is to establish the following upper bound on the number of forkable strings. We

then apply this to control the probability of a common prefix violation.

Theorem 4.24. Let ∈ (0, 1) and let w be a characteristic string drawn from {0, 1}

n

by indepen-

dently assigning each w

i

= 1 with probability (1 − )/2. Then Pr[w is forkable] = 2

−Ω(

√

n)

.

Here the Ω() notation hides a constant that depends on . Note that in subsequent work, Russell

et al. [42] improved this bound to 2

−Ω(n)

.

25

Structural features of forks: Reach, gap, reserve, and margin.

Definition 4.25 (Gap, reserve and reach). Let F ` w be a closed fork and let

ˆ

t denote the (unique)

tine of maximum length in F . We define the gap of a tine t, denoted gap(t), to be the difference in

length between

ˆ

t and t; thus

gap(t) = length(ˆt) − length(t) .

We define the reserve of a tine t to be the number of adversarial indices appearing in w after the

last index in t; specifically, if t is given by the path (r, v

1

, . . . , v

k

), where r is the root of F , we define

reserve(t) = |{i | w

i

= 1 and i > `(v

k

)}| .

We remark that this quantity depends both on F and the specific string w associated with F . Finally,

for a tine t we define

reach(t) = reserve(t) − gap(t) .

Definition 4.26 (Margin). For a closed fork F ` w we define ρ(F ) to be the maximum reach taken

over all tines in F :

ρ(F ) = max

t

reach(t) .

Likewise, we define the margin of F , denoted μ(F ), to be the “penultimate” reach taken over edge-

disjoint tines of F : specifically,

margin(F ) = μ(F ) = max

t

1

6

∼t

2

(

min{reach(t

1

), reach(t

2

)}

)

. (2)

We remark that the maxima above can always be obtained by honest tines. Specifically, if t is

an adversarial tine of a fork F ` w, reach(t) ≤ reach(

t), where t is the longest honest prefix of t.

As ∼ is an equivalence relation on the nonempty tines, it follows that there is always a pair of

(edge-disjoint) tines t

1

and t

2

achieving the maximum in the defining equation (2) which satisfy

reach(t

1

) = ρ(F ) ≥ reach(t

2

) = μ(F ).

The relevance of margin to the notion of forkability is reflected in the following proposition.

Proposition 4.27. A string w is forkable if and only if there is a closed fork F ` w for which

margin(F ) ≥ 0.

Proof. If w has no honest indices, then the trivial fork consisting of a single root node is flat, closed,

and has non-negative margin; thus the two conditions are equivalent. Consider a forkable string w

with at least one honest index and let ˆ

i denote the largest honest index of w. Let F be a flat fork

for w and let

F ` w be the closure of F (obtained from F by removing any adversarial vertices from

the ends of the tines of F ). Note that the tine ˆt containing ˆi is the longest tine in

F , as this is the

largest honest index of w. On the other hand, F is flat, in which case there are two edge-disjoint

tines t

1

and t

2

with length at least that of ˆt. The prefixes of these two tines in F must clearly have

reserve no less than gap (and hence non-negative reach); thus margin(

F ) ≥ 0 as desired.

On the other hand, suppose w has a closed fork with margin(F ) ≥ 0, in which case there are

two edge-disjoint tines of F , t

1

and t

2

, for which reach(t

i

) ≥ 0. Then we can produce a flat fork by

simply adding to each t

i

a path of gap(t

i

) vertices labeled with the subsequent adversarial indices

promised by the definition of reserve().

In light of this proposition, for a string w we focus our attention on the quantities

ρ(w) = max

F `w,

F closed

ρ(F ) , μ(w) = max

F `w,

F closed

μ(F ) ,

26

and, for convenience,

m(w) = (ρ(w), μ(w)) .

Note that this overloads the notation ρ(·) and μ(·) so that they apply to both forks and strings,

but the setting will be clear from context. We remark that the definitions do not guarantee a priori

that ρ(w) and μ(w) can be achieved by the same fork, though this will be established in the lemma

below. In any case, it is clear that ρ(w) ≥ 0 and ρ(w) ≥ μ(w) for all strings w; furthermore, by

Proposition 4.27 a string w is forkable if and only if μ(w) ≥ 0. We refer to μ(w) as the margin of

the string w.

In preparation for the proof of Theorem 4.24, we establish a recursive description for these

quantities.

Lemma 4.28. m() = (0, 0) and, for all nonempty strings w ∈ {0, 1}

∗

,

m(w1) = (ρ(w) + 1, μ(w) + 1) , and

m(w0) =

(ρ(w) − 1, 0) if ρ(w) > μ(w) = 0,

(0, μ(w) − 1) if ρ(w) = 0,

(ρ(w) − 1, μ(w) − 1) otherwise.

Furthermore, for every string w, there is a closed fork F

w

` w for which m(w) = (ρ(F

w

), μ(F

w

)).

Proof. The proof proceeds by induction. If w = , define F

to be the trivial fork; F

` w is the

unique closed fork for this string and m() = (0, 0) = (ρ(F

), μ(F

)), as desired.

In general, we consider m(w

′

) for a string w

′

= wx—where w ∈ {0, 1}

∗

and x ∈ {0, 1}; the

argument recursively expands m(w

′

) in terms of m(w) and the value of the last symbol x. In each

case, we consider the relationship between two closed forks F v F

′

where F ` w and F

′

` w

′

= wx.

In the case where x = 1, we must have F = F

′

as graphs, because the forks are assumed to

be closed; it is easy to see that the reach of any tine t of F ` w has increased by exactly one

when viewed as a tine of F

′

` w

′

. We write reach

F

′

(t) = reach

F

(t) + 1, where we introduce the

notation reach

() to denote the reach in a particular fork. It follows that ρ(F

′

) = ρ(F ) + 1 and

μ(F

′

) = μ(F ) + 1. If F

∗

` w

′

is a closed fork for which ρ(F

∗

) = ρ(w

′

), note that F

∗

may be

treated as a fork for w and, applying the argument above, we find that ρ(w

′

) ≤ ρ(w) + 1. A similar

argument implies that μ(w

′

) ≤ μ(w) + 1. On the other hand, by induction there is a fork F

w

for

which m(w) = (ρ(F

w

), μ(F

w

)) and hence m(w

′

) ≥ (ρ(w) + 1, μ(w) + 1). We conclude that

m(w

′

) = (ρ(w) + 1, μ(w) + 1) . (3)

Moreover, m(w

′

) = (ρ(F

w

), μ(F

w

)), where F

w

is treated as a fork for w

′

= w1.

The case when x = 0 is more delicate. As above, we consider the relationship between two

closed forks F ` w and F

′

` w

′

= w0 for which F v F

′

. Here F

′

is necessarily obtained from F

by appending a path labeled with a string of the form 1

a

0 to the end of a tine t of F . (In fact,

it is easy to see that we may always assume that this is appended to an honest tine.) In order

for this to be possible, gap(t) ≤ reserve(t) (which is to say that reach(t) ≥ 0) and, in particular,

gap(t) ≤ a ≤ reserve(t): for the first inequality, note that the depth of the new honest vertex

must exceed that of the deepest (honest) vertex in F and hence a ≥ gap(t); as for the second

inequality, there are only reserve(t) possible adversarial indices that may be added to t and hence

a ≤ reserve(t). We define the quantity ˜a ≥ 0 by the equation a = gap(t) + ˜a and let t

′

denote the

tine (of F

′

) resulting by extending t in this way. We say that ˜a is the parameter for this pair of

forks F v F

′

.

27

Of course, every honest tine t of F is an honest tine of F

′

and it is clear that reach

F

′

(t) =

reach

F

(t) − (˜a + 1), as the length of the longest tine t

′

in F

′

exceeds the length of the longest tine

of F by exactly ˜a + 1. Note that the reach of the new honest tine t

′

(in F

′

) is always 0, as both

gap(t

′

) and reserve(t

′

) are zero. It remains to describe how μ(w) and ρ(w) are determined by this

process.

The case ρ(w) > μ(w) = 0. By induction, there is a fork F

w

for which m(w) = (ρ(F

w

), μ(F

w

)).

Let t

1

and t

2

be edge-disjoint tines of F

w

for which ρ(F

w

) = reach(t

1

) and μ(F

w

) = reach(t

2

).

Define F

′

` w

′

to be the fork obtained by extending the tine t

2

of F

w

with parameter ˜a = 0

to yield a new tine t

′

2

in F

′

. Then reach

F

′

(t

1

) = ρ(w) − 1 and reach

F

′

(t

′

2

) = 0. It follows that

ρ(w0) ≥ ρ(w) − 1 and μ(w0) ≥ 0. We will show that ρ(w0) ≤ ρ(w) − 1 and that μ(w0) ≤ 0,

in which case we can conclude that

ρ(w0) = ρ(w) − 1 and μ(w0) = 0 .

Moreover, the fork F

w

′

= F

′

achieves these statistics, as desired.

We return to establish that ρ(w0) ≤ ρ(w) − 1 and that μ(w0) ≤ 0. Let F

∗

` w0 be a closed

fork for which ρ(w0) = ρ(F

∗

) and let F ` w be the unique closed fork for which F v F

∗

;

as above, let ˜a denote the parameter for this extension. Let t

∗

be an honest tine of F

∗

so

that reach

F

∗

(t

∗

) = ρ(w0). If t

∗

is a tine of F , reach

F

∗

(t

∗

) = reach

F

(t

∗

) − (˜a + 1) ≤ ρ(w) − 1.

Otherwise t

∗

was obtained by extension and reach

F

∗

(t

∗

) = 0 ≤ ρ(w) − 1 by assumption. In

either case ρ(w0) ≤ ρ(w) − 1, as desired. It remains to show that μ(w0) ≤ 0. Now consider

F

∗

` w0 to be a closed fork for which μ(F

∗

) = μ(w0). Let t

∗

1

and t

∗

2

be two edge-disjoint

honest tines of F

∗

so that reach

F

∗

(t

∗

1

) = ρ(F

∗

) and reach

F

∗

(t

∗

2

) = μ(F

∗

) = μ(w0). Let F ` w

be the unique closed fork for which F v F

∗

and let ˜a be the parameter for this extension.

If both t

∗

1

and t

∗

2

are tines of F , reach

F

∗

(t

∗

i

) = reach

F

(t

∗

i

) − (˜a + 1) and, in particular,

reach

F

(t

∗

1

) ≥ reach

F

(t

∗

2

). It follows that reach

F

(t

∗

2

) ≤ μ(F ) ≤ μ(w) = 0 and hence that

μ(w0) < 0. Otherwise, one of the two tines was the result of extension and has zero reach()

in F

∗

. As reach

F

∗

(t

∗

1

) ≥ reach

F

∗

(t

∗

2

), in either case it follows that μ(F

∗

) = reach

F

∗

(t

∗

2

) ≤ 0,

as desired.

The case ρ(w) = 0. By induction, there is a fork F

w

for which m(w) = (ρ(F

w

), μ(F

w

)). Let t

1

and t

2

be edge-disjoint tines of F

w

for which ρ(F

w

) = reach(t

1

) and μ(F

w

) = reach(t

2

). Define

F

′

` w

′

to be the fork obtained by extending the tine t

1

of F

w

with parameter ˜a = 0 to yield

a new tine t

′

1

in F

′

. Then reach

F

′

(t

′

1

) = 0 and reach

F

′

(t

2

) = reach

F

(t

2

) − 1. It follows that

ρ(w0) ≥ 0 and μ(w0) ≥ μ(w) − 1. We will show that ρ(w0) ≤ 0 and that μ(w0) ≤ μ(w) − 1,

in which case we can conclude that

ρ(w0) = 0 and μ(w0) = μ(w) − 1 .

Moreover, the fork F

w

′

= F

′

achieves these statistics, as desired.

We return to establish that ρ(w0) ≤ 0 and that μ(w0) ≤ μ(w) − 1. Let F

∗

` w0 be a

closed fork for which ρ(w0) = ρ(F

∗

) and let F ` w be the unique closed fork for which

F v F

∗

; as above, let ˜a denote the parameter for this extension. Let t

∗

be an honest tine

of F

∗

so that reach

F

∗

(t

∗

) = ρ(w0). Note that t

∗

cannot be a tine of F ; if it were then

reach

F

∗

(t

∗

) = reach

F

(t

∗

) − (˜a + 1) ≤ ρ(w) − 1 < 0 which contradicts ρ(w0) ≥ 0. Thus t

∗

was obtained by extension and reach

F

∗

(t

∗

) = 0. It remains to show that μ(w0) ≤ 0. Now let

F

∗

` w0 be a closed fork for which μ(F

∗

) = μ(w0). Let t

∗

1

and t

∗

2

be two edge-disjoint honest

tines of F

∗

so that reach

F

∗

(t

∗

1

) = ρ(F

∗

) and reach

F

∗

(t

∗

2

) = μ(F

∗

) = μ(w0). Let F ` w be the

28

unique closed fork for which F v F

∗

and let ˜a be the parameter for this extension. Similarly,

t

∗

1

cannot be a tine of F ; if it were, ρ(F

∗

) = reach

F

∗

(t

∗

1

) = reach

F

(t

∗

1

) − (˜a + 1) ≤ ρ(F ) − 1 ≤

ρ(w) − 1 < 0 which contradicts ρ(F ) ≥ 0. It follows that t

∗

1

must extend a tine t

1

of F for

which reach

F

(t

1

) = 0, because extension can only occur for tines of non-negative reach and

ρ(F ) = 0 = ρ(w). Thus t

∗

2

is a tine of F and t

1

6

∼ t

∗

2

so that reach

F

(t

∗

2

) ≤ μ(F ) ≤ μ(w) and

we conclude that μ(w0) = reach

F

∗

(t

∗

2

) ≤ reach

F

(t

∗

2

) − 1 ≤ μ(w) − 1, as desired.

The case ρ(w) > 0, μ(w) 6 = 0. By induction, there is a fork F

w

for which m(w) = (ρ(F

w

), μ(F

w

)).

Let t

1

and t

2

be edge-disjoint tines of F

w

for which ρ(F

w

) = reach(t

1

) and μ(F

w

) = reach(t

2

).

In fact, any extension of F

w

will suffice for the construction; for concreteness, define F

′

`

w

′

to be the fork obtained by extending the tine t

1

of F

w

with parameter ˜a = 0. Then

reach

F

′

(t

i

) = reach

F

w

(t

i

) − 1. It follows that ρ(w0) ≥ ρ(w) − 1 and μ(w0) ≥ μ(w) − 1. We

will show that ρ(w0) ≤ ρ(w) − 1 and that μ(w0) ≤ μ(w) − 1, in which case we can conclude

that

ρ(w0) = ρ(w) − 1 and μ(w0) = μ(w) − 1 .

Moreover, the fork F

w

′

= F

′

achieves these statistics, as desired.

We return to establish that ρ(w0) ≤ ρ(w) − 1 and that μ(w0) ≤ μ(w) − 1. Let F

∗

` w0 be a

closed fork for which ρ(w0) = ρ(F

∗

) and let F ` w be the unique closed fork for which F v F

∗

;

as above, let ˜a denote the parameter for this extension. Let t

∗

be an honest tine of F

∗

so that

reach

F

∗

(t

∗

) = ρ(w0). Note that if t

∗

is a tine of F then reach

F

∗

(t

∗

) = reach

F

(t

∗

) − (˜a + 1) ≤

ρ(w) − 1; otherwise t

∗

is obtained by extension and reach

F

∗

(t

∗

) = 0 ≤ ρ(w) − 1, as desired.

(Recall that ρ(w) > 0.) It remains to show that μ(w0) ≤ μ(w) − 1. Now let F

∗

` w0 be a

closed fork for which μ(F

∗

) = μ(w0). Let t

∗

1

and t

∗

2

be two edge-disjoint honest tines of F

∗

so

that reach

F

∗

(t

∗

1

) = ρ(F

∗

) and reach

F

∗

(t

∗

2

) = μ(F

∗

) = μ(w0). Let F ` w be the unique closed

fork for which F v F

∗

and let ˜a be the parameter for this extension. If both t

∗

1

and t

∗

2

are

tines of F then reach

F

∗

(t

∗

i

) = reach

F

(t

∗

i

) − (˜a + 1) and, in particular, reach

F

(t

∗

1

) ≥ reach

F

(t

∗

2

)

so that reach

F

(t

∗

2

) ≤ μ(w) and reach

F ∗

(t

∗

2

) ≤ μ(w) − 1, as desired.

