Proof-of-Work-based Consensus in Expected-Constant Time*

Juan Garay[1], Aggelos Kiayias[2] and Yu Shen[3]

[1]Texas A&M University, garay@cse.tamu.edu
[2]University of Edinburgh and IOG, aggelos.kiayias@ed.ac.uk
[3]University of Edinburgh, shenyu.tcv@gmail.com

Abstract

In the traditional consensus problem (aka Byzantine agreement), parties are required to agree on a common value despite the malicious behavior of some of them, subject to the condition that if all the honest parties start the execution with the same value, then that should be the outcome. This problem has been extensively studied by both the distributed computing and cryptographic protocols communities. With the advent of blockchains, whose main application—a distributed ledger—essentially requires that miners agree on their views, new techniques have been proposed to solve the problem, and in particular in so-called "permissionless" environments, where parties are not authenticated or have access to point-to-point channels and, further, may come and go as they please.

So far, the fastest way to achieve consensus in the proof-of-work (PoW)-based setting of Bitcoin, takes $O(\mathsf{polylog}\kappa)$ number of rounds, where κ is the security parameter. We present the first protocol in this setting that requires **expected-constant** number of rounds. Furthermore, we show how to apply securely sequential composition in order to yield a fast distributed ledger protocol that settles *all* transactions in expected-constant time. Our result is based on a novel instantiation of "m-for-1 PoWs" on parallel chains that facilitates our basic building block, Chain-King Consensus. The techniques we use, via parallel chains, to port classical protocol design elements (such as Phase-King Consensus, super-phase sequential composition and others) into the permissionless setting may be of independent interest.

*An abridged version of this paper appears in *Proc. EUROCRYPT 2024*.

Contents

1 Introduction

Byzantine agreement (BA, aka consensus) is a classical problem introduced in [PSL80] that asks n parties to agree on a message so that three properties are satisfied: (i) termination, (ii) agreement and (iii) validity, in a setting where any t of the parties may behave maliciously. Validity enforces the non-triviality of solutions, as it requires that if the non-faulty/"honest" parties start the execution with the same value, then that should be the output value.

BA has been classically considered in a "permissioned setting": the parties running the protocol are setup so they are able to reliably and directly communicate with each other, or have access to a public-key directory that reliably lists all their public keys. This is captured by a suitable network or *trusted setup* assumption. The "permissionless setting," on the other hand, was introduced with the development of the Bitcoin blockchain [Nak08], and refers to an environment where parties may enter the protocol execution at will, the communication infrastructure is assumed to deliver messages without reliably identifying their origin, and the trusted setup is reduced to the existence of an unpredictable public string—the "genesis block" (which sometimes for simplicity we will just refer to as a CRS [common reference string], or "public-state setup" [GK20]).

BA in the permissionless setting above using proofs of work (PoW)[1] was first (formally) studied in [GKL15]. In terms of running time, the protocols presented in [GKL15] run in $O(\mathsf{polylog}\kappa)$ rounds, where κ is the security parameter and address the binary input case, where the parties wish to agree on a single bit. Subsequent work improved on various aspects at the expense of stronger assumptions. For example, Andrychowicz and Dziembowski [AD15] offered a multi-valued BA protocol also based on PoWs (RO) but with no trusted setup, assuming in addition the existence of existentially unforgeable signatures, and with a running time proportional to the number of parties. The latter was in turn improved by Garay *et al.* [GKLP18] to $O(\mathsf{polylog}\kappa)$ rounds, and just assuming PoWs and no trusted setup. Recently, an expected-constant-round BA protocol was introduced by Das *et al.* [DEF$^+$22], by requiring in addition to the Andrychowicz and Dziembowski [AD15] assumptions the existence of *verifiable delay functions* (VDFs) [BBBF18]. Refer to Table 1 for a comparison of existing PoW-based (or "PoW-inspired") BA protocols.

Protocol	**Setup & assumptions**	**Round complexity**
[AD15]	RO + SIG	$O(n)$
[GKL15]	CRS + RO	$O(\mathsf{polylog}\kappa)$
[GKLP18]	RO	$O(\mathsf{polylog}\kappa)$
[EFL17]	RO + SIG + TLP	Expected $O(1)$
[DEF$^+$22]	RO + SIG + VDF	Expected $O(1)$
This paper	CRS + RO	Expected $O(1)$

Table 1: *Round complexity of PoW-based (or PoW-inspired) permissionless Byzantine agreement protocols, with their corresponding setup and cryptographic assumptions.*

Given the above state of the art, in this work we focus on the question of solving permissionless BA in the original PoW-based blockchain model of Bitcoin with expected-constant round complexity.

[1] As implemented in the Bitcoin blockchain, via hash functions modeled as a *random oracle* (RO) [BR93].

1.1 Overview of Our Results

We present a new permissionless PoW-based multi-valued BA protocol that has expected-constant round complexity and demonstrate how it can be used to solve permissionless *state machine replication* (SMR, or, equivalently, a *distributed ledger*) [Sch90] with fast settlement. In more detail, our results are as follows.

A new PoW-based permissionless consensus protocol. We put forth *Chain-King Consensus*—the first PoW-based permissionless consensus protocol that achieves agreement and validity in *expected-constant* time. Our construction is based on mining on parallel chains, and "emulating" a classical "phase-king" consensus protocol [BGP89] with a randomized chain (the "chain-king") selection rule on top of the parallel chains construction. Our protocol is based on the following ideas.

First, we revisit the parallel chain technique (cf. [FGKR18, BKT^{+}19, FGKR20]) as a method for combining multiple blockchains advancing in parallel. Our key observation is that running $m = \mathsf{polylog}(\kappa)$ parallel chains is sufficient to maintain independence via an $m{\times}1$[2] PoW technique [GKL15] (while prior work set $m = \Theta(\kappa)$ and hence at best was only able to argue "sub-independence"; see [FGKR20]). In fact, our protocol runs m independent instances of 2×1 PoWs, with the latter component being responsible for transaction processing.[3] The key property we utilize is that in a constant number of rounds, a fraction of the m parallel chains will be sufficiently advanced to offer a form of "common prefix" property (cf. [GKL15]) with a constant probability of success.

Second, and contrary to prior work on parallel chains, we "slice" the chain progression into stages where parallel chains can cross-reference each other. In the first stage, parties converge on their views and ensure fresh randomness is introduced; in the second stage they process transactions; and in the third, they prepare for the cross referencing by the upcoming stage, after which the stages rotate indefinitely. A key property of our cross-referencing rule is the concept of a *dense chain*—a strengthening of the concepts of "chain growth" and "chain quality" [GKL15]. Given the short length of each stage (a constant number of rounds), chain density ensures that the adversary faces difficulties to create multiple compromised chains. The key conclusion of this chain structure is *phase-oblivious agreement*, which refers to the fact that, on a large fraction of chains, the majority of input values are contributed by honest parties.

The core agreement component of our protocol follows the "phase king" approach (cf. [BG89, BGP89]). The key idea of porting this protocol design technique to the permissionless setting is to map the chains in the parallel chains cluster to the roles of the different parties in the classical protocol. As a result, the king itself is one of the chains. Moreover, due to the "dilution" of adversarial power that occurs in the parallel chains setting, we can set the king deterministically to be a specific chain. This technique, which may be of independent interest, results in our "Chain-King Consensus" algorithm.

Chain-King Consensus is one-shot, in the sense that it will provide just a single instance of agreement in the permissionless setting in expected-constant time. The natural question given such protocol is whether it is possible to apply sequential self-composition with running time remaining expected linear in the number of instances. This is a delicate task due to non-simultaneous termination (cf. [CCGZ16]). We provide a round-preserving sequential composition solution that first adapts Bracha termination [Bra84] to the permissionless setting and reduces the "termination slack" among honest parties to 1 phase. Then, we adapt the super-phase expansion technique of [CCGZ16] to widen the interval between state updates from 1 phase to 4 phases. We identify a

[2] Pronounced "m-for-1."

[3] As in [GKL15], the "transactions" being processed in a BA protocol are the input values being proposed by the parties.

set of good properties for a sequence of phases that when they occur parties that are in different timelines can converge on the same single phase and make a unanimous decision to update their state.

A new PoW-based permissionless fast SMR protocol. Given that Chain-King Consensus is a one-shot multi-valued Byzantine agreement protocol terminating in expected constant rounds, next we show how to build a state machine replication protocol on top of its sequential composition. The resulting protocol achieves consistency and expected-constant-time liveness for *all types* of transactions (including the conflicting ones). This answers a question left open in previous work on PoW-based fast ledgers [BKT$^+$19, FGKR20], where fast settlement of transactions was offered only for non-conflicting transactions, thus making our ledger construction the first expected-constant processing time ledger in the PoW setting. We note that fast processing of conflicting transactions can be crucial for many applications such as sequencing smart contract operations. We also describe how it is possible to "bootstrap from genesis": this essential operation permits new parties to join the protocol execution as well as facilitate third party observers who wish to connect and parse the distributed ledger in order to issue transactions or read transaction outputs.

1.2 Related Work

Round complexity of synchronous BA protocols. For "classical" BA protocols with deterministic termination, it is known that $t + 1$ rounds [FL82] are necessary, where t denotes the upper bound on the number of corrupted parties, and matching upper bounds exist, both in the information-theoretic and cryptographic settings [LSP82, DS83, GM93].

The linear dependency of the number of rounds on the number of corrupted parties can be circumvented by introducing randomization. Rabin [Rab83] showed that consensus reduces to an "oblivious common coin" (OCC)—i.e., a common view of the honest parties of some public randomness. As a result, randomized protocols with linear corruption resiliency and probabilistic termination in expected-constant rounds is possible. Later on, Feldman and Micali [FM88] showed how to construct an OCC "from scratch" and gave the first expected-constant-time Byzantine agreement protocol, tolerating the optimal number of corrupted parties (less than 1/3 of the total number of parties), in the information-theoretic setting. In the setting where trusted private setup (i.e., a PKI) is provided, Katz and Koo [KK06] presented an expected-constant-round BA protocol with optimal resiliency (less than 1/2 in the cryptographic setting).

We already mentioned that with the advent of blockchains, BA protocols that do not rely on a fixed set of participants became possible. For PoW-based BA protocols, please refer to the beginning of this section. Regarding Proof-of-Stake protocols, Algorand [CM19] uses *verifiable random functions* (VRFs) to self-elect parties, and agreement and validity are achieved in expected-constant time.

Regarding BA protocols based on some other assumptions, we note that in an unpublished manuscript (also mentioned in the introduction) [EFL17], Eckey, Faust and Loss design an expected-constant-round BA protocol based on PoWs and time-lock puzzles (TLPs). Further, Das *et al.* [DEF$^+$22] propose a BA protocol based on the much stronger primitive of *verifiable delay functions* (VDFs) that also terminate in expected-constant time.

Many PoWs from one PoW. As mentioned in the introduction, Garay, Kiayias and Leonardos [GKL15] showed how to use a Nakamoto-style blockchain to solve BA. Achieving the optimal corruption threshold of less than 1/2 of the participants, however, presented some challenges, which were resolved by the introduction of a technique called "2×1 PoW," which is used to compose two modes of mining, one for blocks and one for inputs. In a nutshell, in 2×1 PoW, a random oracle output is checked twice with respect to *both* its leading zeros and tailing zeros. Sufficient leading zeros implies the success of mining a block, and that's the original—i.e., Bitcoin's—approach to assess and verify whether a PoW has been produced, while sufficient *trailing* zeros imply the success of mining an input. This scheme guarantees that both mining procedures can be safely composed and the adversary is bound to its original computational power and is not able to favor one PoW operation over the other.

The 2×1 PoW primitive has found applications in many other scenarios (e.g., [PS17]) and its generalization—m×1 PoW—makes parallel chains possible and has been used to improve transaction throughput [BKT$^+$19] and for accelerating transaction confirmation [FGKR20]. We note that, in the case of parallel chains existing m×1 PoW constructions cannot achieve full independence on all parallel chains. We elaborate on this in Appendix A.

Non-simultaneous termination and sequential composition. A consequence of the round complexity "acceleration" provided by randomized BA protocols is that their termination is probabilistic and not necessarily simultaneous [DRS90]. This is problematic when this type of BA protocol is invoked by a higher-level protocol. More specifically, parties would not be able to figure out when to safely return to the higher-level protocol and start the next execution. One solution is to run randomized BA protocols for $O(\mathsf{polylog}\kappa)$ rounds where κ is the security parameter. The running time is still independent of the number of parties, and, with overwhelming probability, parties would terminate and be able to start the next execution when $O(\mathsf{polylog}\kappa)$ rounds have elapsed. A more sophisticated sequential composition approach is to employ so-called "Bracha termination" and "super-round" expansion in order to preserve an expected-constant round complexity (cf. [CCGZ16]). We adapt these techniques to the permissionless setting.

Settlement latency in state machine replication. Most PoW-based SMR protocols achieve liveness in a time which is a function of the security parameter, hence suffering from long transaction settlement latency. The "Ledger Combiner" approach [FGKR20] proposes a novel grade assignment function to build a virtual ledger on top of different parallel ledgers, achieving constant settlement time but only for *non-conflicting* transactions. Prism [BKT$^+$19] also gives a PoW-based parallel chain protocol with expected-constant settlement time, but only for non-conflicting transactions. Other approaches to fast transaction settlement include Algorand's [CM19], which being Proof-of-Stake-based, achieves expected-constant settlement delay for all types of transactions. Finally, Momose and Ren [MR22] achieve expected-constant confirmation delay, assuming a PKI and VRFs.

2 Model and Preliminaries

Our model of computation follows Canetti's formulation of "real world" notion of protocol execution [Can00a, Can00b] for multi-party protocols. Inputs are provided by an environment program $\mathcal{Z}$ to parties that execute the protocol Π. The adversary $\mathcal{A}$ is a single entity that takes control of corrupted parties. $\mathcal{A}$ can take control of parties on the fly (i.e., "adaptive") and is allowed to observe honest parties' actions before deciding her reaction (i.e., "rushing"). To specify the "resources" that may be available to the instances running protocol Π—for example, access to reliable point-to-point channels or a "diffuse" channel (see below)—we will follow the approach of describing them as *ideal functionalities* in the terminology of [Can00b].

Clock, random oracle, diffusion and CRS functionalities. We divide time into discrete intervals called "rounds." Parties are always aware of the current round (i.e., synchronous processors) and this is captured by a global clock $\mathcal{G}_{\text{CLOCK}}$ [KMTZ13]. By convention, the hash function H to generate PoWs is modeled as a random oracle $\mathcal{F}_{\text{RO}}$. Message dissemination is synchronous and it guarantees that all honest messages sent at the current round to be delivered to all honest parties at the beginning of the next round. This synchronous communication behavior is captured by $\mathcal{F}_{\text{DIFFUSE}}$ and the adversarial power is limited to reorder messages and let honest parties receive messages originally from $\mathcal{A}$ in two adjacent rounds by selectively choosing the receiver in the first round. Finally, we model a public-state setup by the common reference string (CRS) functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ with some distribution with sufficiently high entropy. A full specification of the above resources can be found in Appendix B.

Honest majority. We express our honest majority condition in terms of parties' computational power, measured in particular by the number of RO queries that they are allowed per round, as opposed to by the number of parties (which are assumed to have equal computational power—cf. [GKL15]).

Definition (Honest majority). *Let h_r, t_r denote the number of honest and corrupted random oracle queries at round r respectively. For all $r \in \mathbb{N}$, it holds that $h_r > t_r$.*

To limit the adversary to make a certain number of queries to $\mathcal{F}_{\text{RO}}$, we adopt the "wrapper functionality" approach (cf. [GMPY11, BMTZ17, GKO$^+$20]) $\mathcal{W}(\mathcal{F}_{\text{RO}})$ that wraps the corresponding resource, thus enforcing the limited access to it.

Byzantine agreement. We adapt the definition of the consensus problem (aka Byzantine agreement [LSP82]) to our permissionless setting (cf. [GKL15]). Note that here agreement implies (eventual) termination.

Definition (Byzantine agreement). *A protocol Π solves Byzantine Agreement in the synchronous setting provided it satisfies the following two properties:*

- **Agreement:** *There is a round after which all honest parties return the same output if queried by the environment.*
- **Validity:** *The output returned by an honest party P equals the input of some party P' at round 1 that is honest at the round P 's output is produced.*

Blockchain notation. A block with target $T \in \mathbb{N}$ is a quadruple of the form $\mathcal{B} = \langle ctr, r, h, x\rangle$ where $ctr, r \in \mathbb{N}$, $h \in \{0,1\}$ and $x \in \{0,1\}^*$. A blockchain $\mathcal{C}$ is a (possibly empty) sequence of blocks; the rightmost block by convention is denoted by head$(\mathcal{C})$ (note head$(\varepsilon) = \varepsilon$). These blocks are chained in the sense that if $\mathcal{B}_{i+1} = \langle ctr, r, h, x\rangle$, then $h = H(\mathcal{B}_i)$, where $H(\cdot)$ is cryptographic hash function with output in $\{0,1\}^\kappa$. We adopt $\mathsf{TS}(\mathcal{B})$ to denote the timestamp of $\mathcal{B}$; and, slightly abusing the notations and omitting the time r, we will use $\mathcal{C}^{\lceil k}$ to denote the chain from pruning all blocks $\mathcal{B}$ such that $\mathsf{TS}(\mathcal{B}) \geq r - k$. Let $\mathbb{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\rangle$ denote m *parallel* chains and $\mathbb{C}_j$ the j-th chain $\mathcal{C}_j$ in $\mathbb{C}$.