To complete the argument, we consider the case that one of the tines t

∗

i

arises by extension.

Note that in this case reach

F

∗

(t

∗

2

) ≤ 0, as either t

∗

2

is obtained by extension so that it has zero

reach, or t

∗

1

is obtained by extension so that reach

F

∗

(t

∗

2

) ≤ reach

F

∗

(t

∗

1

) = 0. Here we further

separate the analysis into two cases depending on the sign of μ(w):

• If μ(w) > 0, then reach

F

∗

(t

∗

2

) ≤ 0 ≤ μ(w) − 1, as desired.

• If μ(w) < 0 then t

∗

2

cannot be the extension of a tine in F . To see this, suppose to the

contrary that t

∗

2

extends a tine t

2

of F ; then reach

F

(t

2

) ≥ 0. Additionally, t

∗

1

must be a

tine of F , edge-disjoint from t

2

, and reach

F

(t

∗

1

) = reach

F

∗

(t

∗

1

) + (˜a + 1) > 0. It follows

that μ(w) ≥ μ(F ) ≥ 0, a contradiction.

The other possibility is that t

∗

1

is an extension of a tine t

1

of F in which case reach

F

(t

1

) ≥

0. Note that t

∗

2

is a tine of F and edge-disjoint from t

1

; thus min(reach

F

(t

∗

2

), reach

F

(t

1

)) ≤

μ(F ) < 0 and reach

F

(t

∗

2

) ≤ μ(F ). We conclude that reach

F

∗

(t

∗

2

) = reach

F

(t

∗

2

)−(˜a+1) ≤

μ(w) − 1, as desired.

With this recursive description in place, we return to the proof of Theorem 4.24, which we

restate below for convenience.

Theorem 4.24, restated. Let ∈ (0, 1) and let w be a string drawn from {0, 1}

n

by independently

assigning each w

i

= 1 with probability (1 − )/2. Then

Pr[w is forkable] = 2

−Ω(

√

n)

.

29

Proof of Theorem 4.24. The theorem concerns the probability distribution on {0, 1}

n

given by in-

dependently selecting each w

i

∈ {0, 1} so that

Pr[w

i

= 0] =

1 +

2

= 1 − Pr[w

i

= 1] .

For the string w

1

. . . w

n

chosen with the probability distribution above, define the random variables

R

t

= ρ(w

1

. . . w

t

) and M

t

= μ(w

1

. . . w

t

) .

Our goal is to establish that

Pr[w forkable] = Pr[M

n

≥ 0] = 2

−Ω(

√

n)

.

We extract from the statement of Lemma 4.28 some facts about these random variables.

R

t

> 0 =⇒

{

R

t+1

= R

t

+ 1 if w

t+1

= 1,

R

t+1

= R

t

− 1 if w

t+1

= 0;

(4)

M

t

< 0 =⇒

{

M

t+1

= M

t

+ 1 if w

t+1

= 1,

M

t+1

= M

t

− 1 if w

t+1

= 0;

(5)

R

t

= 0 =⇒

R

t+1

= 1 if w

t+1

= 1,

R

t+1

= 0 if w

t+1

= 0,

M

t+1

< 0 if w

t

= 0.

(6)

In light of the properties (4) above, the random variables R

t

are quite well-behaved when

positive—in particular, considering the distribution placed on each w

i

, they simply follow the

familiar biased random walk of Figure 6. Likewise, considering the properties (5), the random

variables M

t

follow a biased random walk when negative. The remainder of the proof combines

these probability laws with (6) and the fact that M

t

() ≤ R

t

() to establish that M

n

< 0 with high

probability.

0

−1

· · ·

1

· · ·

p

qq

p

p

q

p

q

Figure 6: The simple biased walk where p = (1 + )/2 and q = 1 − p.

We recall two basic facts about the standard biased walk associated with the Markov chain of

Figure 6. Let Z

i

∈ {±1} (for i = 1, 2, . . .) denote a family of independent random variables for

which Pr[Z

i

= 1] = (1 − )/2. Then the biased walk given by the variables Y

t

=

∑

t

i

Z

i

has the

following properties.

Constant escape probability; gambler’s ruin. With constant probability, depending only on

, Y

t

6

= 1 for all t > 0. In general, for each k > 0,

Pr[∃t, Y

t

= k] = α

k

, (7)

for a constant α < 1 depending only on . (In fact, the constant α is (1 − )/(1 + ); see, e.g.,

[29, Chapter 12] for a complete development.)

30

Concentration (the Chernoff bound). Consider T steps of the biased walk beginning at state

0; then the resulting value is tightly concentrated around −T . Specifically, E[Y

T

] = −T and

Pr

[

Y

T

> −

T

2

]

= 2

−Ω(T )

. (8)

(The constant hidden in the Ω() notation depends only on . See, e.g., [1, Cor. A.1.14].)

Partitioning the string w, we write w = w

(1)

· · · w

(

√

n)

where w

(t)

= w

1+a

t−1

. . . w

a

t

and a

t

= dt

√

ne,

for t = 0, 1, . . .. Let R

(0)

= 0 and R

(t)

= R

a

t

; similarly define M

(0)

= 0 and M

(t)

= M

a

t

. Fix δ

to be a small constant.

We define three events based on the random variables R

(t)

and M

(t)

:

Hot We let Hot

t

denote the event that R

(t)

≥ δ

√

n and M

(t)

≥ −δ

√

n.

Volatile We let Vol

t

denote the event that −δ

√

n ≤ M

(t)

≤ R

(t)

< δ

√

n.

Cold We let Cold

t

denote the event that M

(t)

< −δ

√

n.

Note that for each t, exactly one of these events occurs—they partition the probability space. Then

we will establish that

Pr[Cold

t+1

| Cold

t

] ≥ 1 − 2

−Ω(

√

n)

, (9)

Pr[Cold

t+1

| Vol

t

] ≥ Ω() , (10)

Pr[Hot

t+1

| Vol

t

] ≤ 2

−Ω(

√

n)

. (11)

Cold

Vol

Hot

≈ 1

θ(1)

θ(1)

≈ 0

Figure 7: An illustration of the transitions between Cold, Vol, and Hot.

Note that the event Vol

0

occurs by definition. Assuming these inequalities, we observe that the

system is very likely to eventually become cold, and stay that way. In this case, Cold

√

n

occurs,

M

◦

n

< δ

√

n < 0, and w is not forkable. Specifically, note that the probability that the system

ever transitions from volatile to hot is no more than 2

−Ω(

√

n)

(as transition from Vol to Hot is

bounded above by 2

−Ω(

√

n)

, and there are no more than

√

n possible transition opportunities).

Note, also, that while the system is volatile, it transitions to cold with constant probability during

each period. In particular, the probability that the system is volatile for the entire process is no

31

more that 2

−Ω(

√

n)

. Finally, note that the probability that the system ever transitions out of the

cold state is no more than 2

−Ω(

√

n)

(again, there are at most

√

n possible times when this could

happen, and any individual transition occurs with probability 2

−Ω(

√

n)

). It follows that the system

is cold at the end of the process with probability 1 − 2

−Ω(

√

n)

.

It remains to establish the three inequalities (9), (10), and (11).

Inequality (9): This follows directly from (4) and (7). Specifically, in light of (5) the random

variables M

i

follow the probability law of the simple biased walk when they are negative.

Conditioned on M

(t)

= M

a

t

< −δ

√

n, the probability that any future M

s

ever climbs to the

value −1 is no more than α

−δ

√

n

= 2

−Ω(n)

, as desired. (Here α < 1 is a fixed constant that

depends only on .)

Inequality (10): This follows from (4), (6), (7), and (8). Specifically, conditioned on Vol

t

, R

(t)

≤

δ

√

n. Recall from (4) that the random variables R

i

follow the probability law of the simple

biased walk when they are positive. Let D be the event that R

i

> 0 for all a

t

≤ i < a

t

+2δ

√

n.

According to (8), then, where we take T = 2δ

√

n, Pr[D] ≤ 2

−Ω(

√

n)

. With near certainty, then,

the random variables R

i

visit the value 0 during this period. Observe that if R

i

= 0 then,

by (6), M

i+1

≤ −1 with constant probability and (conditioned on this), by (7), with constant

probability the subsequent random variables M

j

do not return to the value 0. Additionally,

in light of (8), the probability that there is a sequence w

i

. . . w

j

of length at least 2(δ/)

√

n

for which

j

∑

k=i

{

1 if w

k

= 1,

−1 if w

k

= 0.

≥ −δ

√

n

is no more than (

√

n)

2

2

−Ω(

√

n)

. It follows that with constant probability, the walk (of R

i

)

hits 0, as described above, and then M

i

terminates at a value less than −δ

√

n.

Inequality (11): This follows from (4), (6), (7), and (8). Specifically, conditioned on Vol

t

, R

(t)

≤

δ

√

n. Recall from (4) that the random variables R

i

follow the probability law of the simple

biased walk when they are positive. Let D be the event that R

i

> 0 for all a

t

≤ i < a

t

+2δ

√

n.

According to (8), then, where we take T = 2δ

√

n, Pr[D] ≤ 2

−Ω(

√

n)

. With near certainty,

then, the random variables R

i

visit the value 0 during this period. Conditioned on D, in order

for R

a

t+1

≥ δ

√

n there must be a sequence of these random variables 0 = R

i

, R

i+1

, . . . , R

j

=

bδ

√

nc so that none of these take the value 0 except the first. (Such a sequence arises by

taking i to be the last time the variables R

a

t

, . . . visit 0 and j the first subsequent time

that the sequence is larger than δ

√

n.) In light of (7), the probability of such a subsequence

appearing at a particular value for i is no more than α

−δ

√

n

. It follows that the probability

that R

a

t+1

≥ δ

√

n is less than

√

nα

−δ

√

n

= 2

−Ω(

√

n)

, as desired.

Exact probabilities of forkability for explicit values of n. In order to gain further insight

regarding the density of forkable strings, we exactly computed the probability that a string w—

drawn so that each w

i

is independently assigned the value 1 with probability p ∈ {.40, .41, . . . , .50}—

is forkable for several different lengths. These results are presented in Figure 8.

Reducing common prefix to forkability. Returning now to the challenge of common prefix,

we note that the random assignment of slots to stakeholders given by F

LS

guarantees that the

coordinates of the associated characteristic string w are independent Bernoulli random variables

taking the value 1 with probability equal to the adversarial stake. Thus Theorem 4.24 establishes

32

0.4 0.42 0.44 0.46 0.48 0.5

0

0.2

0.4

0.6

0.8

Probability that w

i

= 1

Probability of forkability

Probability of Forkability

n = 500

n = 1000

n = 1500

n = 2000

Figure 8: Graphs of the probability that a string drawn so that each bit is independently assigned

the value 1 with a certain probability is forkable. Graphs for string lengths n = 500, 1000, 1500, 2000

are shown for probabilities .40, .41, . . . , .49, .50.

that no execution of the protocol π

iSPoS

can induce two tines (chains) of maximal length with no

common prefix.

In the context of π

iSPoS

, however, we wish to establish a much stronger common prefix property:

any pair of chains which could, in principle, be presented by the adversary to an honest party must

have a “recent” common prefix, in the sense that removing a small number of blocks from the

shorter chain results in a prefix of the longer chain.

To formally articulate and prove this property, we introduce a further definition regarding tines

and forks.

Definition 4.29 (Divergence). Let F be a fork for a string w ∈ {0, 1}

∗

. For two viable tines t and

t

′

of F we define the notation t/t

′

by the rule

t/t

′

= length(t) − length(t ∩ t

′

) ,

where t ∩ t

′

denotes the common prefix of t and t

′

. Then define the divergence of two viable tines

t

1

and t

2

to be the quantity

div(t

1

, t

2

) =

t

1

/t

2

if `(t

1

) < `(t

2

),

t

2

/t

1

if `(t

2

) < `(t

1

),

max(t

1

/t

2

, t

2

/t

1

) if `(t

1

) = `(t

2

).

We overload this notation by defining divergence for F as the maximum over all pairs of viable

tines:

div(F ) = max

t

1

, t

2

viable

tines of F

div(t

1

, t

2

) .

Finally, define the divergence of w to be the maximum such divergence over all possible forks for w:

div(w) = max

F `w

div(F ) .

Observe that if div(t

1

, t

2

) ≤ k and, say, `(t

1

) ≤ `(t

2

), the tine t

dk

1

is a prefix of t

2

.

33

Divergence and common prefix violations. Divergence directly reflects the possibility of a

common prefix violation. In particular, the characteristic string w satisfies k-cp if and only if

div(w) ≤ k. We first establish that a string with large divergence must have a large forkable

substring. We then apply this in Theorem 4.31 below to conclude that characteristic strings arising

from π

iSPoS

(that is, for which each bit is a Bernoulli random variable) are unlikely to have large

divergence and, hence, possess the common prefix property. The specific fork of relevance, as

described above, is F

A

arising from the protocol; however, the analysis below will show that no

fork realizes large divergence.

Theorem 4.30. Let w ∈ {0, 1}

∗

. Then there is forkable substring ˇw of w with | ˇw| ≥ div(w).

Proof. Consider a string w ∈ {0, 1}

n

, a fork F ` w, and a pair of viable tines (t

1

, t

2

) for which

`(t

1

) ≤ `(t

2

) and t

1

/t

2

= div(w) . (12)

We may further assume that

|`(t

2

) − `(t

1

)| is minimum among all pairs of tines for which (12) holds. (13)

We begin by identifying the substring ˇw; the remainder of the proof is devoted to constructing

a flat fork for ˇw to establish forkability. Let y denote the last vertex on the tine t

1

∩ t

2

, as in the

diagram below, and let α , `(y) = `(t

1

∩ t

2

).

y

t

1

t

2

Let β denote the smallest honest index of w for which β ≥ `(t

2

), with the convention that if there

is no such index we define β = n + 1. It follows that `(t

1

) < β. Specifically, if `(t

1

) = `(t

2

) = i then

this label i is adversarial and hence `(t

1

) < β; otherwise, either `(t

1

) < `(t

2

) ≤ β or β = n + 1 and

thus `(t

1

) < β.

These indices, α and β, distinguish the substring ˇw = w

α+1

. . . w

β−1

which will be the subject

of the remainder of the proof. As the function `(·) is strictly increasing along any tine, observe that

| ˇw| = β − α − 1 ≥ `(t

1

) − `(y) ≥ length(t

1

) − length(t

1

∩ t

2

) = div(w) ,

so ˇw has the desired length and it suffices to establish that it is forkable.

We briefly summarize the proof before presenting the details. We begin by establishing several

structural properties of the tines t

1

and t

2

that follow from the assumptions (12) and (13) above.

To establish that ˇw is forkable we then extract from F a flat fork (for ˇw) in two steps: (i.) the fork

F is subjected to some minor restructuring to ensure that all “long” tines pass through y; (ii.) a

flat fork is constructed by treating the vertex y as the root of a portion of the subtree of F labeled

with indices of ˇw. At the conclusion of the construction, segments of the two tines t

1

and t

2

will

yield the required “long, disjoint” tines satisfying the definition of forkable.

We observe, first of all, that the vertex y cannot be adversarial: otherwise it is easy to construct

an alternative fork

˜

F ` w and a pair of tines in

˜

F that achieve larger divergence. Specifically,

construct

˜

F from F by adding a new (adversarial) vertex ˜y to F for which `(˜y) = `(y), adding an

edge to ˜y from the vertex preceding y, and replacing the edge of t

1

following y with one from ˜y; then

the other relevant properties of the fork are maintained, but t

1

/t

2

—and hence the divergence—of

the resulting tines has increased by one. (See the diagram below.)

34

y

˜y

t

1

t

2

A similar argument implies that the fork F

0

` w

1

. . . w

α

obtained by including only those vertices

of F with labels less than or equal to α = `(y) has a unique vertex of depth depth(y) (namely, y

itself). In the presence of another vertex ˜y (of F

0

) with depth depth(y), “redirecting” t

1

through

˜y (as in the argument above) would likewise result in a fork with larger divergence. Note that `(·)

would indeed be increasing along this new tine (resulting from redirecting t

1

) because `(˜y) ≤ `(y)

according to the definition of F

0

. As α is the last index of the string, this additionally implies that

F

0

has no vertices of depth exceeding depth(y).

We remark that the minimality assumption (13) implies that any honest index h for which

h < β has depth no more than min(length(t

1

), length(t

2

)): specifically,

h < β =⇒ d(h) ≤ min(length(t

1

), length(t

2

)) . (14)

To see this, consider an honest index h < β and the tine t

h

for which `(t

h

) = h. Recall that t

1

and t

2

are viable; as h < `(t

2

) it follows immediately that d(h) ≤ length(t

2

). Similarly, if h ≤ `(t

1

)

then d(h) ≤ length(t

1

), so it remains to settle the case when `(t

1

) < h < `(t

2

): in particular, in

this regime we wish to likewise guarantee that d(h) ≤ length(t

1

). For the sake of contradiction,

assume that length(t

h

) = d(h) > length(t

1

). Considering the tine t

h

, we separately investigate

two cases depending on whether t

h

shares an edge with t

1

after the vertex y. If, indeed, t

h

and t

1

share an edge after the vertex y then t

h

and t

2

do not share such an edge, and we observe that

div(w) ≥ t

h

/t

2

≥ t

1

/t

2

= div(w) (and hence div(w) = div(t

h

, t

2

)) while |`(t

2

) − h| < |`(t

2

) − `(t

1

)|

which contradicts (13). If, on the other hand, t

h

shares no edge with t

1

after y, we similarly observe

that div(w) ≥ t

1

/t

h

≥ t

1

/t

2

= div(w) while |h − `(t

1

)| < |`(t

2

) − `(t

1

)|, which contradicts (13).

In light of the remarks above, we observe that the fork F may be “pinched” at y to yield an

essentially identical fork F

ByC

` w with the exception that all tines of length exceeding depth(y)

pass through the vertex y. Specifically, the fork F

ByC

` w is defined to be the graph obtained

from F by changing every edge of F directed towards a vertex of depth depth(y) + 1 so that it

originates from y. To see that the resulting tree is a well-defined fork, it suffices to check that `(·)

is still increasing along all tines of F

ByC

. For this purpose, consider the effect of this pinching on

an individual tine t terminating at a particular vertex v—it is replaced with a tine t

ByC

defined so

that:

• If length(t) ≤ depth(y), the tine t is unchanged: t

ByC

= t.

• Otherwise, length(t) > depth(y) and t has a vertex z of depth depth(y) + 1; note that

`(z) > `(y) because F

0

contains no vertices of depth exceeding depth(y). Then t

ByC

is

defined to be the path given by the tine terminating at y, a (new) edge from y to z, and the

suffix of t beginning at z. (As `(z) > `(y) this has the increasing label property.)

Thus the tree F

ByC

is a legal fork on the same vertex set; note that depths of vertices in F and

F

ByC

are identical.

By excising the tree rooted at y from this pinched fork F

ByC

we may extract a fork for the

string w

α+1

. . . w

n

. Specifically, consider the induced subgraph F

yC

of F

ByC

given by the vertices

{y} ∪ {z | depth(z) > depth(y)}. By treating y as a root vertex and suitably defining the labels

`

yC

of F

yC

so that `

yC

(z) = `(z) − `(y), this subgraph has the defining properties of a fork for

35

w

α+1

. . . w

n

. In particular, considering that α is honest it follows that each honest index h > α has

depth d(h) > length(y) and hence labels a vertex in F

yC

. For a tine t of F

ByC

, we let t

yC

denote

the suffix of this tine beginning at y, which forms a tine in F

yC

. (If length(t) ≤ depth(y), we define

t

yC

to consist solely of the vertex y.) Note that t

yC

1

and t

yC

2

share no edges in the fork F

yC

.

Finally, let

ˇ

F denote the tree obtained from F

yC

as the union of all tines t of F

yC

so that all

labels of t are drawn from ˇw (as it appears as a prefix of w

α+1

. . . w

n

), and

length(t) ≤ max

h≤| ˇw|

h honest

d(h) .

It is immediate that

ˇ

F ` ˇw. To conclude the proof, we show that

ˇ

F is flat. For this purpose, we

consider the tines t

yC

1

and t

yC

2

. As mentioned above, they share no edges in F

yC

, and hence the

prefixes ˇt

1

and ˇt

2

(of t

yC

1

and t

yC

2

) appearing in

ˇ

F share no edges. We wish to see that these prefixes

have maximum length in

ˇ

F , in which case

ˇ

F is flat, as desired. This is immediate for the tine ˇt

1

because all labels of t

yC

1

are drawn from ˇw and, considering (14), its depth is at least that of all

relevant honest vertices. As for ˇt

2

, observe that if `(t

2

) is not honest then β > `(t

2

) so that, as with

ˇ

t

1

, the tine ˇt

2

is labeled by ˇw so that the same argument, relying on (14), ensures that ˇt

2

has length

at least that of all relevant honest vertices. If `(t

2

) is honest, β = `(t

2

), and the terminal vertex of

t

yC

2

does not appear in

ˇ

F (as it does not index ˇw). In this case, however, length(t

yC

2

) > d(h) for

any honest index of ˇw, and it follows that length( ˇt

2

) = length(t

yC

2

) − 1 is at least the depth of any

honest index of ˇw, as desired.

Theorem 4.31. Let k, L ∈ N and ∈ (0, 1/2). Let w = w

1

. . . w

L

of length L, where each w

i

is

independently distributed in {0, 1} so that Pr[w

i

= 1] = 1/2 − . Then

Pr[w does not possess k-cp] =

cp

(k; L, ) = L exp(−Ω(

√

k)) .

The constant hidden by the Ω() notation depends only on .

Proof. Recall that w violates k-cp precisely when there is a fork F ` w for which div(F ) ≥ k. Thus

we wish to show that the probability that div(w) ≥ k is no more than exp(−Ω(

√

k) + log L).

It follows from Theorem 4.30 that if div(w) ≥ k, there is a forkable substring ˇw of length at

least k. Thus

Pr[w does not possess k-cp] ≤ Pr

[

∃α, β ∈ {1, . . . , L} so that α + k − 1 ≤ β and

w

α

. . . w

β

is forkable

]

≤

∑

1≤α≤L

∑

α+k−1≤β≤L

Pr[w

α

. . . w

β

is forkable]

︸

︷︷ ︸

(∗)

.

According to Theorem 4.24 the probability that a string of length t drawn from this distribution

is forkable is no more than exp(−c

√

t) for a positive constant c. Note that for any α ≥ 1,

L

∑

t=α+k−1

e

−c

√

t

≤

∫

∞

k−1

e

−c

√

t

dt = (2/c

2

)(1 + c

√

k − 1)e

−c

√

k−1

= e

−Ω(

√

k)

and it follows that the sum (∗) above is exp(−Ω(

√

k)). Thus

Pr[w does not possess k-cp] ≤ L · exp(−Ω(

√

k)) ≤ exp(ln L − Ω(

√

k)) ,

as desired.

36

4.7 Security Analysis of the Idealized Protocol

Theorem 4.32. Consider the execution E of π

iSPoS

over a lifetime of L slots with a (1/2 − )-

initially-bounded adversary A and environment Z. Then persistence with parameter k fails to hold

with probability no more than

CP

(k; L, ) +

H

= L · exp(−Ω(

√

k)) +

H

and liveness, with parameter u = 2(k + `), fails to hold with probability no more than

HCG

(1/2, 2k; L, ) +

∃CQ

(`; L, ) +

H

= O(L · [exp(−4

2

k) + exp(−2

2

`)]) +

H

,

where

H

denotes the probability of a collision across all queries to H by the participants in the

protocol (including A). All probabilities are conditioned on ¬Bad

1

/2−

(which is to say that we

assume executions in which the adversary is (1/2 − )-bounded).

Proof. Recall that the F

LS

[F

DSIG

] functionality directly provides a leader selection schedule and—

for any adversarial selection of corrupted players with total stake (1/2 − )—yields a characteristic

string w = w

1

, . . . , w

L

where each w

i

is selected independently and E[w

i

] = 1/2 − . Recall that

• we have the luxury of an ideal signature scheme and, furthermore,

• the probability—taken across all evaluations of the hash function by all participants of the pro-

tocol including the adversary—that a collision is observed is no more than

H

(q) , q

2

/|Range|,

where q is the total number of queries to the hash function and |Range| is the size of the range

of the function.

In light of these properties, unless there is an (unlikely) collision among the queries to the hash

function, the adopted chains form a well-defined fork F

A

` w. We observe that if w satisfies cp, hcg,

and ∃cq, the execution E satisfies CP, HCG, and ∃CQ with the same parameters. (It is possible,

of course, that the execution E actually satisfies these properties with stronger parameters, as the

execution yields a specific fork (F

A

) while the abstract quantities cp, cg, and ∃cq for w are worst

case estimates taken over all forks.)

We observe that a violation of persistence would imply directly a violation of k-CP. As a result

Pr[E violates persistence] ≤ Pr[F

A

violates k-cp] +

H

≤ Pr[w violates k-cp] +

H

≤

CP

(k; L, ) +

H

≤ L · exp(−Ω(

√

k)) +

H

,

where the last inequality follows directly from Theorem 4.31.

Now regarding liveness with parameter u = 2(k + `), we observe that it is implied by (1/2, 2k)-

HCG and `-∃CQ. Specifically, consider a transaction tx being provided as part of input data for a

period S of u slots. Let t be any viable tine that spans S. In light of `-∃CQ, t contains an honestly

generated block associated with a slot sl

1

among the first ` slots of S and—as tx was available to

the honest party generating this block—it must appear in t[0, sl

1

]. Similarly, an honest block must

be associated with a slot sl

2

among the last ` slots of S. As u = 2(k + `), sl

2

≥ sl

1

+ 2k and, as a

consequence of (1/2, 2k)-HCG, the tine t contains at least k blocks after sl

1

(but indexed by slots

in S); specifically, length(t(sl

1

, sl

2

]) ≥ k. This guarantees u-liveness. Thus

Pr[E violates liveness] ≤ Pr[F

A

violates `-∃cq, or (1/2, 2k)-hcg] +

H

≤ Pr[w violates `-∃cq, or (1/2, 2k)-hcg] +

H

≤

∃CQ

(`; L, ) +

HCG

(1/2, 2k; L, ) +

H

≤ L[O(exp(−2

2

`) + O(exp(−4

2

k)] +

H

,

which completes the proof.

37

5 Analysis of the Dynamic Stake Protocol

5.1 Using a Trusted Beacon

In the static version of the protocol in the previous section, we assumed that stake was static during

the whole execution (i.e., one epoch), meaning that stake changing hands inside a given epoch does

not affect leader election. Now we put forth a modification of protocol π

SPoS

that can be executed

over multiple epochs in such a way that each epoch’s leader election process is parameterized by

the stake distribution at a certain designated point of the previous epoch, allowing for change in

the stake distribution across epochs to affect the leader election process. As before, we construct

the protocol in a hybrid model, enhancing the F

LS

ideal functionality to now provide randomness

and auxiliary information for the leader election process throughout the epochs (the enhanced

functionality will be called F

DLS

). We then discuss how to implement F

DLS

using only F

LS

and in

this way reduce the assumption back to the simple common random string selected at setup.

Before describing the protocol for the case of dynamic stake, we need to explain the modification

of F

LS

so that multiple epochs are considered. The resulting functionality, F

DLS

, allows stakeholders

to query it for the leader selection data specific to each epoch. F

DLS

is parameterized by the initial

stake of each stakeholder before the first epoch e

1

starts; in subsequent epochs, parties will take into

consideration the stake distribution in the latest block of the previous epoch’s first R −k

′

slots (k

′

is

parameter we set below). Given that there is no predetermined view of the stakeholder distribution,

the functionality F

DLS

will provide only a random string and will leave the interpretation according

to the stakeholder distribution to the party that is calling it. The effective stakeholder distribution

is the sequence S

1

, S

2

, . . . defined as follows: S

1

is the initial stakeholder distribution; for slots

{(j − 1)R + 1, . . . , jR} for j ≥ 2 the effective stakeholder S

j

is determined by the stake allocation

that is found in the latest block with timestamp less than (j − 1)R − k

′

, provided all honest parties

agree on it, or is undefined if the honest parties disagree on it. The functionality F

DLS

is defined

in Figure 9. For maximum flexibility, and to anticipate the needs of the next phase of our analysis,

we allow the adversary to perform a lookahead on the beacon value for E ≥ 0 slots prior to the

onset of the epoch. To account for this lookahead the value of k

′

must be suitably adjusted; see

below.

Functionality F

DLS

F

DLS

incorporates the diffuse and key/transaction functionality from Section 2 and is parameter-

ized by the public keys and respective stakes of the initial (before epoch e

1

starts) stakeholders

S

0

= {(vk

1

, s

0

1

), . . . , (vk

n

, s

0

n

)} a distribution D and a leader selection function F. In addition, F

DLS

operates as follows:

• Genesis Block Generation Upon receiving (genblock req, U

i

) from stakeholder U

i

it operates

as functionality F

LS

on that message.

• Signature Key Pair Generation It operates as functionality F

LS

.

• Epoch Randomness Update Upon receiving (epochrnd req

, U

i

, e

j

) from stakeholder U

i

, if j ≥ 2

is the current epoch, F

DLS

proceeds as follows. If ρ

j

has not been set, F

DLS

samples ρ

j

← D.

Then, F

DLS

sends (epochrnd, ρ

j

) to U

i

. The adversary is allowed a lookahead of E ≥ 0 slots to

this interface where E is a parameter of the functionality.

Figure 9: Functionality F

DLS

with beacon lookahead parameter E ≥ 0.

We now describe protocol π

DPoS

, which is a modified version of π

SPoS

that updates its genesis

38

block data (and thus the leader selection process) for every new epoch. The protocol also adopts

an adaptation of the static maxvalid

S

function, defined so that it narrows selection to those chains

which share common prefix. Specifically, it adopts the following rule, parameterized by a prefix

length k:

Function maxvalid

k

(C, C). Returns the longest chain from C ∪ {C} that does not fork

from C more than k blocks. If multiple exist it returns C, if this is one of them, or it

returns the one that is listed first in C.

As a matter of notation, we simply refer to this rule as maxvalid() when the parameter can be

inferred from context.

Protocol π

DPoS

is described in Figure 10 and functions in the F

DLS

-hybrid model. We also define

an idealised version of this protocol π

iDPoS

for which it holds that access to the digital signature is

performed via the interface of F

DSIG

.

Protocol π

DPoS

; with parameters k, `, E, R, L satisfying 2(k + `) + E ≤ R.

π

DPoS

is a protocol run by a set of stakeholders, initially equal to U

1

, . . . , U

n

, interacting with F

DLS

over

a sequence of L slots S = {1, . . . , L} divided in epochs of length R < L. π

DPoS

proceeds as follows:

1. Initialization Stakeholder U

i

∈ {U

1

, . . . , U

n

}, receives from the key registration interface its

public and secret key. Then it receives the current slot from the diffuse interface and in case it

is sl

1

it sends (genblock req

, U

i

) to F

LS

, receiving (genblock, S

0

, ρ, F) as the answer. U

i

sets the

local blockchain C = B

0

= (S

0

, ρ) and the initial internal state st = H(B

0

). Otherwise, it receives

from the key registration interface the initial chain C, sets the local blockchain as C and the initial

internal state st = H(head(C)).