Finally, we introduce some basic string notation, which will be useful when describing our multi-chain-oriented PoW mechanism. For a κ-bit string s, where κ is the security parameter, we will use s_i ($i \in [m]$) to denote the i-th bit of s, $[s]_{i\sim m}$ to denote the i-th segment after s is equally divided into m segments—i.e., $[s]_{i\sim m} = s_{[(i-1)*\kappa/m]+1}, \ldots, s_{i*\kappa/m}$. We will write $[s]^{\mathsf{R}}$ as the reverse of string s (i.e., by flipping all its bits), and use $[s]^{\mathsf{R}}_{i\sim m}$ to denote the reverse of the i-th segment.

3 *Chain-King* Consensus

In this section, we present our permissionless expected-constant-time Byzantine agreement protocol, which we name $\mathsf{ChainKingConsensus}$[4]. We first sketch the basic protocol approach—parallel chains

[4] Drawing from the "Phase King" approach to solve classical consensus [BGP89].

in Section 3.1 and phase-based chain-selection rule in Section 3.2. Then, we describe the main protocol in Section 3.3. We next show how to adapt the one-shot protocol execution using sequential composition in order to decide on a series of outputs in Section 3.4.

3.1 Parallel Chains and $m\times 1$ Proofs of Work

We introduce a new approach to achieve full independence of mining on parallel chains while preserving the original simple structure. At a high level, our scheme emulates an ideal setting of m parallel oracles while bounding the security loss that such parallel mining incurs. More specifically, the protocol will run $m = \Theta(\mathsf{polylog}\kappa)$ parallel chains; note that the number of bits allocated on each chain will still be super-logarithmic in the security parameter (i.e., $\kappa/\Theta(\mathsf{polylog}\kappa) = \Omega(\mathsf{polylog}\kappa)$), and hence the protocol will allow an arbitrary number of participants. Later we will show that (i) poly-logarithmically many parallel chains suffice to achieve the desired convergence; and (ii) poly-logarithmically many bits (those will be the bits available to each of the parallel random oracle invocations) will suffice to eliminate bad events with respect to the random oracle.

Our parallel chain structure. We will use $m = \Theta(\log^2 \kappa)$ parallel chains as the basic building block for $\mathsf{ChainKingConsensus}$. Importantly, on each chain we will employ the 2×1 PoW technique [GKL15] to bind the mining process of the chain with input messages (which will be used to reach consensus; details in Section 3.3). At a high level, this can be viewed as running m ideal parallel repetitions of a 2×1 PoW blockchain.

We will call the blocks that form the blockchains a *chain-block* (or block for short) and denote it by $\mathcal{B}$, and the application data field, which will contain consensus-related values, we will call an *input-block*, and denote it by $\mathtt{IB}$. Since the protocol will run an m chain production procedure and m input-block production procedure, we will make a one-to-one correspondence between the chain-blocks and input-blocks. More precisely, the input-block produced by the i-th segment of the RO output will only be valid on the i-th parallel chain. See Figure 1 for an illustration of the RO output and how successes on the bounded mining procedures are achieved.

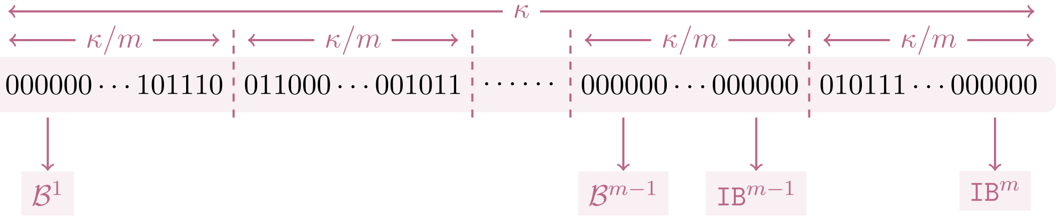


Figure 1: *The mining process on our parallel chain. We assume that the target value is appropriately set so that at least 6 leading zeroes imply a success of the chain's block mining and at least 6 tailing zeroes blocks implies a success of the input mining. The blocks' superscript denotes on which chain they will be valid.*

We now provide details on the blocks' structure. Since the mining procedure of chain-blocks and input-blocks are bound together, they share the same block header $\langle ctr, r, h, st, h', val\rangle$, which is a concatenation of random nonce $ctr \in \mathbb{N}$, timestamp $r \in \mathbb{N}$, previous hash reference $h \in \{0,1\}^\kappa$, block state $st \in \{0,1\}^*$ (Merkle root of content), input freshness $h' \in \{0,1\}^\kappa$, and input message $val \in \{0,1\}^*$. Note that the previous hash h is a string of κ bits, consisting of m segments of the previous block hash of length κ/m. Block state st is a concatenation of m block states; this is by convention the Merkle tree root of block content whose details we will omit for now (later on we will use $\mathsf{Blockify}$ to denote the procedure of generating block states). Input freshness h' is a string of κ bits and can be extracted from the (local) chain by procedure ExtractInputFreshness. We defer the details of this algorithm to Section 3.2 and use it in a black-box way here. The input value *val* is the message that is of concern to the consensus protocol. For instance, in the case of binary consensus, $val \in \{0, 1\}$ and in multi-valued consensus *val* is a value picked from a larger input domain. Looking ahead, we note that when performing "slack" reduction and sequential composition of protocol instances, *val* may convey additional information (Section 3.4). Moreover, we remark that for all parallel chains, parties will try to mine the input-block that contains the *same* input message, hence, unlike h, st and h', in this field all values only need to appear once.

We note that as multiple mining procedures are bound together, for a valid block with respect to a particular chain, those header bits associated with other procedures become "dummy" and will only be useful when validating whether the block corresponds to a successful PoW. We now provide details about such dummy information. Regarding a valid chain-block on the i-th chain, only the nonce *ctr*, timestamp r, i-th segment of previous hash reference $[h]_{i \sim m}$ and i-th segment of block state $[st]_{i \sim m}$ are useful information. All other bits in h and st, along with input freshness reference h' and input message *val* are dummy information and they are merely used in the PoW validation[5] On a similar vein, for input-blocks that are valid on the i-th chain, only the nonce *ctr*, timestamp r, input message *val* and i-th segment of fresh randomness $[h']_{i \sim m}$ are useful information; all other bits in h', previous hash reference h and block content root st are dummy information.

We are now ready to describe the mining procedure. Given a parallel chain $\mathbb{C}$, block state st and input *val*, first, the protocol extracts the previous hash reference by concatenating the i-th segment of block hash computed from the tip of the i-th chain (recall that each segment is a (κ/m)-bit string). (When the chain is empty it refers to the corresponding segment in the CRS.) Next, after calling ExtractInputFreshness on $\mathbb{C}$ to obtain the input randomness, the protocol queries the random oracle and gets output u. Then, it divides u into m segments of equal length and iterates over those segments. If the original i-th segment is less than T, the protocol successfully mines a new chain-block on the i-th chain and appends it to $\mathbb{C}$. If the reverse of i-th segment is less than T, the protocol succeeds in mining an input-block and stores it locally (and will be diffused in the future). See Algorithm 1 for a full description of the mining procedure.