2. Chain Extension For every slot sl ∈ S, every online stakeholder U

i

performs the following steps:

(a) If a new epoch e

j

, with j ≥ 2, has started, U

i

defines S

j

to be the stakeholder distribution

drawn from the most recent block with timestamp less or equal to (j − 1)R − 2(k + `) − E

where E ≥ 0 is the lookahead the adversary is allowed on the beacon and sends

(epochrnd req

, U

i

, e

j

) to F

LS

, receiving (epochrnd, ρ

j

) as answer.

(b) Collect all valid chains received via broadcast into a set C, verifying that for every chain

C

′

∈ C and every block B

′

= (st

′

, d

′

, sl

′

, σ

′

) ∈ C

′

it holds that Vrf

vk

′

(σ

′

, (st

′

, d

′

, sl

′

)) = 1, where

vk

′

is the verification key of the stakeholder U

′

= F(S

j

′

, ρ

j

′

, sl

′

) with e

j

′

being the epoch in

which the block B

′

belongs (as determined by sl

′

). U

i

computes C

′

= maxvalid(C, C), sets C

′

as the new local chain and sets state st = H(head(C

′

)).

(c) If U

i

is the slot leader determined by F(S

j

, ρ

j

, sl) in the current epoch e

j

, it generates

a new block B = (st, d, sl, σ) where st is its current state, d ∈ {0, 1}

∗

is the data and

σ = Sign

sk

i

(st, d, sl) is a signature on (st, d, sl). U

i

computes C

′

= C|B, broadcasts C

′

, sets

C

′

as the new local chain and sets state st = H(head(C

′

)).

3. Transaction generation as in protocol π

SPoS

.

Figure 10: Protocol π

DPoS

With a similar argument as in Proposition 4.9 we have the following statement.

Proposition 5.1. For each PPT A, Z it holds that there is a PPT S so that

EXEC

F

DLS

[SIG]

π

DPoS

,A,Z

(λ) and EXEC

F

DLS

[F

DSIG

]

π

iDPoS

,S,Z

(λ)

are computationally indistinguishable.

39

In the light of this proposition in the remaining of this section we will focus on the analysis of

iDPoS.

Remark 1. The modification to maxvalid(·) to not diverge more than k blocks from the last chain

possessed will require stakeholders to be online at least every k slots. The relevance of the rule

comes from the fact that as stake shifts over time, it will be feasible for the adversary to corrupt

stakeholders that used to possess a stake majority at some point without triggering Bad

1

/2−δ

and

thus any adversarial chains produced due to such an event should be rejected.

We are now ready to state the main result of the section that establishes that the π

DPOS

protocol

is a robust transaction ledger under the environmental conditions that we have assumed. Recall

that in the dynamic stake case we have to ensure that the adversary cannot exploit the way stake

changes over time and corrupt a set of stakeholders that will enable the control of the majority of

an elected committee of stakeholders in an epoch. In order to capture this dependency on stake

“shifts,” we introduce the following property.

Definition 5.2 (Stake shift). Consider two slots sl

1

, sl

2

and an execution E. The stake shift between

sl

1

and sl

2

is the maximum statistical distance, taken over all chains C adopted by an honest party,

of the two weighted-by-stake distributions that are defined by C[0 : sl

1

] and C[0 : sl

2

].

With this definition in place we are ready to state the main theorem.

Theorem 5.3. Fix parameters k, `, , σ, E, R, and L, where R ≥ 2(k + `) + E and L is an

integer multiple of R. Consider an execution E of π

iDPoS

with lifetime L coupled with a (1/2 − )-

initially-bounded adversary A with corruption delay D = R + E + 2(k + `) and beacon lookahead

E and environment Z exhibiting a stake-shift of no more than σ over any period of ` slots. Then

persistence, with parameter k, and liveness, with parameter u = 2(k+`), are violated with probability

no more than

CP

(k; L, ) +

HCG

(1/2, 2k; L, ) +

∃CQ

(`; L, ) +

H

≤ L ·

[

exp(−Ω(

√

k)) + O(exp(−2

2

`))

]

+

H

.

Here

H

denotes the probability of a collision occurring among the queries to H (including those of

A). The above probabilities are in the conditional space ¬Bad

1

/2−−σ

.

Proof. Consider an execution E generated by π

iDPoS

with a (1/2 − )-initially-bounded adversary A

with corruption delay D = R+E +2(k +`) and environment Z. We implicitly condition throughout

on ¬Bad

1

/2−−σ

by simply working in this conditioned probability space; we furthermore condition

on the event that there are no collisions among the queries made to the hash function by all

participants and the adversary: rather than adjusting the ambient probability space for this second

conditioning, we introduce an error term

H

into our security guarantees. In terms of the random

variable E, we begin by defining several other random variables. The most important family of

random variables capture the high-level chain properties of the protocol:

Chain guarantees. For each t, Chain

t

is the event that, over the course of epochs numbered 1

through t,

• CP is satisfied with parameter k,

• ∃CQ is satisfied with parameter `, and

• HCG is satisfied with parameters (1/2, 2k).

40

We remark, by reasoning parallel to that of Theorem 4.32, that these properties directly imply

CG with parameters (k/(2k + 2`), 2(k + `)). (In particular, this guarantees growth of k blocks

over any period of 2(k + `) slots.)

For convenience, we adopt the convention that Chain

0

occurs by fiat.

We continue to define a number of ancillary random variables which only take on relevant values

in case Chain

t

occurs.

Instantaneous common prefix. When Chain

t

occurs, the chains adopted by the family of honest

players at the outset of slot tR share a common prefix through slot tR − 2(k + `); we let C

(t)

denote this common chain. This follows directly from the chain properties discussed above,

which guarantee k-CP and growth of k blocks over any window of 2(k + `) slots.

The analysis below critically relies on the property that C

(t)

is immutable: specifically, when

Chain

t

occurs all honest participants agree on C

(t)

and, as maxvalid

k

can only revise the last

k blocks of a currently adopted chain, C

(t)

will be a prefix of all future chains held by the

honest parties.

If Chain

t

does not occur, we define C

(t)

= ⊥.

Instantaneous stake distribution; instantaneous characteristic string. S

1

is defined by the

genesis block. For each t > 1, we define S

t

to be the stake distribution determined by C

(t−1)

if Chain

t−1

occurs; otherwise, we set S

t

= ⊥.

For each t, we define a random variable w

(t)

∈ {0, 1}

R

∪ {⊥

R

} so that:

• If Chain

t−1

occurs, then w

(t)

is the characteristic string associated with epoch t. More

precisely, this is the characteristic string defined by (i) the leader election schedule deter-

mined by stake distribution S

t

and the beacon ρ

t

, and (ii) the set of parties corrupted, or

selected for corruption, by the adversary as of slot R(t − 1) − E. (As we work with delay

D = R + E + 2(k + `), this includes all parties that will be under adversarial control at

any point during epoch t or the 2(k + `) slots following this epoch.)

• If Chain

t−1

does not occur, w

(t)

= ⊥ · · · ⊥

︸

︷︷ ︸

R

.

• For later convenience, we also identify the random variables Hon

t

which indicate if a

majority of slots in epoch t are honest (which is to say that the Hamming weight of

w

(t)

is strictly less than R/2). Specifically, we define Hon

t

= ⊥ if w

(t)

= ⊥, Hon

t

= 1 if

∑

w

(t)

i

< R/2, and Hon

t

= 0 otherwise.

We let w

(<t)

denote the concatenation w

(1)

· · · w

(t−1)

.

We use the word “instantaneous” above to emphasize that these random variables are defined by

the state of the protocol at a particular time slot, and that the various features they describe (e.g.,

stake distributions, chains, etc.) evolve over the future course of the protocol.

We note some critical properties of these random variables.

First of all, considering that Chain

t

⇒ Chain

s

for all s ≤ t, the occurrence of Chain

t

implies that

a variety of protocol data established during these epochs persists through the rest of the protocol.

Specifically, C

(1)

· · · C

(t)

and, more generally, each C

(s)

is a prefix of the chains adopted by all

honest parties during epochs s + 1, s + 2, . . .. As a result, the honest parties unanimously agree on

the stake distribution S

s

, for each 1 ≤ s ≤ t + 1, both at the end of the epoch in which they are

defined and throughout the rest of the protocol.

41

Continuing to discuss the ramifications of Chain

t

, the fact that the honest parties agree on

S

1

, . . . , S

t+1

and, of course, agree on the beacon values, yields persistent agreement on the leader

schedules for the first t + 1 epochs. As a matter of analysis, we remark that Chain

t

thus unambigu-

ously defines the characteristic string w = w

(1)

. . . w

(t+1)

and a fork F

t+1

A

` w associated with all

chains adopted by honest parties. Note, in particular, the graph structure of F

t+1

A

is simply defined

by the unambiguous leader schedules, the structure of the chains held by honest players, the fact

that no collisions are observed by the hash function, and the ideal signature scheme. To check

that F

t+1

A

` w, it suffices to ensure that any slot behaving adversarially vis-a-vis the structure of

the fork—for example, violating the longest chain rule or generating multiple blocks—is correctly

reflected by the definition of w. Note, however, that a block signed by the leader of slot i (and

associated with slot number i) can only be adopted by an honest player during the next 2(k + `)

slots, as this is sufficient time for any player to accumulate a chain of length k beyond which no

alteration to slot i is possible. As w surely identifies a slot as adversarial if the leader will be under

adversarial control during the following 2(k + `) slots, this definition of w supports the fork F

t+1

A

,

as desired.

We shift our attention to the distribution determined on w. Conditioned on Chain

t

, as noted

above, the stake distribution S

t+1

is unambiguously determined by C

(t)

[0 : Rt − 2(k + `) − E] (cf.

part (a) of the Chain Extension step of Figure 10). As of slot Rt − E, when the beacon is exposed

to the adversary, this chain is stable in the view of all honest parties; that is—applying `-∃CQ and

(1/2, 2k)-HCG as above—any honest party has added at least k blocks to the chain so that it will

exist as a prefix of all honest parties’ chains for the remainder of the protocol. It follows that the

beacon value ρ

t+1

is independent of S

t+1

. Consider now the adversarial stake distribution indicated

by C[0 : Rt − E − 2(k + `)]. As the last block of this chain might not be honestly generated, we

cannot directly apply the guarantee provided by Bad

1

/2−−σ

to this distribution; however, in light

of `-∃CQ, there is an honestly generated block on C appearing no more than ` slots beyond the last

block of C[0 : Rt − E − 2(k + `)] so that, recalling the guarantee of σ stake shift over any ` slots,

the adversarial stake ratio given by C[0 : Rt − E − 2(k + `)] is no more than 1/2 − . It follows that

Pr[w

(t)

r

= 1 | w

(<t)

, w

(t)

1

, . . . , w

(t)

r−1

] ≤ 1/2 −

and that the fork F

t+1

A

determined by epoch t + 1 of the protocol has the property that F

t+1

A

`

w

(1)

. . . w

(t+1)

.

To complete the proof, consider the “virtual characteristic string” x obtained by substituting

all ⊥ in w = w

(1)

. . . w

(L/R)

with 0. Note that if Chain

t

fails for some t, the string w—and hence

the string x—must violate k-cp, `-∃cq, or (1/2, 2k)-HCG. Observe, also, that the string x, taking

values in {0, 1}

L

has the property that for each t

Pr[x

i

= 1 | x

1

, . . . , x

i−1

] ≤ 1/2 −

and hence, by Lemma 4.18, that the random variable x is stochastically dominated by the random

variable b = b

1

. . . , b

L

∈ {0, 1}

L

given by independently assigning each b

i

= 1 with probability

exactly 1/2 − . The event that a characteristic string violates k-cp, `-∃cq, or (1/2, 2s)-cg is clearly

monotone, as any violation is preserved by monotonically increasing the set of adversarial slots. We

remark, additionally, that if a characteristic string satisfies k-cp then, for any z ≥ 2k, the Hamming

weight of the string is less than z/2 over any interval of z slots (otherwise there is an immediate

k-cp violation). In light of Lemma 4.19, Lemma 4.22, and Theorem 4.31, it follows that

Pr[x violates k-cp, `-∃cq, or (1/2, 2k)-cg] ≤ Pr[b violates k-cp, `-∃cq, and (1/2, 2k)-cg]

≤ L

[

O(exp(−2

2

`) + exp(−Ω(

√

k))

]

+

H

.

42

To conclude, observe that if x satisfies k-cp, s-∃cq, and (1/2, 2s)-cg, then Chain

t

holds for all t and

hence w = x. In light of the comment above, in this case Hon

t

is also satisfied for all t. It follows,

in this case, that E satisfies k-CP, `-∃CQ, and (1/2, 2k)-CG, as desired.

These properties imply persistence with parameter k and liveness with parameter u = 2(k + `).

Liveness and persistence for π

DPoS

. We observe now that Proposition 5.1 and Theorem 5.3

can be combined to show the security of protocol π

DPoS

in the setting of a trusted beacon: in

particular, if there is an attack against persistence or liveness against π

DPoS

, it can be transformed

to an attack against π

iDPoS

. It follows that π

DPoS

possesses the same liveness and persistence

properties as π

iDPoS

, with an extra error term to account for failure of the signature scheme. We

omit further details as we turn our focus to the final protocol, where we show how the beacon can

be implemented.

5.2 Simulating a Trusted Beacon

While protocol π

DPoS

handles multiple epochs and takes into consideration changes in the stake

distribution, it still relies on F

DLS

to perform the leader selection process. In this section, we show

how to remove the dependency to F

DLS

through a Protocol π

DLS

, which allows the stakeholders to

compute the randomness and auxiliary information necessary in the leader election. The resulting

modified protocol that we will denote Π[π

DPoS

, π

DLS

] operates in the F

LS

hybrid world.

Recall that the only essential difference between F

LS

and F

DLS

is the continuous generation of

random strings ρ

2

, ρ

3

, . . . for epochs e

2

, e

3

, . . .. The idea is simple: protocol π

DLS

will use a coin

tossing protocol to generate unbiased randomness that can be used to define the values ρ

j

, j ≥ 2

bootstrapping on the initial random string and initial honest stakeholder distribution. However,

notice that the adversary could cause a simple coin tossing protocol to fail by aborting. Thus, we

build a coin tossing scheme with “guaranteed output delivery.”

Protocol π

DLS

is described in Figure 12 and uses a publicly verifiable secret sharing (PVSS) [44].

The resulting combined protocol Π[π

DPoS

, π

DLS

] operates as π

DPoS

while it runs π

DLS

to support the

random beacon generation for each epoch. For simplicity we will describe the combined protocol in

the random oracle model, i.e., with access to a random oracle functionality F

RO

; nevertheless, it is

also possible to achieve alternative realisations that do not depend on the random oracle abstraction.

We will take advantage of a simulation argument that will exploit the fact that the PVSS scheme and

the coin-flipping protocol built on top of it simulates a perfect beacon with negligible distinguishing

advantage. Simulation here suggests that, in the case of honest majority, there is a simulator that

interacts with the adversary and produces indistinguishable protocol transcripts when given the

beacon value after the commitment stage. While describing such a simulator is outside the scope of

our exposition, such a simulator can be inferred from the PVSS schemes of [44] or [18] realized in

the random oracle model where one can take advantage of programmability of the oracle. We note

that a random oracle is by no means necessary and it is possible to derive a simulator by taking

advantage of a “common reference string” (CRS) embedded into the genesis block.

Commitments and Coin Tossing. A coin tossing protocol allows two or more parties to obtain

a uniformly random string. A classic approach to construct such a protocol is by using commitment

schemes. In a commitment scheme, a committer carries out a commitment phase, which sends

evidence of a given value to a receiver without revealing it; later on, in an opening phase, the

committer can send that value to the receiver and convince it that the value is identical to the

value committed to in the commitment phase. Such a scheme is called binding if it is hard for

43

the committer to convince the receiver that he was committed to any value other than the one for

which he sent evidence in the commitment phase, and it is called hiding if it is hard for the receiver

to learn anything about the value before the opening phase. We denote the commitment phase

with randomness r and message m by Com(r, m) and the opening as Open(r, m).

In a standard two-party coin tossing protocol [11], one party starts by sampling a uniformly

random string u

1

and sending Com(r, u

1

). Next, the other party sends another uniformly random

string u

2

in the clear. Finally, the first party opens u

1

by sending Open(r, u

1

,) and both parties

compute output u = u

1

⊕ u

2

. Note, however, that in this classical protocol the committer may

selectively choose to “abort” the protocol (by not opening the commitment) once he observes the

value u

2

. While this is an intrinsic problem of the two-party setting, we can avoid this problem

in the multi-party setting by relying on a verifiable secret sharing scheme and an honest majority

amongst the protocol participants.

Publicly Verifiable Secret Sharing (VSS). A secret sharing scheme allows a dealer P

D

to

split a secret σ into n shares distributed to parties P

1

, . . . , P

n

, such that no adversary corrupting up

to t parties can recover σ. In a Verifiable Secret Sharing (VSS) scheme [26], there is the additional

guarantee that the honest parties can recover σ even if the adversary corrupts the shares held by

the parties that it controls and even if the dealer itself is malicious. We define a VSS scheme as a

pair of efficient dealing and reconstruction algorithms (Deal, Rec). We are only interested in VSS

where the secret is a random string. The dealing algorithm Deal(t, n) takes as input the number of

shares to be generated n as well as a parameter t ≤ n and outputs shares σ

1

, . . . , σ

n

of a random

value σ. The reconstruction algorithm Rec takes as input shares σ

1

, . . . , σ

n

and outputs the secret

σ as long as no more than t shares are corrupted (unavailable shares are set to ⊥ and considered

corrupted). A publicly verifiable secret sharing scheme allows any third party to verify the validity

of the shares in a non-interactive way without learning the shares themselves. This is achieved by

publishing auxiliary verification information that can later be attested to correspond to the shares

when those are revealed. Specifically, there is a Setup algorithm which is assumed to be executed

honestly, a key generation algorithm KeyGen that is executed by each shareholder resulting in a

secret and a public-key, as well as having an encryption for each σ

i

, denoted by β

i

, from which σ

i

is

recoverable via the secret-key of the i-th stakeholder by running algorithm Decrypt. Finally, a Verify

algorithm is capable of verifying all encrypted shares using the public-key and setup information.

We refer to the discrete logarithm based PVSS of [44] or [18] for more details.

Constructing Protocol π

DLS

. The main problem to be solved when realizing F

DLS

with a

protocol run by the stakeholders is that of generating uniform randomness for the leader selection

process while tolerating adversaries that may try to interfere by aborting or feeding incorrect

information to parties. In order to generate uniform randomness ρ

j

for epoch e

j

, j ≥ 2, the elected

stakeholders for epoch e

j−1

will employ a coin tossing scheme for which all honest parties are

guaranteed to receive output as long as there is an honest majority. The protocol has two stages,

commit and reveal, and it employs two parameters k, `. The stages of the protocol are presented

in Figure 11.

The commitment phase, consisting of 2k + 3` slots, proceeds as follows: for 1 ≤ i ≤ R, stake-

holder U

i

engages in PVSS by executing Deal(R, R/2) and posts the output shares to the blockchain.

Later, 2k + 3` slots after the beginning of the commitment phase, players check whether their

blockchain up to slot ` of the commitment phase contains proper PVSS shares from at least R/2+1

of the selected stakeholders; this requires running the PVSS verification for each commitment.

Assuming this is the case, the reveal stage commences. In the reveal stage, which lasts for `

44

CG

∃CQ

` slots

. . .. . .. . .

Commit

∃CQ

` slots

. . .. . .. . .

HCG

2k slots

. . .. . .. . .

Synchronize

∃CQ

` slots

. . .. . .. . .

∃CQ

` slots

. . .. . .. . .

Reveal

Stake distribution deadline

Figure 11: The stages of the protocol π

DPoS

that use the blockchain as a broadcast channel in an

epoch of R ≥ 2k + 4` slots.

slots, for 1 ≤ i ≤ R, stakeholder U

i

reveals the share it received for each commitment, cf. Protocol

π

DLS

in Figure 12. We remark that it is possible, at the expense of expanding the length of an

epoch somewhat more, to run a more efficient opening stage first and then “force-open” only those

commitments that were kept private. Given that, optimistically, no force-open will be required, the

overall communication complexity in this case would improve.

As a prelude to the next section, we will show how π

DLS

protocol implements the F

DLS

func-

tionality based on the F

LS

functionality and a “bulletin board” F

BB

. Specifically, the functionality

operates as follows: it is parameterized by k, ` and a stakeholder distribution S

0

. Whenever one of

the stakeholders submits a message m this is passed to the adversary and after ` slots it is reported

as pending to any party querying the bulletin board. Finally after 2(k + `) slots, all pending mes-

sages submitted before 2(k + `) slots are finalized in some slot selected by the adversary and are

reported to any party querying the functionality together with a slot they are finalised in, which

can be anywhere from 1 up to ` slots later than the the slot they were originally submitted. Below

we use F

BB

as a global functionality, cf. [17].

Proposition 5.4 (essentially from [44]). For each PPT A, Z it holds that there is a PPT S so that

EXEC

F

LS

,F

RO

,F

BB

π

DLS

,A,Z

(λ) and EXEC

F

DLS

,F

BB

S,Z

(λ)

are computationally indistinguishable under the Computational Diffie Hellman assumption where

the lookahead parameter of F

DLS

is set to E = `.

Proof. Fixing any A, the simulator S runs A and controls the honest parties’ postings with respect

to the underlying π

DLS

protocol following the round structure of the protocol. We recall here the

structure of the PVSS protocol of [44]. There are public parameters g, G and each party has a

registered public-key of the form y

i

= G

x

i

. To make a commitment each party picks a random

polynomial p(·) of degree d = R/2 with coefficients a

0

, a

1

, . . . , a

d

. It performs a commitment

by publishing the values C

0

= g

a

0

, . . . , C

d

= g

a

d

as well as Y

1

= y

p(1)

1

, . . . , Y

R

= y

p(R)

R

. Let

X

i

=

∏

d

j=0

C

i

j

j

= g

p(i)

. Using a proof of equality of discrete logarithms the party also publishes

for each i, a proof that log

g

(X

i

) = log

y

i

(Y

i

). A commitment is verified by checking all proofs of

discrete-log equality. Note that a commitment may be opened by publishing S = G

a

0

together

with a proof of equality of discrete logarithms log

G

(S) = log

g

(C

0

). The hash of the product of

opened commitments is the value of the next beacon. The parties “force” open the commitments

of each other by publishing the values: (i, S

i

) with S

i

= Y

x

−1

i

i

) together with a proof of equality of

discrete-logarithms log

S

i

(Y

i

) = log

Y

i

(G). Subsequently it is possible to recover each random value

by computing the Lagrange coefficients λ

1

, . . . , λ

d

with respect to the indexes that are available,

say i

1

, . . . , i

d

and compute S =

∏

d

j=1

S

λ

j

i

j

.

45

We next describe a simulated execution of the protocol that is based on a DDH tuple 〈g, G, C, D〉.

During the initial commit stage, S follows the protocol of each honest party modifying the com-

mitment C

0

to be computed as C

a

0

as opposed to g

a

0

. For simplicity, assume that the adversarial

parties are 1, . . . , R/2. In the simulated execution, the public-keys y

R/2+1

, . . . , y

R

are selected as

powers of g. The polynomial values p(1), . . . , p(R/2) are selected on behalf of the honest party

at random. Subsequently, the share encryptions y

p(1)

1

, . . . , y

p(R/2)

R/2

are calculated. Note that the

values p(R/2 + 1), . . . , p(R) cannot be evaluated directly since the value p(0) is determined by

log

g

(C

0

) = a

0

· log

g

C which is unknown to the simulator. However it is possible for S to calculate

Y

R/2+1

= y

p(R/2+1)

R/2+1

, . . . , Y

R

= y

p(R)

R

by performing Lagrange interpolation and taking advantage of the fact that log

g

(y

i

) for i ∈

{R/2 + 1, . . . , R} is known. Specifically, the simulator can calculate g

p(R/2)+1

, . . . , g

p(R)

by a suit-

able Lagrange interpolation over the bases: g

p(1)

, . . . , g

p(R/2)

, C

a

0

= g

p(0)

and then, by raising them

to the corresponding honest party secret-keys, the simulator obtains Y

R/2+1

, . . . , Y

R

. In a similar

manner the values C

1

= g

a

1

, . . . , C

d

= g

a

d

can be calculated by Lagrange interpolation with the

bases g

p(1)

, . . . , g

p(R/2)

, C

a

0

= g

p(0)

. All proofs performed by S on behalf of the honest party will

be simulated taking advantage of random oracle programmability.

Observe that the PVSS opening for the honest party is equal to D

a

0

. S follows the above

strategy for all honest parties while it receives all the PVSS values contributed by the malicious

party. At the conclusion of the 2(k + `) round, S queries F

DLS

to obtain the beacon value of the

next epoch. At this moment S is ready to program the random oracle so that the next epoch is

fixed to the correct beacon value; we call this the critical moment of the simulation. Note that

programmability will be achievable provided that the value D

a

0

for a PVSS of an honest party can

be set to a desired random value; if this value is already defined then S fails. Assuming the failure

event does not happen, given that S controls a majority of parties, it is capable of extracting the

PVSS value contributed by each malicious party; note that this requires the proper secret-key of

the honest party, which in turn requires log

g

(G).

Next we argue that the failure event happens with negligible probability. For the sake of

contradiction suppose that it happens with some probability α which is non-negligible. We perform

the simulation as above setting the values 〈g, G, C〉 to a given instance of the CDH problem for a

random honest party. Subsequently at the critical moment of the simulation we terminate returning

as output the value X

1/a

0

where X is a random input value in the random oracle table. Given that

the failure event happens with probability α, we conclude that this algorithm breaks CDH with

probability α/(Rq) where is q is the number of queries posed to the random oracle.

5.3 Robust Transaction Ledger

The key idea is to combine the π

DLS

protocol and its ability to simulate F

DLS

as shown in Proposi-

tion 5.4 with the recursive argument of Theorem 5.3 observing that the blockchain itself can play

the role of the F

BB

that π

DLS

requires. The resulting combined protocol Π[π

DPoS

, π

DLS

] executes

both operations of π

DPoS

and π

DLS

in this order in each slot while it queues any messages to be

transmitted for diffusion by either protocol and transmits them at the end of each slot.

Theorem 5.5. Fix parameters k, `, E, , σ, R, and L, where R ≥ 2k + 4` and L is an integer

multiple of R. Consider an execution E of Π[π

DPoS

, π

DLS

] with lifetime L coupled with a (1/2 − )-

initially-bounded adversary A with corruption delay D = 2R and environment Z exhibiting a stake-

46

Protocol π

DLS

with parameter k, `.

π

DLS

is a protocol run by a subset of elected stakeholders each one corresponding to a slot during an

epoch e

j

that lasts R ≥ 2k + 4` slots. The stakeholders without loss of generality are denoted by

U

1

, . . . , U

R

and they are not necessarily distinct.

Precondition. We assume that the Setup algorithm for the PVSS has been executed and its output

is posted in F

BB

as well as each stakeholder U

i

has performed KeyGen to obtain y

i

, s

i

and posted the

public-key y

i

to F

BB

. (If some of the stakeholders’ keys are missing the same protocol is executed with

R adjusted accordingly).

1. Commitment Phase (2k + 3` slots) At slot R − 2k − 4` of epoch e

j

, for 1 ≤ i ≤ R, stakeholder

U

i

performs Deal(R, R/2, y

1

, . . . , y

R

) to obtain the encryptions β

1

, . . . , β

n

and posts them to F

BB

.

2. Reveal Phase (` slots) At slot R −` of epoch e

j

, for 1 ≤ i ≤ R, U

i

runs Decrypt(s

i

, β

li

) to recover

the encrypted shares σ

li

for any U

l

that successfully posted in F

BB

R encrypted shares that verify

correctly (i.e., pass Verify) and were finalized in F

BB

during the first ` slots of the commitment

phase. Subsequently it posts σ

li

to the F

BB

.

The implementation of epochrnd req

is then as follows.

• Given input (epochrnd req

, U

i

, e

j

) , the values ρ

j

l

are calculated to be equal to H(Rec(σ

l1

, . . . , σ

lR

))

assuming that U

l

posted to F

BB

all shares correctly (otherwise ρ

j

l

is set to 0). Finally the epoch

randomness is set to ρ

j

= ⊕

R

l=1

ρ

j

l

. The responce is then set to (epochrnd, ρ

j

).

Figure 12: Protocol π

DLS

using F

LS

, F

BB

, F

RO

and an underlying PVSS scheme.

shift of σ over ` slots. Then persistence, with parameter k, and liveness, with parameter u = 2(k+`),

are violated with probability no more than

ε := L ·

[

exp(−Ω(

√

k)) + O(exp(−2

2

`))

]

+

H

+ (L/R)

DLS

+

DSIG

.

Here

H

denotes the probability of a collision occurring among the queries to H (including those of

A),

DSIG

is the distinguishing advantage of the digital signature implementation,

DLS

is the dis-

tance of the DSIG implementation. The above probabilities are in the conditional space ¬Bad

1

/2−−σ

.

Proof. Fix some environment Z and adversary A. As before we first consider the execution E

of Π[π

iDPoS

, π

DLS

]. We consider now a modified execution E

∗

where the execution of the π

DLS

protocol is substituted by the simulation that is provided due to Proposition 5.4. In this modified

execution E

∗

we define the same set of random variables Chain

∗

t

and C

∗(t)

, Hon

∗

t

for each epoch t as

in the proof of Theorem 5.3. The stakeholder distribution for each beacon simulation that samples

the beacon for the next epoch is drawn from the immutable chain C

∗(t)

.

The simulation might stop for two reasons. First it might be the case that the chain C

∗(t)

= ⊥,

i.e., the immutable chain of the stakeholders is not defined and hence no stakeholder distribution

can be determined to execute the beacon protocol. The second case where it might fail is because

of a dishonest majority among the R selected stakeholders, i.e., Hon

t

= 0.

We now observe that in the modified execution E

∗

the proof of Theorem 5.3 can be repeated

identically and we can infer that the probability of B

∗

, the event that persistence or liveness is

violated or the simulation fails in E

∗

, is at most

L ·

[

exp(−Ω(

√

k)) + O(exp(−2

2

`))

]

+

H

.

Now consider the event B that either persistence or liveness is violated in an execution E of the

protocol Π[π

iDPoS

, π

DLS

] and let ˜t be the index of the earliest epoch that this takes place (or ˜t = ∞

47

if the event never happens). We define the intermediate hybrid distributions between E and E

∗

denoted by E

(t)

for which it holds that the beacon generation in the first t epochs is performed in

simulation as in E

∗

while the remaining are performed as in E by performing π

DLS

. Observe that

E

(0)

= E and E

(L/R)

= E

∗

. Furthermore, conditional on 1 ≤ t <

˜

t, it holds that the computational

distance between E

(t−1)

and E

(t)

is at most

DLS

. It follows that the probability of B is bounded

by the probability of B

∗

plus (L/R) ·

DLS

. As a result, persistence and liveness are violated in E

with probability at most

[

exp(−Ω(

√

k)) + O(exp(−2

2

`))

]

+

H

+ (L/R) ·

DLS

.

Armed with this result, and using proposition 5.1 we can infer that there is a simulator S

1

that

idealises the digital signature used in the protocol and acts as the attacker in an execution E.

∣

∣

∣

Pr

[

EXEC

F

LS

[SIG],F

RO

Π[π

DPoS

,π

DLS

],A,Z

(λ)

]

− Pr

[

EXEC

F

LS

[F

DSIG

],F

RO

Π[π

iDPoS

,π

DLS

],S

1

,Z

(λ)

]∣

∣

∣

≤

DSIG

.