Algorithm 1 ParallelPoW($\mathbb{C}, r, st, val$)

```
$h \leftarrow \varepsilon$
for $i = 1$ to $m$ do
    if $\mathbb{C}_i = \varepsilon$ then
        $h \leftarrow h \,\|\, 0^{\kappa/m}$
    else
        $\mathcal{B} \leftarrow \text{head}(\mathbb{C}_i)$ and $h \leftarrow h \,\|\, [H(\mathcal{B})]_{i \sim m}$
    end if
end for
$h' \leftarrow$ ExtractInputFreshness($\mathbb{C}, r$)                    ◁ Call Algorithm 2
$u \leftarrow H(ctr, r, h, st, h', val)$
IB $\leftarrow \varepsilon$
▷ Check if PoW succeeds on any chain/input-block.
for $i = 1$ to $m$ do
    if $[u]_{i \sim m} < T$ then $\mathbb{C}^{(i)} \leftarrow \mathbb{C}^{(i)} \,\|\, \langle ctr, r, h, st, h', val \rangle$    ◁ Extend chain
```

[5] We note that later on (Section 3.2), after we introduce *phase-based* parallel chains, initial blocks in each phase will have to provide a good fresh randomness h' in order to pass the cross-chain validation check.

```
        if $[u]^{\mathsf{R}}_{i\sim m} < T$ then IB $\leftarrow \langle ctr, r, h, st, h', val \rangle$
    end for
    $ctr \leftarrow ctr + 1$
    return $\mathbb{C}$, IB
```

Basic properties of our parallel chain structure. As a warm-up, we present a preliminary analysis of our parallel chain structure. The goal is to show that, when appropriately parameterized, a constant fraction of the parallel chains will have "good" properties.

Our main analytical approach follows that in [GKL14, GKL17], where the focus is on whether an execution on a *single* chain is *typical*—i.e., whether random variables related to the execution on this single chain stays close to their expected values and bad events with respect to the RO never happen. In [GKL14] it is proved that any execution of the protocol for a number of rounds at least polylogarithmic in the security parameter, is *typical* with overwhelming probability.

Here we apply the above technique to a *constant* number of rounds and adapt it to our setting where the mining procedure of chain-blocks and input-blocks are bound together. Importantly, we are interested in the random variables expressing the total number of rounds that *at least* one honest chain-block (resp., input-block) is produced, the total number of rounds that *exactly* one honest chain-block is produced, and the total number of adversarial successes on chain-blocks (resp., input-blocks). For the sake of conciseness, here we provide an informal description of typical executions and defer all full definitions and proofs to Appendix D.1.

Definition 1 (Typical execution, informal). *An execution is* typical *if for any set of at least k consecutive rounds, bad events (collisions) on the RO never happen, and the following quantities stay close to their expected values:*

(i) the number of rounds where at least *one honest chain-block (resp., input-block) is produced;*

(ii) the number of rounds where exactly *one honest chain-block is produced;*

(iii) the number of adversarial chain-blocks (resp., input-blocks).

Note that since in our case k is a constant, the probability that in a k-round window the execution is typical is constant due to Chernoff bounds (Theorem 9). Hence, the probability that an execution running for $L = \mathsf{poly}(\kappa)$ rounds is typical will be negligible. Nonetheless, let us consider a constant number $\rho \in \mathbb{N}^+$ of rounds. When the protocol is appropriately parameterized, the execution running for ρ steps will be typical with constant probability. The intuition here is that, the number of windows of at least k rounds within the period of ρ rounds is $\Theta(\rho^2)$. For any constant $\beta < 1$, when the probability that a k-round time window is typical is α, then by choosing $\rho \leq \sqrt{\ln \beta / \ln \alpha}$ we get the desired convergence probability. Moreover, for the same β, ρ can be chosen as an arbitrary multiple of k (see Appendix D.1 for details).

Given the full independence of the m mining processes, we show that when the number of m parallel chains is sufficiently large, the success probability of a single execution being typical translates to the *fraction* of typical executions among the m parallel executions, yielding the following:

Theorem 1. *For any $\beta < 1$, running $m = \Theta(\log^2 \kappa)$ parallel chains as described above for a constant number ρ of rounds, results in at least a β fraction of them being typical with overwhelming probability in κ.*

3.2 From Parallel Chains to Phase Oblivious Agreement

Given that running parallel chains from the CRS enjoy good properties only when the lifetime of the execution is bounded by a constant (Theorem 1), we now show how to combine the parallel chain structure with a novel phase-based cross-chain reference scheme in order to provide fresh randomness and extend the protocol running time to any polynomial in terms of the security parameter. This gives us novel chain validation and selection rules. Moreover, we show that in each phase, the approach achieves what we call *phase oblivious agreement*, which serves as an essential building block in our ChainKingConsensus protocol.

In this section, we assume static participation where parties are always online and their number is fixed yet unknown to any protocol participant. Later on (Section 4.2), we elaborate on how to let new joining parties synchronize with other participants.

Protocol phases. We divide the protocol execution time into sequential, non-overlapping phases of length ρ rounds. Note that ρ is a constant and at round i parties are in the $\lceil i/\rho \rceil$-th phase (the phase index starts at 1). As we assume synchronous processors, parties are always aware of the current round and phase numbers (they maintain local variables `r` and `phase` to store this information).

In contrast to the "conventional" longest-chain consensus rule where parties keep extending chains starting from the genesis block, in our protocol in each phase parties will build parallel chains separately, which we will denote $\mathbb{C}^{(i)}$, and the j-th chain in the i-th phase by $\mathbb{C}_j^{(i)}$. Let $\mathbb{C}$ now denote the sequence of parallel chains in each phase—i.e., $\mathbb{C} = \mathbb{C}^{(1)}, \mathbb{C}^{(2)}, \ldots, \mathbb{C}^{(i)}$. In the first phase, $\mathbb{C}^{(1)}$ points to the CRS, thus the adversary starts the computation simultaneously with the honest parties. Unfortunately, the CRS is only available at the onset of the execution, and hence, naïvely, there is no method to prevent the adversary from mining into the future—e.g., when it is in phase i, he can mine blocks for phase $i + 1$. If pre-mining is possible for an unbounded time, then no security guarantees can be achieved in the $(i + 1)$-th phase even if typical execution holds.

One conventional method to solve the pre-mining problem in blockchains (cf. [PS17]) consists of referring to a stable block with randomness that is unpredictable to the adversary (e.g., an honest block). Unfortunately, since phases here only last for a constant number of rounds, thus far there is no approach that would enable parties to explicitly agree on common unpredictable randomness in constant time (as this would directly imply full agreement on a non-trivial fact, which is our goal). Without a full agreement on common randomness, the adversary can split the honest computational power by building a chain with randomness that is acceptable by, say, half of the honest parties but that will be rejected by the rest. In such way the adversary can then split the honest computational power and thus completely break the security of the protocol.

To overcome the failure of the conventional common fresh randomness approach, we propose a new scheme called "cross-chain reference" to secure the execution on parallel chains in the second and subsequent phases. In short, cross-chain reference asks for all chains in the i-th phase to point to a large fraction of the chains in the $(i - 1)$-th phase that are "dense," a property which we will elaborate on soon.

As a preparation for securing phase-based parallel chains, we first introduce the structure of a phase (see Figure 2). A phase, consisting of ρ rounds, is further divided into three non-overlapping *stages*. A block is assigned to a specific stage based on its timestamp. The first stage, *view convergence*, consists of the first ρ_{view} rounds in a phase. It guarantees that at the end of this stage, on sufficiently many parallel chains, honest parties agree on a common prefix *obliviously* and they input some *recent* randomness so that the adversary cannot pre-compute too many blocks for the next stage. Then, the second stage, *output generation*, which consists of ρ_{output} rounds after the view convergence stage, is used to decide the output of this phase. Only input blocks that are included by chain-blocks in this stage will be considered in the decision making procedure at the end of this phase (details in Section 3.3). The length ρ_{output} is chosen sufficiently large so that the honest input-blocks account for the majority on sufficiently many parallel chains. Finally, the last stage, *reference convergence*, consists of the last ρ_{ref} rounds. This last stage is used to secure the blocks that will be pointed by the cross-chain reference. Note that ρ_{ref} is also the upper bound on adversarial pre-mining—i.e., the adversary cannot start to mine blocks in the next phase ρ_{ref} rounds earlier than the honest parties.