From the above it follows immediately that persistence and liveness are violated in an execution of

Π[π

DPoS

, π

DLS

] with the probability stated in the theorem.

Remark 2. We note that it is easy to extend the adversarial model to include fail-stop (and recover)

corruptions in addition to Byzantine corruptions. The advantage of this mixed corruption setting,

is that it is feasible to prove that we can tolerate a large number of fail-stop corruptions (arbitrarily

above 50%). The intuition behind this is simple: the forkable string analysis still applies even if

an arbitrary percentage of slot leaders is rendered inactive. The only necessary provision for this

would be expand the parameter k inverse proportionally to the rate of non-stopped parties; if the

rate is constant asymptotically the same analysis would apply. For a further investigation in this

direction see the follow up work, Ouroboros Genesis [4].

6 Covert Adversaries

The general notion of fork defined in Definition 4.11 above reflects the possibility that adversarial

slot leaders may broadcast multiple blocks for a single slot; such adversaries may simultaneously

extend many different chains. While this provides the adversary significant opportunities to in-

terfere with the protocol, it leaves a suspicious “audit trail”—multiple signed blocks for the same

slot—which conspicuously deviates from the protocol.

This motivates our consideration of a restricted class of covert adversaries, who broadcast no

more than one block per slot. Such an adversary may still deviate from the protocol by extend-

ing short chains, but does not produce such suspicious evidence and hence its strategy is more

“deniable”: it can blame network delays for its actions.

6

6.1 Covert Forks, and Covertly Forkable Strings

Such an adversary yields a restricted notion of fork, defined below:

Definition 6.1. Let F ` w be a fork for a string w ∈ {0, 1}

∗

. We say that F is covert if the

labeling ` : V → {0, 1, . . .} is injective. In particular, no adversarial index labels more than one

node.

6

Contrast this with a more general adversary that attempts to fork by signing two different blocks for the same

slot; such an adversary cannot merely blame the network for such a deviation.

48

As in the general case, we define a notion of forkable string for such adversaries.

Definition 6.2. We say that a string w is covertly forkable if there is a flat covert fork F ` w.

Covert adversaries and forks have much simpler structure than general adversaries. In particu-

lar, a string is covertly forkable if and only if a majority of its indices are adversarial. This provides

an analogue of Proposition 4.27 for covertly forkable strings.

Proposition 6.3. A string w ∈ {0, 1}

n

is covertly forkable if and only if wt(w) ≥ n/2.

Proof. Let w be a covertly forkable string and F ` w a flat covert fork. As F is flat, there are

two edge disjoint tines, t

1

and t

2

, with length equal to height(F ) and it follows that the number

of vertices in F is at least 2 · height(F ) + 1. In this covert case the labeling function is injective,

and it follows that n ≥ 2 · height(F ). (Recall that the root vertex is labeled by 0, which is not an

index into w.) On the other hand, the height of F is at least the number of honest indices of w.

We conclude that the length of w is at least twice the number of honest indices, as desired.

If wt(w) ≥ n/2, we can produce a flat covert fork F ` w by placing all honest indices on a

common tine t

1

and selecting length(t

1

) adversarial indices to form an edge-disjoint second tine

t

2

.

As the structure of covertly forkable strings is so simple, an analogue of Theorem 4.24 for the

density of covertly forkable strings follows directly from standard large deviation bounds.

Theorem 6.4. Let ∈ (0, 1) and let w be a string drawn from {0, 1}

n

by independently assigning

each w

i

= 1 with probability 1/2 − . Then

Pr[w is covertly forkable] = 2

−Θ(

2

n)

.

Proof. This follows from standard estimates for the cumulative density function of the binomial

distribution.

Exact probabilities of covert forkability for explicit values of n. For comparison with

the general case, we computed the probability that a string drawn from the binomial distribution

is covertly forkable. These results are presented in Figure 13. (Note that these probabilities are

simply appropriate evaluations of the cumulative density function of the binomial distribution.)

Analogous results for the general case appeared in Figure 8.

6.2 Common Prefix with Covert Adversaries

We revisit the notion of common prefix in the setting of covert adversaries. We define the covert

divergence of w to be the maximum divergence over all possible covert forks for w:

cdiv(w) = max

F `w

F covert

div(F ) .

As in the setting with general adversaries, we wish to establish that a string with large covert

divergence must have a large covertly forkable substring. A direct analogue of Theorem 4.31 then

implies that characteristic strings arising from π

iSPoS

are unlikely to have large covert divergence

and, hence, possess the common prefix property against covert adversaries.

We record an analogue of Theorem 4.30 for covert adversaries.

49

0.4 0.42 0.44 0.46 0.48 0.5

0

0.2

0.4

Binomial distribution parameter

Probability of covert forkability

Probability of Covert Forkability

n = 500

n = 1000

n = 1500

n = 2000

Figure 13: Graphs of the probability that a string drawn from the binomial distribution is

covertly forkable. Graphs for string lengths n = 500, 1000, 1500, 2000 are shown with parameters

.40, .41, . . . , .49, .50.

Theorem 6.5. Let w ∈ {0, 1}

∗

. Then there is a covertly forkable substring ˇw of w with | ˇw| ≥

cdiv(w).

Proof. We are more brief, as portions of the proof have direct analogs in the proof of Theorem 4.30.

Consider a covert fork F ` w and a pair of viable tines (t

1

, t

2

) of F for which `(t

1

) ≤ `(t

2

) and

t

1

/t

2

= cdiv(w); we assume, as in the proof of the general case, that this pair of tines minimizes

the quantity |`(t

2

) − `(t

1

)| among all pairs with divergence equal to cdiv(w).

Let y denote the last vertex on the tine t

1

∩ t

2

. In contrast to the setting with a general

adversary, it is not clear that y is honest and this motivates a slightly different choice for the

beginning of the string ˇw: define α to be the largest honest index of w on the tine t

1

∩ t

2

, with the

convention that α = 0 if there is no such index. As in the proof of Theorem 4.30, define β to be

the smallest honest index of w for which β ≥ `(t

2

), with the convention that β = n + 1 if there is

no such honest index. Then define ˇw = w

α+1

. . . w

β−1

; as in the proof of Theorem 4.30 it is easy

to confirm that | ˇw| = (β − 1) − α ≥ `(t

1

) − `(t

1

∩ t

2

) ≥ cdiv(w). The remainder of the proof argues

that ˇw is covertly forkable.

As in the proof of Theorem 4.30, the depth d(h) of any honest index h < β is no more

than min(length(t

1

), length(t

2

)): if h ≤ `(t

1

) this follows directly from the definition of viabil-

ity. Otherwise, `(t

1

) < h < `(t

2

) and we consider the tine t

h

labeled with h: if length(t

h

) ≥

min(length(t

1

), length(t

2

)) then the tine t

h

, coupled with either t

1

or t

2

, would produce a pair

of tines with divergence no less than div(t

1

, t

2

), but for which |`(·) − `(·)| is strictly less than

|`(t

1

) − `(t

2

)|.

To complete the proof, we define an injective function i : H → A, where H denotes the set of

honest indices in {α + 1, . . . , β − 1} and A the complement—the set of adversarial indices of ˇw. The

existence of such a function implies that |H| ≤ |A| and hence that ˇw is covertly forkable by the

criterion given in Proposition 6.3. Let A

′

⊂ A denote the set of adversarial indices of ˇw appearing

as a label on either of the two tines t

1

and t

2

. The function i is defined as follows: i(h), for an

honest index h ∈ H, is the smallest (adversarial) index of A

′

which labels a vertex at depth equal

50

to d(h). Assuming that this function is well-defined it is clearly injective, as labels cannot appear

on multiple vertices of a covert fork and depths of honest vertices are pairwise distinct.

To confirm that i(h) is well-defined, note that for any h ∈ H we must have d(α) < d(h) ≤

min(length(t

1

), length(t

2

)) and hence there is at least one vertex v on each of t

1

and t

2

with depth

equal to d(h); furthermore, by the defining properties of α and β, this vertex is labeled with

an index of ˇw. If d(h) ≤ length(t

1

∩ t

2

), there is a common vertex v on these tines for which

length(v) = d(h); note that this vertex cannot be honest by the definition of α, so i(h) = `(v) is

well-defined in this case. If d(h) > length(t

1

∩ t

2

), the two tines have distinct vertices at depth

d(h), and one of these must then be adversarial—thus i(h) is well-defined in this case as well.

Finally, we remark that the proof of Theorem 4.31 applies with minor adaptations to the covert

case.

Theorem 6.6. Let k, R ∈ N and ∈ (0, 1/2). Let w = w

1

. . . w

R

be drawn in {0, 1}

R

so that each

w

i

independently takes the value 1 with probability 1/2 − . Then

Pr[cdiv(w) ≥ k] ≤ R exp(−Ω(

2

k)) .

This directly bounds the probability that a covert adversay can effect a common prefix violation in

an execution of π

iSPoS

.

Proof. The proof of Theorem 4.31 applies directly; in this case the asymptotics rely on Theorem 6.4

and the fact that

∞

∑

t=k

e

−ct

≤

∫

∞

k−1

e

−ct

dt = O(1) · e

−ck

= e

−ck+O(1)

,

where the hidden constant may depend on c, but not k.

7 Anonymous Communication and Stronger Adversaries

The protocols constructed in the previous section are proven secure against delayed adaptive cor-

ruptions, meaning that, after requesting to corrupt a given party U

i

, the adversary has to wait for

D slots before the corruption actually happens. Naturally, it is desirable to make D as small as

possible, or even eliminate it altogether to achieve security against a standard adaptive adversary.

The delay is required because the adversary must not be able to corrupt parties in direct response

to knowledge of the leader election schedule. (Recall that the protocol determines this schedule

on an epoch-by-epoch basis, so that such an attack would be particularly devastating.) However,

notice that the slot leaders are selected by weighting public keys by stake, while the adversary

can only choose to corrupt a user U

i

without knowing its public key. Thus, the adversary must

be able to observe communication between U

i

and the Diffuse functionality in order to determine

which public key is associated with user U

i

and detect when U

i

is selected as a slot leader. We will

show that we can eliminate the delay by extending our model with a sender anonymous broadcast

channel (provided by the Diffuse functionality) and having the environment activate all parties in

every round. We introduce the following modifications in the ideal functionalities:

• Diffuse Functionality: The functionality will work as described in Section 2 except that it

will remove all information about the sender U

s

of every message before delivering it to the

receiver U

r

’s inbox (input tape), thus ensuring that the sender remain anonymous.

7

7

In practice, a sender anonymous broadcast channel with properties akin to those of the Diffuse functionality can

be implemented by Mix-networks [19] or DC-networks [20] that can be executed by the nodes running the protocol.

51

• Key and Transaction Functionality: The functionality will work as described in Section 2 ex-

cept that it will allow immediate corruption of a user U upon receiving a message (Corrupt, U )

from the adversary.

Apart from these modifications in the ideal functionalities, we also change the environment

behavior by requiring that it activates all users at every slot sl

j

. Having all parties being activated

at every slot results in an anonymity set of size equal to the number of honest parties, making it

difficult for the adversary to associate a given public key with a user (i.e., any of the honest parties

could be associated with a given public key that is not associated with a corrupted party). In this

extended model we can reprove Theorem 5.5 without a delay D by strengthening the restrictions

that are imposed on the environment in the following way.

• We will say the adversary is restricted to less than

1

/2 − δ relative stake for windows of length

D if for all sets of consecutive slots of length D, the sum over all corrupted keys of the

maximum stake held by each key during this period of D slots (in any possible S

j

(r) where

U

j

is an honest party) is no more than 1/2 − δ of the minimum total stake during this period.

In case the above is violated an event Bad

1

/2−δ

D

becomes true for the given execution.

Using the above strengthened condition, we can remove the corruption delay requirement D in

Theorem 5.5 by assuming that Bad

1

/2−δ

is substituted with Bad

1

/2−δ

D

.

8 Incentives

So far our analysis has focused on the cryptographic setting where a set of honest players operate

in the presence of an adversary who may corrupt some of the players. In this section we consider

the setting of a coalition of rational players and their incentives to deviate from honest protocol

operation.

8.1 Input Endorsers

In order to address incentives, we modify further our basic protocol to assign two different roles

to stakeholders. As before, in each epoch there is a set of elected stakeholders that are the slot

leaders of the epoch responsible for issuing blocks and forming the randomness of the next epoch.

Together with those there is a (not necessarily disjoint) set of stakeholders called the endorsers.

Now each slot has two types of stakeholders associated with it: the slot leader who will issue the

block as before and the slot endorser who will endorse the input to be included in the next slot.

(We remark that one can adapt this discussion to a setting with multiple endorsers; however, we

assume a single input endorser per slot in this description.) While this seems like an insignificant

modification it gives us a room for improvement for the following reason: endorsers’ contributions

will be acceptable even if they are d slots late, where d ∈ N is a parameter of the protocol. (In

particular, the protocol calls for slot leaders to include in their block any inputs endorsed by the

previous d endorsers and not appearing in the existing chain.) Note that blocks and endorsed

inputs are diffused independently with each block containing from 0 up to d endorsed inputs.

Note that in case no valid endorser input is available when the slot leader is about to issue

the block, the leader will go ahead and issue an empty block, i.e., a block without any actual

inputs (e.g., transactions in the case of a transaction ledger). Note that slot endorsers—just like

slot leaders—are selected by stake weight and are thus a representative sample of the stakeholder

population. In the case of a transaction ledger, the same transaction may be included by many

52

input endorsers simultaneously. In case that a transaction is multiply present in the blockchain

its first occurrence only will be its “canonical” position in the ledger. The enhanced protocol,

π

DPOSwE

, can be easily seen to have the same persistence and liveness behaviour as π

DPOS

: the

modification with endorsers does not provide any possibility for the adversary to prevent the chain

from growing, accepting inputs, or being consistent. However, if we measure chain quality in terms

of number of endorsed inputs included this produces a more favorable result: it is easy to see that

the number of endorsed inputs originating from a set of stakeholders S in any k-long portion of

the chain is proportional to the relative stake of S with high probability. This stems from the fact

that it is sufficient that a single honest block is created for all the endorsed inputs of the last d

slots to be included in it. Assuming d ≥ ` (the ∃CQ parameter from previous sections), any set of

stakeholders S will be an endorser in a subset of the d slots with probability proportional to its

cumulative stake; the result follows.

As in bitcoin, stakeholders that issue blocks are incentivized to participate in the protocol

by collecting transaction fees. Contrary to bitcoin, of course, one does not need to incentivize

stakeholders to invest computational resources to issue blocks. Rather, availability and transaction

verification should be incentivized. Nevertheless, they have to be incentivized to be online often.

Any stakeholder, at minimum, must be online and operational in the following circumstances.

• In the slot prior to a slot she is the elected shareholder so that she queries the network and

obtains the currently longest blockchain as well as any endorsed inputs to include in the block.

• In the slot during which she is the elected shareholder so that she issues the block containing

the endorsed inputs.

• In a slot during the commit stage of an epoch where she is supposed to issue the PVSS

commitment of her random string.

• In a slot during the reveal stage of an epoch where she is supposed to issue the required

opening shares.

• In general, in sufficient frequency, to check whether she is an elected shareholder for the next

or current epoch.

• In a slot during which she is the elected input endorser so that she issues the endorsed input

(e.g., the set of transactions) that requires processing all available transactions and verifying

them.

In order to incentivize the above actions in the setting of a transaction ledger, fees can be

collected from those that issue transactions to be included in the ledger which can then be transfered

to the block issuers. In bitcoin, for instance, fees can be collected by the miner that produces a

block of transactions as a reward. In our setting, similarly, a reward can be given to the parties

that are issuing blocks and endorsing inputs. The reward mechanism does not have to be block

dependent as advocated in [38]. In our setting, it is possible to collect all fees of transactions

included in a sequence of blocks in a pool and then distribute that pool to all shareholders that

participated during these slots. For example, all input endorsers that were active may receive reward

proportional to the number of inputs they endorsed during a period of rounds (independently of

the actual number of transactions they endorsed). Other ways to distribute transaction fees are

also feasible (including the one that is used by bitcoin itself—even though the bitcoin method is

known to be vulnerable to attacks, e.g., the selfing-mining attack).

53

The reward mechanism that we will pair with input endorsers operates as follows. Let C be a

chain consisting of blocks B

0

, B

1

, . . .. Consider the sequence of blocks that cover the j-th epoch

denoted by B

1

, . . . , B

s

with timestamps in {jR + 1, . . . , (j + 1)R} that contain an r ≥ 0 sequence

of endorsed inputs that originate from the j-th epoch (some of them may be included as part of

the j + 1 epoch). We define the reward pool P

(j)

all

to be equal to the sum of the transaction fees

that are included in the endorsed inputs that correspond to the j-th epoch. If a transaction occurs

multiple times (as part of different endorsed inputs) or even in conflicting versions, only the first

occurrence of the transaction is taken into account (and is considered to be part of the ledger at

that position) in the calculation of P

(j)

all

, where the total order used is induced by the order the

endorsed inputs that are included in C. In the sequence of these blocks, we identify by L

1

, . . . , L

R

the slot leaders corresponding to the slots of the epoch and by E

1

, . . . , E

r

the input endorsers that

actively contributed the sequence of r endorsed inputs. Subsequently, the i-th stakeholder U

i

can

claim a reward up to the amount

(

β ·

|{j | U

i

= E

j

}|

r

+ (1 − β) ·

|{j | U

i

= L

j

}|

R

)

· P

(j)

all

,

where β ∈ [0, 1] is a parameter of the protocol. Claiming a reward is performed by issuing a

“coinbase” type of transaction at any point after 2(k + `) slots in a subsequent epoch to the one

that a reward is being claimed from.

Observe that the above reward mechanism has the following features: (i.) it rewards elected

committee members for just being committee members, independently of whether they issued a

block or not; (ii.) it rewards the input endorsers with the inputs that they have contributed; (iii.)

it rewards entities for epoch j after slot jR + 2(k + `).

We proceed to show that our system is a δ-Nash (approximate) equilibrium, cf. [35, Section

2.6.6]. Specifically, the theorem states that any coalition deviating from the protocol can add at

most an additive δ to its total rewards.

A technical difficulty in the above formulation is that the number of players, their relative stake,

as well as the rewards they receive are based on the transactions that are generated in the course of

the protocol execution itself. To simplify the analysis we will consider a setting where the number

of players is static, the stake they possess does not shift over time and the protocol has negligible

cost to be executed. We observe that the total rewards (and hence also utility by our assumption

on protocol costs) that any coalition V of honest players are able to extract from the execution

lasting L = tR + 2(k + `) + 1 slots, is equal to

R

V

(E) =

t

∑

j=1

P

(j)

all

(

β

IE

j

V

(E)

R

+ (1 − β)

SL

j

V

(E)

r

j

)

provided that E satisfies CP with parameter k, ∃CQ is satisfied with parameter `, and HCG is

satisfied with parameters (1/2, 2k)), and where r

j

is the total endorsed inputs emitted in the j-th

epoch (and possibly included at any time up to the first ` slots of epoch j + 1), P

(j)

all

is the reward

pool of epoch j, SL

j

V

(E) is the number of times a member of V was elected to be a slot leader

in epoch j and IE

j

V

(E) the number of times a member of V was selected to endorse an input in

epoch j. We set by convention the value of R

V

(E) to 0 when E is an execution where the basic

underlying properties of the blockchain fail (in particular CP with parameter k, ∃CQ is satisfied

with parameter `, and HCG is satisfied with parameters (1/2, 2k)). Finally, observe that the actual

rewards obtained by a set of rational players V in an execution E might be different from R

V

(E);

for instance, the coalition of V may never endorse a set of inputs in which case they will obtain a

smaller number of rewards.

54

We will establish the fact that our protocol is a δ-Nash equilibrium by proving that the coalition

V , even deviating from the proper protocol behavior, it cannot obtain utility that exceeds R

V

(E)+δ

for some suitable constant δ > 0.

Theorem 8.1. Fix any δ > 0 and polynomially related parameters k, `, λ; under the same condi-

tions and restrictions as in Theorem 5.5, the honest strategy in the protocol is a δ-Nash equilibrium

against any coalition of players represented as an adversary A, provided that the maximum total

rewards P

all

provided in all possible protocol executions is bounded by a polynomial in λ.

Proof. Consider a coalition of rational players V —restricted as in the statement of the theorem—

that engages in a protocol execution together with a number of other players that follow the protocol

faithfully for a total number of L epochs. We will show that any deviation from the protocol will

not result in substantially higher rewards for V . Observe that based on Theorem 5.5, no matter

the strategy of V , with probability 1 − ε for some function ε negligible in the security parameters

k, `, λ the protocol will enable all users to obtain the rewards they are entitled to as slot leaders and

input endorsers. The latter stems from the following. First, from liveness, at least one honest block

will be included every ` slots and hence, in each epoch, all input endorsers that follow the protocol

will have the opportunity to enter their endorsed inputs as many times they were elected to be.

Second, the rewards received will be proportional to the times each party is an input endorser and

issued a block successfully as well as equal to the number of times it is a slot leader. As a result,

except with error ε, the utility received by coalition V is equal to R

V

. It follows that player V has

expected utility at most E[R

V

] + εP

All

, where P

All

is the maximum amount of rewards produced

in any possible execution. The result follows by the assumption in the statement of the theorem

since εP

All

≤ δ for sufficiently large λ.

Remark 3. In the above theorem, for simplicity, we assumed that protocol costs are not affecting

the final utility (in essence this means that protocol costs are assumed to be negligible). Nevertheless,

it is straightforward to extend the proof to cover a setting where a negative term is introduced in

the payoff function for each player proportional to the number of times inputs are endorsed and the

number of messages transmitted for the beacon protocol. The proof would be resilient to these mod-

ifications because endorsed inputs and beacon protocol messages cannot be stifled by the adversary

and hence the reward function can be designed with suitable weights for such actions that offsets

their cost. Still note that the rewards provided are assumed to be “flat” for both slots and endorsed

inputs and thus the costs would also have to be flat. We leave for future work the investigation of

a more refined setting where costs and rewards are proportional to the actual computational steps

needed to verify transactions and issue blocks.

Remark 4. The reward function described, only considers the number of times an entity was an

input endorser without considering the amount of work that was put to verify the given transactions.

Furthermore it is not sensitive to whether a slot leader issued a block or not in its assigned time slot.

We next provide some context behind these choices. First suppose that slot leaders do not receive

a reward when they do not issue a block. It is easy to see that when all parties follow the protocol

the parties will receive the proportion from the reward pool that is associated to block issuance

roughly proportional to their stake. Nevertheless, a malicious coalition can easily increase the ratio

of these rewards by performing a block witholding attack (in this case this would amount to a selfish

mining attack). Given that this happens with non-negligible probability a straightforward definition

of R

V

(E) that respects this assignment is vulnerable to attack and hence a δ-Nash equilibrium

theorem cannot be shown. Next, we consider the case of extending the reward function so that input

endorsers that are rewarded based on the transactions they verify (as opposed to the flat reward

55

we considered in the above theorem). Special care is necessary to design this function. Indeed the

straightforward way to implement it, which is if the first input endorser to verify a transaction that

is part of the pool can make a higher claim for its fee, then there is a strategy for an adversary

to deviate from the protocol and improve its ratio of rewards: perform block withholding and/or

endorsed input censorship to remove endorsed inputs from the blockchain that originate to honest

parties. Then include the removed transactions in an endorsed input that will be transmitted in the

last possible opportunity. As before, given the attack, the natural way to define R

V

(E) is susceptible

to it and hence a δ-Nash equilibrium theorem cannot be shown. A possible direction for ameliorating

this problem is to share the transaction fee of a transaction between all the input endorsers that

endorsed it. This suggests the following modification to the protocol: whenever you are an input

endorser you should attempt to include all transactions that you have collected for a sequence of k

slots and retransmit your endorsed input in case it is removed from the main chain. We leave the

analysis of such class of reward mechanisms for future work.

9 Stake Delegation

As discussed in the previous section, stakeholders must be online in order to generate blocks when

they are selected as slot leaders. However, this might be unattractive to stakeholders with a small

stake in the system. Moreover, requiring that a majority of elected stakeholders participate in the

coin tossing protocol for refreshing randomness introduces a strain on the on the stakeholders and

the network, since it might require broadcasting and storing a large number of commitments and

shares.

We mitigate these issues by providing a method for reducing the size of the group of stakehold-

ers that engage in the coin tossing protocol. Instead of the elected stakeholders directly forming

the committee that will run coin tossing, a group of delegates will act on their behalf. In more

detail, we put forth a delegation scheme, whereby stakeholders will authorize other entities, called

delegates, who may be stakeholders themselves, to represent them in the coin tossing protocol. A

delegate may participate in the protocol only if it represents a certain number of stakeholders whose

aggregate stake exceeds a given threshold. Such a participation threshold ensures that a “fragmen-

tation” attack, that aims to increase the delegate population in order to hurt the performance of

the protocol, cannot incur a large penalty as it is capable to force the size of the committee that

runs the protocol to be small (it is worth noting that the delegation mechanism is similar to mining

pools in proof-of-work blockchain protocols).

9.1 Minimum Committee Size

To appreciate the benefits of delegation, recall that in the basic protocol (π

DPoS

) a committee

member selected by weighing by stake is honest with probability 1/2 + (this being the fraction

of the stake held by honest players). Thus, the number of honest players selected by k invocations

of weighing by stake is a binomial distribution. We are interested in the probability of a malicious

majority, which can be directly controlled by a Chernoff bound. Specifically, if we let Y be the

number of times that a malicious committee member is elected then

Pr[Y ≥ k/2] = Pr[Y ≥ (1 + δ)(1/2 − )k]

≤ exp(− min{δ

2

, δ}(1/2 − )k/4)

< exp(−δ

2

(1/2 − )k/4)

for δ = 2/(1 − 2). Assuming < 1/4, it follows that δ < 1.

56

Consider the case that = 0.05; then we have the bound exp(−0.00138 · k) which provides

an error of 1/1000 as long as k ≥ 5000. Similarly, in the case = 0.1, we have the bound

exp(−0.00625k) which provides the same error for k ≥ 1100.

We observe that in order to withstand a significant number of epochs, say 2

15

(which, if we

equate a period with one day, will be 88 years), and require error probability 2

−40

, we need that

k ≥ 32648.

In cases where the wealth in the system is not concentrated among a small set of stakeholders

the above choice is bound to create a very large committee. (Of course, the maximum size of the

committee is k.)

9.2 Delegation Scheme

The concept of delegation is simple: any stakeholder can allow a delegate to generate blocks on her

behalf. In the context of our protocol, where a slot leader signs the block it generates for a certain

slot, such a scheme can be implemented in a straightforward way based on proxy signatures [12].

A stakeholder can transfer the right to generate blocks by creating a proxy signing key that

allows the delegate to sign messages of the form (st, d, sl

j

) (i.e., the format of messages signed in

Protocol π

DPoS

to authenticate a block). In order to limit the delegate’s block generation power

to a certain range of epochs/slots, the stakeholder can limit the proxy signing key’s valid message

space to strings ending with a slot number sl

j

within a specific range of values. The delegate

can use a proxy signing key from a given stakeholder to simply run Protocol π

DPoS

on her behalf,

signing the blocks this stakeholder was elected to generate with the proxy signing key. This simple

scheme is secure due to the Verifiability and Prevention of Misuse properties of proxy signature

schemes, which ensure that any stakeholder can verify that a proxy signing key was actually issued

by a specific stakeholder to a specific delegate and that the delegate can only use these keys to

sign messages inside the key’s valid message space, respectively. We remark that while proxy

signatures can be described as a high level generic primitive, it is easy to construct such schemes

from standard digital signature schemes through delegation-by-proxy as shown in [12]. In this

construction, a stakeholder signs a certificate specifying the delegates identity (e.g., its public key)

and the valid message space. Later on, the delegate can sign messages within the valid message

space by providing signatures for these messages under its own public key along with the signed

certificate. As an added advantage, proxy signature schemes can also be built from aggregate

signatures in such a way that signatures generated under a proxy signing key have essentially the

same size as regular signatures [12].

An important consideration in the above setting is the fact that a stakeholder may want to

withdraw her support to a stakeholder prior to its proxy signing key expiration. Observe that proxy

signing keys can be uniquely identified and thus they may be revoked by a certificate revocation

list within the blockchain.

9.2.1 Eligibility threshold

Delegation as described above can ameliorate fragmentation that may occur in the stake distribu-

tion. Nevertheless, this does not prevent a malicious stakeholder from dividing its stake to multiple

accounts and, by refraining from delegation, induce a very large committee size. To address this,

as mentioned above, a threshold T , say 1%, may be applied. This means that any delegate repre-

senting less a fraction less than T of the total stake is automatically barred from being a committee

member. This can be facilitated by redistributing the voting rights of delegates representing less

than T to other delegates in a deterministic fashion (e.g., starting from those with the highest stake

57

and breaking ties according to lexicographic order). Suppose that a committee has been formed,

C

1

, . . . , C

m

, from a total of k draws of weighing by stake. Each committee member will hold k

i

such votes where

∑

m

i=1

k

i

= k. Based on the eligibility threshold above it follows that m ≤ T

−1

(the maximum value is the case when all stake is distributed in T

−1

delegates each holding T of

the stake).

10 Attacks Discussion

We next discuss a number of practical attacks and indicate how they are reflected by our modeling

and mitigated.

Double spending attacks In a double spending attack, the adversary wishes to revert a trans-

action that is confirmed by the network. The objective of the attack is to issue a transaction, e.g.,

a payment from an adversarial account holder to a victim recipient, have the transaction confirmed

and then revert the transaction by, e.g., including in the ledger a second conflicting transaction.

Such an attack is not feasible under the conditions of Theorem 5.5. Indeed, persistence ensures

that once the transaction is confirmed by an honest player, all other honest players from that point

on will never disagree regarding this transaction. Thus it will be impossible to bring the system to

a state where the confirmed transaction is invalidated (assuming all preconditions of the theorem

hold). See the next section for an experimental discussion about double spending.

Grinding attacks In stake grinding attacks, the adversary tries to influence the slot leader

selection process to improve its chances of being selected to generate blocks (which can be used to

perform other attacks such as double spending). Basically, when generating a block that is taken

as input by the slot leader selection process, the adversary first tests several possible block headers

and block contents in order to find the one that gives it the best chance of being selected as a

slot leader again in the future. While this attack affects PoS based cryptocurrencies that collect

randomness for the slot leader selection process from raw data in the blockchain itself (i.e. from

block headers and contents), our protocol uses a coin tossing protocol that is proven to generate

unbiased uniform randomness as discussed in Section 5.2. We show that an adversary cannot

influence the randomness generated in Figure 12, which is guaranteed to be uniformly random,

thus guaranteeing that slot leaders are selected with probability proportional to their stake.

Transaction denial (censorship) attacks In a transaction denial attack, the adversary wishes

to prevent a certain transaction from becoming confirmed. For instance, the adversary may want

to target a specific account and prevent the account holder from issuing an outgoing transaction.

Such an attack is not feasible under the conditions of Theorem 5.5. Indeed, liveness ensures that,

provided the transaction is attempted to be inserted for a sufficient number of slots by the network,

it will be eventually confirmed.

Desynchronization attacks In a desynchronization attack, a shareholder behaves honestly but

is nevertheless incapable of synchronizing correctly with the rest of the network. This leads to

ill-timed issuing of blocks and being offline during periods when the shareholder is supposed to

participate. Such an attack can be mounted by preventing the party’s access to a time server or

any other mechanism that allows synchronization between parties. Moreover, a desynchronization

may also occur due to exceedingly long delays in message delivery. Our model allows parties to

become desynchronized by incorporating them into the adversary. No guarantees of liveness and

58

persistence are provided for desynchronized parties and thus we can get security as long as parties

with less than 50% of stake get desynchronized. If more than 50% parties get desynchronized our

protocol can fail. More general models like partial synchrony [23, 39] are interesting to consider

in the PoS design setting. See the follow up work, Ouroboros Praos, for more information on this

topic [22].