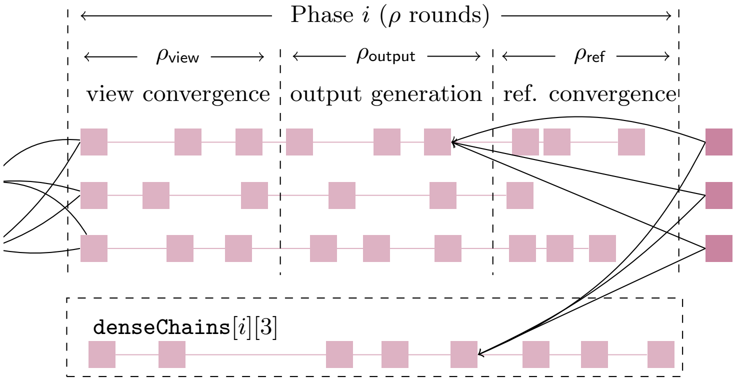


Figure 2: *An illustration of a party* P*'s local parallel chains* $\mathbb{C}_{\mathsf{local}}$ *and dense chains* `denseChains`*. In order for initial blocks in phase* $i+1$ *to be valid, they should point to at least 2 dense chains in phase* i*. In this toy example, all blocks point to the first dense chain in* $\mathbb{C}_{\mathsf{local}}$ *and the third dense chain in* `denseChains`*. Note that the second chain is not dense.*

Dense chains. Next, we introduce a new concept called *dense chains* which asks for the density of a chain (in terms of the number of blocks with timestamps in a given time period) and can also be used as a proof of "chain growth" (cf. [GKL14]). Specifically, let τ, s denote the density parameter. We say a chain $\mathcal{C}$ is "dense" (formal definition coming up below) in a time window (i.e., sequence of rounds) $\{u, \ldots, v\}$ if for any set S of at least s consecutive rounds in $\{u, \ldots, v\}$, $\mathcal{C}$ has at least $\tau \cdot |S|$ blocks with timestamps in S.

When referring to a single chain $\mathcal{C}$ in phase-based parallel chains (e.g., the j-th chain in the i-th phase), we will say that $\mathcal{C}$ is dense if for any set S of ρ_{ref} consecutive rounds in the output generation and reference convergence stages, there are *more than* $\rho_{\mathsf{ref}} \cdot \tau$ blocks in $\mathcal{C}$ reporting timestamps in S. Formally:

Definition 2 (Dense chains). *A chain* $\mathcal{C}$ *is a* (τ, s, u, v)-dense chain *if for any set* $S = \{p, \ldots, q\}$ *of consecutive rounds such that* $u \le p < q \le v$ *and* $|S| > s$*, there are at least* $\tau \cdot |S|$ *blocks in* $\mathcal{C}$ *with timestamp in* S*. A chain* $\mathcal{C}$ *is a* dense chain on phase i *if it is a* $(\tau, \rho_{\mathsf{ref}}, (i-1)\rho + \rho_{\mathsf{view}}, i \cdot \rho)$*-dense chain—i.e., the chain is dense in the last two stages of the* i*-th phase.*

We choose the density parameter τ in such a way that when typical execution property holds on a single chain, the following two properties are guaranteed: (i) even if the adversary completely stops producing PoWs, the honest parties by themselves can produce a dense chain; and (ii) in the i-th phase, the adversary cannot come up with a dense chain before the reference convergence stage.

With foresight, the purpose of dense chains is to secure the execution of future phases, by asking parties to provide sufficiently many dense chains as a proof of having invested enough computational power before the current phase.

Cross-chain references. Next, we elaborate on the cross-chain reference approach which we use to "link" neighboring phases (this provides unpredictability so that the adversary can only pre-mine for a bounded amount of time). At a high level, a cross-chain reference on an initial block in the j-th chain and i-th phase is a κ-bit string consisting of m pointers to m sufficiently deep blocks on chains in the $(i-1)$-th phase. These deep blocks are picked as the last blocks in the output generation stage, one on each chain. Their hashes (that is, the j-th segment of a block hash, for a block on the j-th chain) are concatenated to form the κ-bit string. We assign this reference to the input freshness h' (recall our block header structure in Section 3.1) in the initial blocks on each chain's i-th phase. For a cross-chain reference to be considered valid, it should point to at least a large fraction of deep blocks in *dense* chains in the previous phase[6]. However, these dense chains are not necessarily required to match the parties' own chains of the previous phase, but can be attached as a proof of validity.

To facilitate the chain validation and selection algorithm, a party P maintains local variables $\mathbb{C}_{\mathsf{local}}$ to record her own parallel chains and `denseChains` to bookkeep all valid (single) dense chains that are not in $\mathbb{C}_{\mathsf{local}}$. Note that `denseChains` and $\mathbb{C}_{\mathsf{local}}$ are diffused together. In more detail, `denseChains` is a two dimension vector with `denseChains`$[i][j]$ containing a (possibly empty) set of (single) dense chains that a party has seen as the j-th chain in i-th phase. Party P also maintains a local variable `chainBuffer` which contains all pairs of $\langle \mathbb{C}, \texttt{denseChains} \rangle$ that P receives at the beginning of the round. Refer to Figure 2 for an illustration of our phase-based parallel chain.

We now formalize the $\mathsf{ExtractInputFreshness}$ procedure (see Algorithm 2) which parties use to extract cross-chain reference and fresh randomness for input-blocks. Specifically, when this algorithm is called in the view convergence stage of the i-th phase $(i > 1)$, it returns a κ-bit string which is a concatenation of hashes of the blocks with largest block height whose timestamp is less than $(i-1)\rho - \rho_{\mathsf{ref}}$ on each chain. When $\mathsf{ExtractInputFreshness}$ is called in the output generation stage, it returns the concatenation of m hashes of the blocks that are k-rounds before the end of the view convergence stage in this phase (we will show later that k is the parameter for common prefix on typical chains). When this algorithm is called at any other time, it returns an all-zero string.

Algorithm 2 $\mathsf{ExtractInputFreshness}(\mathbb{C}, r)$

```
if $r \leq \rho_{\mathsf{view}}$ then return $0^\kappa$                              ◁ First phase
if $r \mod \rho > \rho - \rho_{\mathsf{ref}}$ then return $0^\kappa$                ◁ Reference convergence stage
$h' \leftarrow \varepsilon, i \leftarrow \lceil r/\rho \rceil$
if $r \mod \rho \leq \rho_{\mathsf{view}}$ then                                 ◁ Get cross-chain reference
    for $j$ from $1$ to $m$ do
        $\mathcal{C} \leftarrow \mathbb{C}_j^{(i-1)}$ and $\mathcal{B} \leftarrow \mathrm{head}(\mathcal{C}^{\lceil \rho_{\mathsf{ref}}})$          ◁ Extract chain in previous phase
        $h' \leftarrow h' \parallel [H(\mathcal{B})]_{j\sim m}$
    end for
else                                                           ◁ Get input freshness for IB
    $r^* \leftarrow (i-1) \cdot \rho + \rho_{\mathsf{view}} - k$               ◁ $k$ rounds before the end of view convergence.
    for $j$ from $1$ to $m$ do
```

[6] We cannot require the cross-chain reference to point to all the dense chains in previous phase for two reasons: (i) when typical execution fails it can be the case that neither the honest parties nor the adversary produce a dense chain; and (ii) the adversary can split parties by delivering a private adversarial dense chain to only some of them.

```
        $\mathcal{C} \leftarrow \mathbb{C}_j^{(i)}$ and let $\mathcal{B}$ be the block in $\mathcal{C}$ with largest block height and $\mathsf{TS}(\mathcal{B}) < r^*$
        $h' \leftarrow h' \parallel [H(\mathcal{B})]_{j\sim m}$
    end for
end if
return $h'$
```

Parallel-chain validation algorithm. Recall that in our protocol, we use a $m\times 1$ PoW scheme to mine m parallel chains, and, on each chain, we use 2×1 PoW to bind the mining process of chain-blocks $\mathcal{B}$ and input-blocks IB together; moreover, we divide chains into phases and introduce cross-chain reference to link neighbouring phases. Our validation rule will consider the validity of all blocks, chains (in a single phase) and the cross-chain references. Specifically, a parallel chain $\mathbb{C}$ (with its associated denseChains) will be considered valid if the following holds (refer to Algorithm 8 in Appendix C for a full description):

- *Valid single chains.* For any $\mathcal{C} = \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ (either a $\mathbb{C}_j^{(i)}$ or in denseChains$[i][j]$), $\mathcal{C}$ should be a valid single chain. More specifically, $\mathcal{C}$ is a valid chain if (i) all blocks are the result of successful PoWs; (ii) all blocks' state st match their corresponding block content; and (iii) for all $i > 1$, B_i refers to the hash of B_{i-1}. Additionally, for chains in the first phase, $\mathcal{B}_1$ should point to the CRS.
- *Valid input blocks.* For any input block IB included in $\mathcal{C}$ in the i-th phase, IB should pass the following check: (i) it reports a unique hash among all input-blocks; (ii) the timestamp of IB falls in the output generation stage; (iii) IB is a successful PoW and contains a valid input message val; and (iv) IB points to the last block on $\mathcal{C}$ with timestamp less than $(i-1)\rho + \rho_{\mathsf{view}} - k$ (i.e., good fresh randomness).
- *Valid cross-chain reference.* In the i-th phase ($i > 1$), all initial blocks of chains in $\mathbb{C}^{(i)}$ and denseChains$[i]$ report good cross-chain reference. In order for a cross-chain reference to be good, at least a $\beta > 3/4$ fraction of hashes should match the last blocks in the output generation stage on dense chains in the $(i-1)$-th phase, either in $\mathbb{C}$ or denseChains. Note that their positions should also match—i.e., the j-th segment of reference should match a deep block in $\mathbb{C}_j^{(i-1)}$ or denseChains$[i-1][j]$.

We remark that our chain validation rule is different from that used in both the single chain validation as well as in all previous parallel-chain constructions due to its novel cross-chain reference mechanism. Specifically, starting in the second, the initial block on a single chain $\mathcal{C}$ does not directly point to the last block in the previous phase—i.e. its previous state reference h becomes dummy. As long as $\mathcal{C}$ provides a valid cross-chain reference and forms a valid single chain, $\mathcal{C}$ will be considered as valid. We note that since previous state references (the hash pointer) between neighboring phases are not continuous, the adversary is allowed to keep extending the head of the chains in the previous phase by keeping mining and inserting blocks. Moreover, as our protocol does not ask for cross-references to all previous chains, it is also possible that honest parties never hold exactly the same parallel chain.

We now provide some more intuition on these two new properties. Regarding the adversarial extension of chains from previous phases, parties will check-point their chains phase-by-phase (see the chain selection rule below), hence this does not undermine the security of online parties. Regarding the possible disagreement on a certain fraction of the parallel chains, we note that this is unavoidable. Otherwise, if parties were aware that they would achieve a full agreement on a specific phase, this would directly imply that they reach consensus (and with simultaneous termination!). Our goal is to let honest parties share parallel chains such that in each phase, they *obliviously* agree on the prefix of a large fraction of the chains.

Parallel-chain selection algorithm. We now introduce the chain selection algorithm. In a nutshell, this algorithm does not update local parallel chains as a whole; rather, it updates each single chain in the current phase—i.e., after phase i has passed, $\mathbb{C}_{\mathsf{local}}$ is check-pointed up to phase i and all chains in the previous phases will never be changed. When parties are in the first phase, they use the longest chain rule to select each single chain separately. When parties are in the i-th phase ($i > 1$), a party P processes the chains stored in `chainBuffer` as follows:

- *Filter invalid chains.* For any $\mathbb{C} \in$ `chainBuffer`, if $\mathbb{C}$ is not a valid chain, P rejects $\mathbb{C}$ immediately and removes it (as well as its associated dense chains) from `chainBuffer`.
- *Update* `denseChains`. For all $i' < i$ and $j \in [m]$, P updates $\texttt{denseChains}[i'][j]$ as follows. If there is a valid dense chain $\mathcal{C}$ as the j-th chain in phase i' (either in $\mathbb{C}$ or $\texttt{denseChains}[i'][j]$ from another party) that forks from all the chains in $\texttt{denseChains}[i'-1][j]$ for more than ρ_{ref} rounds, P adds $\mathcal{C}$ to $\texttt{denseChains}[i'-1][j]$ (i.e., it bookkeeps new dense chains with new cross-chain reference pointer blocks).
- *Adopt longer chains.* P uses the longest chain rule to select chains in the current phase. For any incoming chain $\mathbb{C}$, if $\mathrm{len}(\mathbb{C}_j^{(i)}) > \mathrm{len}(\mathcal{C})$ where $\mathcal{C}$ is the j-th chain in $\mathbb{C}_{\mathsf{local}}^{(i)}$, then P updates $\mathcal{C}$ to $\mathbb{C}_j^{(i)}$.

See Algorithm 3 for a detailed description of the above rules.

Algorithm 3 $\mathsf{UpdateLocalChain}(\mathbb{C}_{\mathsf{local}}, \texttt{chainBuffer})$

```
▷ This algorithm should only be called by fully-synchronized parties.
for $\langle \mathbb{C}, denseChains \rangle \in$ chainBuffer do
    if not IsValidChain($\mathbb{C}$) then Continue                    ◁ Skip invalid chains
    ▷ Add new dense chains in previous phases to denseChains
    for $i$ from 1 to phase $- 1$ do
        for $j$ from 1 to $m$ do
            Parse $\mathcal{C} = \mathbb{C}_j^{(i)}$
            if isDenseChain($\mathcal{C}$) then
                if not $\exists \mathcal{C}' \in$ denseChains$[i][j]$ and $[\mathrm{head}(\mathcal{C}^{\lceil \rho_{\mathsf{ref}}})]_{q \sim m} = [\mathrm{head}(\mathcal{C}'^{\lceil \rho_{\mathsf{ref}}})]_{q \sim m}$ then
                    Add $\mathcal{C}$ to denseChains$[i][j]$
                end if
            end if
            for $\mathcal{C} \in$ denseChains$[i][j]$ do
                if not $\exists \mathcal{C}' \in$ denseChains$[i][j]$ and $[\mathrm{head}(\mathcal{C}^{\lceil \rho_{\mathsf{ref}}})]_{q \sim m} = [\mathrm{head}(\mathcal{C}'^{\lceil \rho_{\mathsf{ref}}})]_{q \sim m}$ then
                    Add $\mathcal{C}$ to denseChains$[i][j]$
                end if
            end for
        end for
    end for
    ▷ Extend chains in current phase using longest chain rule
    for $j$ from $1$ to $m$ do
        Parse $\mathcal{C} = \mathbb{C}_j^{(\mathsf{phase})}$ and $\mathcal{C}'$ as the $j$-th chain in $\mathbb{C}_{\mathsf{local}}^{(\mathsf{phase})}$
        if $\mathrm{len}(\mathcal{C}) > \mathrm{len}(\mathcal{C}')$ then Update the $j$-th chain in $\mathbb{C}_{\mathsf{local}}^{(\mathsf{phase})}$ to $\mathcal{C}$
    end for
end for
```

Phase oblivious agreement. Notice that in each phase, the probability that for a large fraction of chains their execution is typical (Definition 1) is overwhelming. Further, our phase-based parallel-chain structure and density-based chain validation and selection rules guarantee that the adversary can only pre-mine for a bounded amount of time, hence "good" properties—i.e., agreement and chain quality (high enough fraction of honest blocks) on the input-blocks—hold on a large fraction of the chains in every phase. Except that as parties are not able to discern on which chains they have agreement, agreement is achieved *obliviously*, yielding the following:

Theorem 2 (Phase oblivious agreement). *There exist protocol parameterizations such that the following properties hold. Let $\beta \in (3/4, 1)$ and consider a phase i. Let $\mathbb{C}, \mathbb{C}'$ denote the parallel chains held by two honest parties P, P' at rounds r, r' after phase i (i.e., $\min\{r, r'\} > i\rho$), respectively. Then there exists a subset $S \subseteq \{1, 2, \ldots, m\}$ of size larger than $\beta \cdot m$ such that for all $j \in S$, the following two properties hold on chains $\mathcal{C} = \mathbb{C}_j^{(i)}$ and $\mathcal{C}' = \mathbb{C}_j'^{(i)}$.*

- **Agreement:** $\mathcal{C}^{\lceil \rho_{\mathsf{ref}}} = \mathcal{C}'^{\lceil \rho_{\mathsf{ref}}}$.
- **Honest input-block majority:** *For all input-blocks included in the output generation stage of $\mathcal{C}$ and $\mathcal{C}'$, more than half of them are produced by honest parties.*

Refer to Appendix D.2 for a detailed analysis of the algorithms in this section and the proof of Theorem 2.

3.3 From Phase Oblivious Agreement to Chain-King Consensus

In this section we explain how our chain-king consensus protocol can be derived from phase-based parallel chains. We present $\mathsf{ChainKingConsensus}$ as a multi-valued consensus protocol with input domain $V, |V| \geq 2$.[7] For simplicity we assume inputs are scalars, but the formulation can be easily adapted to any other type of input.

At a high-level chain-king consensus can be viewed as following the "phase king" approach (cf. [BG89, BGP89]) with randomized king selection on top of phase-based parallel chains. The execution is based on the iteration of 3 phases. Parties will only terminate at the end of each iteration (i.e., the phase with index a multiple of 3). Two thresholds, more than one half of the number of chains ($> m/2$) and more than three-quarters ($> 3m/4$), are of interest. Importantly, a distinguished chain—the *first* chain $\mathbb{C}_1$—is identified as the *king chain*. This king chain is hard-coded in the protocol and will never change during the whole execution.