Eclipse attacks In an eclipse attack, message delivery to a shareholder is violated due to a

subversion in the peer-to-peer message delivery mechanism. As in the case of desynchronization

attacks, our model allows parties to be eclipse attacked by incorporating them into the adversary.

No guarantees of liveness or persistence are provided for such parties.

51% attacks A 51% attack occurs whenever the adversary controls more than the majority of

the stake in the system. It is easy to see that any sequence of slots in such a case is with very high

probability forkable and thus once the system finds itself in such setting the honest stakeholders

may be placed in different forks for long periods of time. Both persistence and liveness can be

violated.

Bribery Attacks In bribery attacks [13], an adversary deliberately pays miners (through cryp-

tocurrency or fiat money) to work on specific blocks and forks, aiming at generating an arbitrary

fork that benefits the adversary (e.g. by supporting a double spending attack). Miners of PoW

based cryptocurrencies do not have to own any stake in order to mine blocks, which makes this

attack strategy feasible. In this setting, if the adversary offers a bribe higher than the reward

for correctly generating a block, any rational miner has a clear incentive to accept the bribe and

participate in the attack since it increases the miner’s financial outcome. However, in our PoS

based protocol, malicious slot leaders who agree to deliberately attack the system not only risk to

forego any potential profit they would earn from behaving honestly but may also risk to lose equity.

Notice that slot leaders must have money invested in the system in order to be able to generate

blocks and if an attack against the system is observed this might bring currency value down. Even

if the bribe is higher than the reward for correct behavior, the loss from currency devaluation can

easily offset any additional profits made by participating in this attack. Hence, bribery attacks may

be be less effective against a PoS based consensus protocol than a PoW based one. Currently our

rationality model does not formally encompass this attack strategy and investigating its efficacy

against PoS based consensus protocols is left as a future work.

Long-range attacks An attacker who wishes to double spend at a later point in time can mount

a long-range attack [14] by computing a longer valid chain that starts right after the genesis block

where it is the single stakeholder actively participating in the protocol. Even if this attacker

owns a small fraction of the total stake, it can locally compute this chain generating only the

blocks for slots where it is elected the slot leader and keep generating blocks ahead of current

time until its alternative chain has more blocks than the main chain. Now, the attacker can post

a transaction to the main chain, wait for it to be confirmed (and for goods to be delivered in

exchange for the transaction) and present the longer alternative chain to invalidate its previously

confirmed transaction. This attack is ineffective against Ouroboros for two reasons: Protocol π

DLS

will only output valid leader selection data allowing for the protocol to continue if a majority of

the stakeholders participate (or have delegates participate on their behalf) and stakeholders will

reject blocks generated for slots that are far ahead of time. Since the alternative chain is generated

artificially with blocks and protocol messages generated solely by an attacker who controls a small

59

fraction of the stake, the leader selection data needed to start new epochs will be considered invalid

by other nodes. Even if the attacker could find a strategy to generate an alternative chain with

valid leader selection data, presenting this chain and its blocks generated at slots that are far ahead

of time would not result in a successful attack since those blocks far ahead of time would be rejected

by the honest stakeholders and the final alternative chain would be shorter than the main chain.

Nothing at stake attacks The “nothing at stake” problem refers in general to attacks against

PoS blockchain systems that are facilitated by shareholders continuing simultaneously multiple

blockchains exploiting the fact that little computational effort is needed to build a PoS blockchain.

Provided that stakeholders are frequently online, nothing at stake is taken care of by our analysis

of forkable strings (even if the adversary brute-forces all possible strategies to fork the evolving

blockchain in the near future, there is none that is viable), and our chain selection rule that

instructs players to ignore very deep forks that deviate from the block they received the last time

they were online. It is also worth noting that, contrary to PoW-based blockchains, in our protocol

it is infeasible to have a fork generated in earnest by two shareholders. This is because slots are

uniquely assigned and thus at any given moment there is a single uniquely identified shareholder

that is elected to advance the blockchain. Players following the longest chain rule will adopt the

newly minted block (unless the adversary presents at that moment an alternative blockchain using

older blocks). It is remarked in [15] that the “tragedy of commons” might lead stakeholders in

some PoS based schemes to adhere to attacks because they do not have the power to deter attacks

by themselves and would incur financial losses even if they did not join the attack. This would

lead rational stakeholders to accept small bribes in alternative currencies that might at least obtain

some financial gain. However, in the incentive structure of Ouroboros, slot leaders and endorsers

who could potentially join an attack would receive rewards in both the main and the adversarial

chain, resulting in those stakeholders not achieving higher profits by joining the attack.

Past majority attacks As stake moves our assumption is that only the current majority of

stakeholders is honest. This means that past account keys (which potentially do not hold any stake

at present) may be compromised. This leads to a potential vulnerability for any PoS system since a

set of malicious shareholders from the past can build an alternative blockchain exploiting such old

accounts and the fact that it is effortless to build such a blockchain. In light of Theorem 5.5 such

attack can only occur against shareholders who are not frequently online to observe the evolution

of the system or in case the stake shifts are higher than what is anticipated by the preconditions

of the theorem. This can be seen a special instance of the nothing at stake problem, where the

attacker no longer owns any stake in the system and is thus free from any financial losses when

conducting the attack.

Selfish-mining In this type of attack, an attacker withholds blocks and releases them strategi-

cally attempting to drop honestly generated blocks from the main chain. In this way the attacker

reduces chain growth and increases the relative ratio of adversarially generated blocks. In conven-

tional reward schemes, as that of bitcoin, this has serious implications as it enables the attacker

to obtain a higher rate of rewards compared to the rewards it would be receiving in case it was

following the honest strategy. Using our reward mechanism however, selfish mining attacks are

neutralized. The intuition behind this, is that input endorsers, who are the entities that receive

rewards proportionally to their contributions, cannot be stifled because of block withholding: any

input endorser can have its contribution accepted for a sufficiently long period of time after its

endorsement took place, thus ensuring it will be incorporated into the blockchain (due to sufficient

60

chain quality and chain growth). Given that input endorsers’ contributions are (approximately)

proportional to their stake this ensures that reward distribution cannot be affected substantially

by block withholding.

11 Experimental Results

We have implemented a prototype instantiation of Ouroboros in Haskell as well as in the Rust-

based Parity Ethereum client in order to evaluate its concrete performance. More specifically, we

have implemented Protocol π

DPoS

using Protocol π

DLS

to generate leader selection parameters (i.e.,

generating fresh randomness for the weighed stake sampling procedure). For this instantiation, we

use the PVSS scheme of [44] implemented over the elliptic curve secp256r1. This PVSS scheme’s

share verification information includes a commitment to the secret, which is also used as the

commitment specified in protocol π

DLS

; this eliminates the need for a separate commitment to

be generated and stored in the blockchain. In order to obtain better efficiency, the final output ρ

of Protocol π

DLS

is a uniformly random binary string of 32 bytes. This string is then used as a

seed for a PRG (ChaCha in our implementation, [10]) and stretched into R random labels of log τ

bits corresponding to each slot in an epoch. The weighing by stake leader selection process is then

implemented by using the random binary string associated to each epoch to perform the sequence

of coin-flips for selecting a stakeholder. The signature scheme used for signing blocks is ECDSA,

also implemented over curve secp256r1.

11.1 Transaction Confirmation Time Under Optimal Network Conditions

We first examine the time required for confirming a transaction in a setting where the network is

not under substantial load and transactions are processed as they appear.

Adversary BTC OB Covert OB General

0.10 50 3 5

0.15 80 5 8

0.20 110 7 12

0.25 150 11 18

0.30 240 18 31

0.35 410 34 60

0.40 890 78 148

0.45 3400 317 663

Figure 14: Transaction confirmation times in minutes that achieve assurance 99.9% against a hypo-

thetical double spending attack with different levels of adversarial power for Bitcoin and Ouroboros

(both covert and general adversaries).

In Fig. 14 we lay out a comparison in terms of transaction confirmation time between Bitcoin

and Ouroboros showing how much a verifier has to wait to be sure that the best possible

8

double-

spending attack succeeds with probability less than 0.1%. In the case of Bitcoin, we consider a

double-spending attacker that commands a certain percentage of total hashing power and wishes

to revert a transaction. The attacker attempts to double-spend via a block-witholding attack as

described in the same paper (the attacker mines a private fork and releases it when it is long

8

The “best possible” is only in the the case of Ouroboros, for Bitcoin we use the best known attack.

61

enough). In the case of Ouroboros we consider a double spending attacker that attempts to brute

force the space of all possible forks for the current slot leader distribution in a certain segment of

the protocol and commands a certain percentage of the total stake. We consider both the covert

and the general adversarial setting for Ouroboros.

In all of the scenarios, we measure the number of minutes that one has to wait in order to achieve

probability of double spending less than 0.1%. In Fig. 15 we present a graph that illustrates the

speedup graphically.

0.1 0.2 0.3 0.4

4

6

8

10

12

14

16

Adversarial Strength of Blockwitholding Attacker

Speedup OB/BTC

Confirmation time speed up of Ouroboros over BTC

Covert

General

Figure 15: Ouroboros vs. Bitcoin speedup of transaction confirmation time against a hypothetical

double spending attacker for assurance level 99.9%. Ouroboros is at least 10 to 5 times faster for

regular adversaries and 16 to 10 times faster for covert adversaries.

We note that the above measurements compare our Ouroboros implementation with Bitcoin

in the way the two systems are parameterized (with 10 minute block production rate for Bitcoin

and 20 second slots for Ouroboros, a conservative parameter selection). Exploring alternative

parameterizations for Bitcoin (such as making the proof-of-work easier) can speed up the transaction

processing, nevertheless this cannot be done without carefully measuring the impact on overall

security.

11.2 Absolute Performance of Ouroboros

We implemented Ouroboros as an instance of the Rust-based Ethereum Parity client.

9

Subse-

quently, experiments were run using Amazon’s Elastic Compute Cloud (EC2) ‘c4.2xlarge‘ instances

in the ‘us-east-1‘ region with a smaller “runner” instance responsible for coordinating each of the

“worker” instances.

Each experiment consists of several steps:

1. Each worker instance builds a clean Docker image containing a specific revision of our fork of

9

Ethcore - Parity. https://ethcore.io/parity.html

62

the Parity software

10

containing the Ouroboros proof-of-concept changes based on the Parity

1.6.8 release.

2. Each worker instance is started in an “isolated” mode where none of the nodes talk to each

other. During this period, a Parity account is recovered on each node and a start time for

the network is established.

3. Each worker instance is restarted in a production mode that allows communication between

the nodes and transactions to be mined.

4. A single worker instance is informed about all the other nodes. All nodes become aware of

all other nodes via Parity’s peer-to-peer discovery methods.

5. Each worker instance has a number of transactions generated and ingested.

In each experiment, 650,000 total transactions are generated between the participating nodes

who shared stake equally. The amount transferred in any given transaction is small enough to

avoid any account running out of funds. Each instance generates all the transactions using a hard-

coded shared random seed, then keeps the transactions originating from the local user account. 20

transactions are saved in a single JSON file, ready to be directly passed to the Parity RPC endpoint

using the ‘curl‘ command line tool. During ingestion, a single file of 20 transactions is ingested

and one second is spent idle between each file to avoid overwhelming the instances with too many

requests.

Various setups were tested, focusing on adjusting the Ouroboros slot duration and the number

of participating nodes. 10, 20, 30, and 40 nodes were tested, ultimately limited by the number of

instances allowed in a single EC2 region. Slot durations of 5, 10, and 20 seconds were also tested.

Variance between experiments was small. In Figure 16 we present the case of 40 nodes and slot

length of 5 seconds that exhibits a median value of 257.6 transaction per second.

12 Acknowledgements

We thank Ioannis Konstantinou who contributed in a preliminary version of our protocol. We

thank Lars Br¨unjes, Duncan Coutts, and Kawin Worrasangasilpa for comments on previous drafts

of the article. We thank Peter Gaˇzi for comments on previous drafts of the article and assisting

us to generalize Theorem 4.30 to viable forks. We thank Saad Quader for noting an inconsistency

in previous definitions of common prefix and divergence and assisting us to adapt Theorem 4.30.

We thank George Agapov for the prototype implementation of our protocol in Haskell and Jake

Goulding for the Parity based implementation.

References

[1] Noga Alon and Joel Spencer. The Probabilistic Method. Wiley, 3rd edition, 2008.

[2] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space:

When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, Security and

Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September

10

Available from https://github.com/input-output-hk/parity/tree/experiment-2

(020fd77dc70d3f25e0e0f44bd6b1e19ccf3790d3)

63

Figure 16: Measuring transactions per second in a 40 node, equal stake deployment with slot

length of 5 seconds.

3-5, 2014. Proceedings, volume 8642 of Lecture Notes in Computer Science, pages 538–557.

Springer, 2014.

[3] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols

for realistic adversaries. J. Cryptology, 23(2):281–343, 2010.

[4] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas.

Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In

David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, Proceedings of

the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018,

Toronto, ON, Canada, October 15-19, 2018, pages 913–930. ACM, 2018.

[5] Christian Badertscher, Peter Gaˇzi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas.

Ouroboros chronos: Permissionless clock synchronization via proof-of-stake. Manuscript., 2019.

[6] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. CoRR,

abs/1406.5694, 2014.

[7] 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. SIGMETRICS Performance

Evaluation Review, 42(3):34–37, 2014.

[8] Iddo Bentov, Rafael Pass, and Elaine Shi. The sleepy model of consensus. IACR Cryptology

ePrint Archive, 2016:918, 2016.

[9] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. IACR

Cryptology ePrint Archive, 2016:919, 2016.

[10] Daniel J. Bernstein. Chacha, a variant of salsa20. In SASC: The State of the Art of Stream

Ciphers., 2008.

64

[11] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, Advances in Cryptology: A

Report on CRYPTO 81, CRYPTO 81, IEEE Workshop on Communications Security, Santa

Barbara, California, USA, August 24-26, 1981., pages 11–15. U. C. Santa Barbara, Dept. of

Elec. and Computer Eng., ECE Report No 82-04, 1981.

[12] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes

for delegation of signing rights. J. Cryptology, 25(1):57–115, 2012.

[13] Joseph Bonneau. Why buy when you can rent? - bribery attacks on bitcoin-style consensus.

In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner,

and Kurt Rohloff, editors, Financial Cryptography and Data Security - FC 2016 International

Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016,

Revised Selected Papers, volume 9604 of Lecture Notes in Computer Science, pages 19–26.

Springer, 2016.

[44] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application

to electronic voting. In Michael J. Wiener, editor, Advances in Cryptology - CRYPTO ’99,

19th Annual International Cryptology Conference, Santa Barbara, California, USA, August

15-19, 1999, Proceedings, volume 1666 of Lecture Notes in Computer Science, pages 148–164.

Springer, 1999.

A Remarks on Forkable Strings and Divergence

The analysis in the paper demonstrates that if w ∈ {0, 1}

n

is forkable, then there is in fact a flat

fork F ` w with the extra property that F has no more than two “components,” which is to say

that there are two non-empty tines p and q with the property that for any non-empty tine r of F ,

either r ∼ p or r ∼ q. It is natural to ask whether “two tines suffice,” which is to say that there is

always a flat fork F ` w which is the union of two tines.

In fact, there are forkable strings which require three tines to fork. Specifically, consider the

string

w = 0

2k

1

k

(1001)1

k

0

2k

.

The flat fork shown in Figure 17 below proves that w is forkable. However, no two tines can

successfully fork w.

w =

0

· · ·

0 1

· · ·

1

( 1

0 0 1 )

1

· · ·

1 0

· · ·

0

Figure 17: A fork F for the string w = 0

2k

1

k

(1001)1

k

0

2k

; honest vertices are highlighted with

double borders.

67