Similarly to all existing consensus protocols with probabilistic termination, in $\mathsf{ChainKingConsensus}$ parties might terminate at different phases. We measure the quality of non-simultaneous termination by measuring the maximum number of phases that two honest parties can terminate apart from each other:

[7] We remark that our protocol is a multi-valued consensus protocol directly by construction, rather than following the common approach of first designing a binary consensus protocol and then applying the Turpin-Coan pre-processing step [TC84].

Definition 3 (**c-slack termination**). *A protocol* Π *satisfies c-slack termination if any pair of honest parties* P, P' *are guaranteed to terminate* Π *within c phases of each other.*

Input messages and internal variables. So far we have not yet specified the input messages in each phase. In (multi-valued) ChainKingConsensus, at the onset of the protocol execution, party P is activated with an input $v \in V$. P starts to mine input messages (i.e., by setting a variable `val` in the RO query—see Section 3.1) which is their current suggestion for the protocol output; P will terminate based on her local states which we will detail soon.

In addition to variable `val` $\in V$, P locally manages two Boolean variables `lock` and `decide` which are both initialized to false, and a three-valued variable `exit` $\in \{\infty, 1, 0\}$ which is initialized to ∞. In more detail:

- Variable `val` reflects P's suggestion on the output, and can be modified if in certain phases P receives sufficiently many different input values.
- Value `lock` indicates whether P will "listen" to the king-chain (see Algorithm 5 below for details) in the last phase of an iteration. It is set to true if parties are confident that all honest parties will set their `val` to the same value. If `lock` remains false at the end of an iteration, P will update her `val` based on her local view of the king-chain. If P has not decided at the end of an interation and `lock` is set to true, it is reset to false for the next iteration.
- Variable `decide` is used to record whether P decides on her local value `val`. It is set to true only when P is confident that all honest parties are going to agree on the value that she holds, and the adversary is limited to only influencing in which phase parties will terminate. When `decide` is set to true, `val` is fixed and will never change in the future (except with neglibible probability). Further, it is set to true only in the first and second phase of an iteration and is checked at the last phase to see if `exit` needs to be updated.
- Variable `exit` indicates whether P should stop querying the RO and producing blocks. When `exit` $= \infty$, P has not yet reached the end of the iteration when she decides, hence P keeps updating the other variables. When `exit` $= 1$, P have set `decide` to true and hence is ready to output `val`. However, P is not aware if other honest parties have decided, hence P keeps producing blocks with `val`. This will last for one iteration and then `exit` is set to 0. When `exit` $= 0$, P stops making RO queries and stops the execution of (this instance of the) protocol.

We highlight one significant difference between Chain-King Consensus and classical BA protocols. In the classical setting, parties terminate the protocol once they decide on an output. For protocols with probabilistic termination, some honest parties might terminate a few rounds after other honest parties (cf. [DRS90]). Parties who have terminated continue to send the same message to all honest parties (cf. [FM88, KK06]), and the parties that are behind can stick to the previous message if they do not receive any new message from those parties that have already terminated. As it turns out, this strategy essentially relies on the set of participating parties being known, which does not apply the permissionless setting where parties can neither authenticate with each other nor know the source of a message. Hence, in order to let parties that are behind safely terminate, we explicitly distinguish "`decide`," which means parties output their local variable `val`, and "`exit`," which means parties stop (or will stop) the PoW mining process and exit the protocol. We provide more details on "mining for one more iteration" after we introduce the state update algorithm.

Phase output extraction. The decision made at the end of each phase is based on the input messages collected in that phase. Since we have m parallel chains, parties will extract a vector of size m. For the i-th element in the vector, it is extracted from the *median*[8] of input values that appear in the input-blocks, collected in the output generation stage (i.e., blocks with timestamps in $(i\rho - (\rho_{\mathsf{output}} + \rho_{\mathsf{ref}}), i\rho - \rho_{\mathsf{ref}}]$ in i-th phase). Note that since parties might disagree on some bounded fraction of the chains, different honest parties will extract different phase output vectors. Nevertheless, thanks to Theorem 2, two honest output vectors will share a large fraction of common elements obliviously. (See Algorithm 4 for the full description of this process.)

Algorithm 4 $\mathsf{ExtractPhaseVector}(\mathbb{C}, r, tp)$

```
if $r < tp \cdot \rho$ then return $\perp$                    ◁ Too early to extract target phase.
Initialize $\vec{V}$ to an empty array
for $i = 1$ to $m$ do
    Initialize $\vec{M}$ to an empty array
    $\mathcal{C} \leftarrow \mathbb{C}_i^{(tp)}$
    for $\mathcal{B} \in \mathcal{C}$ and $tp \cdot \rho - (\rho_{\mathsf{output}} + \rho_{\mathsf{ref}}) < \mathsf{TS}(\mathcal{B}) \leq tp \cdot \rho - \rho_{\mathsf{ref}}$ do
        Extract input message $val$ from $\mathcal{B}$ and add $val$ to $\vec{M}$
    end for
    Sort $\vec{M}$ then add $\mathsf{med}(\vec{M})$ to $\vec{V}$
end for
return $\vec{V}$
```

State update algorithm. At the end of each phase (i.e., when local clocks reach round $i \cdot \rho$), parties run Algorithm 5 to decide whether to update their local variables or not. It generally follows the randomized phase-king algorithm approach [FG03], but introduces a novel king selection rule and an extra termination iteration.

Algorithm 5 $\mathsf{StateUpdate}$

```
    ▷ This algorithm is called once in each phase. It directly interacts with internal variables
    val, lock, decide and exit.
if $\texttt{r} \mod \rho \neq 0$ then return                         ◁ Not the end of a phase
if $\texttt{exit} = 1$ then
    if $\texttt{phase} \mod 3 = 0$ then $\texttt{exit} \leftarrow 0$            ◁ End of "extra mining iteration"
    return
end if
$\vec{V} \leftarrow \mathsf{ExtractPhaseVector}(\mathbb{C}_{\mathsf{local}}, \texttt{r}, \texttt{phase})$
Let $val$ denote the most frequent element in $V$ and $c$ its frequency
if $\texttt{phase} \mod 3 = 1$ then                                        ◁ Step 1
    if $c > m/2$ then $\texttt{val} \leftarrow val$
    if $c > 3m/4$ then $\texttt{decide} \leftarrow \mathsf{true}, \texttt{lock} \leftarrow \mathsf{true}$
else if $\texttt{phase} \mod 3 = 2$ then                                   ◁ Step 2
    if $c > m/2$ then $\texttt{val} \leftarrow val$
    if $c > 3m/4$ then $\texttt{lock} \leftarrow \mathsf{true}$
else                                                                       ◁ Step 3
```

[8] We note that selecting the median as output is not the only available solution to extract the phase's output. For strong consensus we can extract the plurality (see Remark 1), and for state machine replication we introduce a more refined way to extract output from the king chain (details in Section 4).

```
    if lock = false then val ← $\vec{V}_1$                ◁ Refer to the king chain
    if decide = true then exit ← 1
    ▷ Reset lock for the next iteration
    if decide = false and lock = true then lock ← false
end if
```

We now provide a high-level overview and some intuition about the state update algorithm. In the first phase of an iteration, given phase output vector $\vec{V}$, parties first check if more than $m/2$ chains report the same value *val*. If this is the case, they set their `val` to *val*. Since more than $m/2$ accounts for the majority of the chains, if there exists such value *val* then it will be unique. Further, if in their local view, more than $3m/4$ of the chains report *val*, they set both `decide` and `lock` to `true` and they will decide at the end of iteration. The second phase is almost a repetition of the first one except that in this phase parties will not set `decide` to `true`.

If during the first two phases in an iteration, a party P has never seen more than $3m/4$ of the chains report the same value, P is still "confused" and its internal variable `lock` remains false at the end of the last phase. Under such circumstance, P will refer to the king chain and adopt the median value among the input-messages included—i.e., the first element in phase vector $\vec{V}$. Note that this is different from previous phase-king style constructions, where with deterministic termination, king rotates among $t + 1$ fixed parties (where at least one of them is honest) [BG89, BGP89], while with probabilistic termination, parties first broadcast their `val` and then run an oblivious leader election algorithm to try to agree on an honest king with constant probability [KK06]. In contrast, in our protocol the chain-king is *always* the first chain. Moreover, even though the adversary knows that the first chain is the king, he will not be able to focus on it due to the basic nature of parallel chains. As a result, given that the adversary's power is "diluted," parties agree obliviously with constant probability on the king-chain's value. When the honest parties get lucky, they will start the beginning of the next iteration with a unanimous value in `val`, which guarantees decision; if they do not, they will start the next iteration with a different `val` and they can hope for getting lucky with the next king chain.

Next, we elaborate on the difference between `decide` and `exit` as well as their interaction. As we mentioned earlier, even if parties have decided, they should still participate in the protocol by keeping making RO queries and diffusing blocks with their output value. This is because due to non-simultaneous termination, if parties decide in the current iteration stop from participating in the protocol, then parties that are going to decide in the next iteration would not be able to get enough information since the honest majority condition might be broken. In the classical setting, this is easily circumvented by honest parties who do not receive a message from other parties, reusing their previous message as the current input (cf. [FM88, KK06]). However, in a PoW setting the above strategy is not feasible. Therefore we distinguish the termination of deciding output and mining blocks by using two different variables `decide` and `exit`. Specifically, for any party that decides the output in i-th phase, it should first keep mining for an extra iteration (by setting `exit` to 1 and no longer update `val`, `lock` and `decide`), and then terminate and set `exit` to 0 at the $(i + 3)$-th phase (recall that an iteration consists of 3 phases).[9] After parties set `exit` to 0, they output `val` and exit the protocol.

The ChainKingConsensus protocol. Having presented the various protocol components, we are now

[9] As we show later on in Section 3.4, this termination gap can be reduced to 1 phase by emulating so-called "Bracha termination."

ready to put things together and state what the protocol achieves. During the protocol execution, parties keep updating their local parallel chains and mining their output suggestion. At the end of each phase, they use StateUpdate to update their consensus-related internal variables. Upon setting their `exit` variable to 0, parties terminate the protocol and output `val`.

Protocol ChainKingConsensus

```
▷ We use Blockify to get the string of $m$ block content roots and omit the details on the structure of block content.
Send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and get $(\text{CLOCK-READ}, \text{sid}_C, r)$
if $\texttt{r} = r$ then return                                    ◁ Wait for next round.
$\texttt{r} \leftarrow r, \texttt{phase} \leftarrow \lceil r/\rho \rceil$
if $\texttt{exit} = 0$ then
    Output $\texttt{val}$                                          ◁ Party has terminated.
else
    Fetch information and denote the incoming chains and input-blocks by $\langle \mathbb{C}, denseChains \rangle_1, \ldots, \langle \mathbb{C}, denseChains \rangle_n$ and $\texttt{IB}_1, \ldots, \texttt{IB}_{n'}$
    Add $\langle \mathbb{C}, denseChains \rangle_1, \ldots, \langle \mathbb{C}, denseChains \rangle_n$ to $\texttt{chainBuffer}$
    Add $\texttt{IB}_1, \ldots, \texttt{IB}_{n'}$ to $\texttt{IBBuffer}$
    $\mathbb{C}_{\mathsf{local}} \leftarrow \mathsf{UpdateLocalChain}(\mathbb{C}_{\mathsf{local}}, \langle \mathbb{C}, denseChains \rangle_1, \ldots, \langle \mathbb{C}, denseChains \rangle_n)$
    ▷ Update internal state
    Call StateUpdate                                               ◁ Call Algorithm 5
    ▷ Mine new blocks
    $\texttt{st} \leftarrow \mathsf{Blockify}(\mathbb{C}_{\mathsf{local}}, \texttt{IBBuffer})$
    $\mathbb{C}_{\mathsf{local}}, \texttt{IB} \leftarrow \mathsf{ParallelPoW}(\mathbb{C}_{\mathsf{local}}, \texttt{r}, \texttt{st}, \texttt{val})$     ◁ Call Algorithm 1
    Diffuse $\mathbb{C}_{\mathsf{local}}$
    if $\texttt{IB} \neq \varepsilon$ then Diffuse $\texttt{IB}$
end if
Send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$
```

ChainKingConsensus achieves agreement and validity in an expected-constant number of rounds, and since parties terminate at the end of neighboring phases, it satisfies 3-slack termination (cf. Definition 3). Further, when parties start the protocol with a unanimous input configuration, then they decide at the end of the third phase (except with negligible probability). If they do not start with an unanimous input, then the expected time for decision is $3/(3/4) + 3 = 7$ phases.

Theorem 3. *There exist protocol parameterizations such that* ChainKingConsensus *satisfies agreement, validity and 3-slack termination with expected-constant round complexity.*

Proof. First, we consider the properties of the StateUpdate algorithm.

Claim 1. *There exist protocol parameterizations such that Algorithm 5 satisfies the following properties.*

(a) If at the onset of an iteration, honest parties start unanimously with v, then at the end of the second phase in that iteration, all honest parties set $\texttt{val} = val, \texttt{lock} = \mathsf{true}$ *and* $\texttt{decide} = \mathsf{true}$.

(b) If an honest party P *sets* $\texttt{decide} = \mathsf{true}$ *and* $\texttt{val} = v$ *in an iteration, then all honest parties set* $\texttt{lock} = \mathsf{true}$ *and* $\texttt{val} = v$ *at the end of the second phase in that iteration.*

(c) If an honest party P *sets* $\texttt{lock} = \mathsf{true}$ *and* $\texttt{val} = v$ *in an iteration, then all honest parties set* $\texttt{val} = v$ *at the end of the second phase in that iteration.*

Proof. Suppose ChainKingConsensus is well parameterized such that in any phase, it achieves phase oblivious agreement on at least $(3m/4+1)$ chains.

Regarding Property (a), since honest parties start unanimously with v, by Theorem 2, they reach oblivious argreement on at least $(3m/4+1)$ chains and the majority of the input-blocks are honest hence report the same v. I.e., for any honest party P, at least $(3m/4+1)$ elements in her phase vector $\vec{V}$ are v. Hence P remains `val` as v and set both `lock`, `decide` as `true` at the end of the first phase. Similarly, for the second phase P sees at least $(3m/4+1)$ elements in her phase vector $\vec{V}$ are v so `val` remains unchanged.

For Property (b), if there exists an honest party P setting `decide` to `true` in the first phase, then in P's local view at least $(3m/4+1)$ chains output the same value v. Theorem 2 implies that parties will disagree on at most $m/4$ chains, hence in all other honest parties' local view they see at least $(m/2)$ chains outputting v. I.e., all honest parties set their `val` to v at the end of the first phase. In the second phase, since all parties hold the same value, they see that at least $(3m/4+1)$ elements in their phase vector $\vec{V}$ are v. Therefore, all honest parties set `lock` to `true` and `val` to v at the end of the second phase.

Then we consider Property (c). Since there exists an honest party P setting `lock` to `true` and `val` to v, similar to the arguments in Property (b) we learn that all honest parties will set their `val` to v at the end of that phase. Note that if this is the first phase, then in the second phase they start unanimously so parties will not update their `val`. □

Validity directly follows Property (a).

Regarding agreement, when parties start with different values at the on set of an iteration, either some parties set `decide` to `true` and `val` to v or none of them update `decide`. In the first case, Property (b) implies that all of them set `lock` = `true` and `val` = v so none of the honest party update their value based on the king chain. Hence they start unanimously in the next iteration and since parties that have decided will continue to produce PoWs for one more iteration, all honest parties will set `decide` to `true` and `val` to v in the next iteration. In the second case, note that either some parties have set their `lock` to `true` or none of the honest parties has updated `lock`. If there exists at least one honest party who set her `lock` to `true` and `val` to v, then based on Property (c) all honest parties hold `val` = v. Then, in the third phase, if the execution on king chain is typical, `val` remains as v for all honest parties and they start unanimously in the next iteration which guarantees agreement in the next iteration. Otherwise (also the same for none of the honest parties has updated `lock`), parties start the next iteration with different values and the same argument applies.

The argument for agreement directly implies that honest parties can terminate at most 3 phases apart from each other. We now show that round complexity of ChainKingConsensus is expected-constant. Lemma 10 implies that the protocol parameterization guarantees that the execution on the king chain is typical with probability at least 3/4. I.e., if none of the honest parties decide in an iteration, the probability that all parties start unanimously in the next iteration is at least 3/4. Hence, the number of iterations that parties will all terminate follows the geometric distribution and the expected number of phases is $3/(3/4)+3=7$. □

Remark 1. ChainKingConsensus *also achieves "strong validity" (i.e., that the output equals the input of at least one honest party) if (i) we change the phase output extraction (Algorithm 4) from selecting the median of input-messages to the input-message with the highest plurality; and (ii) the adversarial computational power is bounded by* $t<(1-\delta)n/(|V|-1)$. *(This matches the lower bound in [FG03].)*

Remark 2. *We note that PoW-based* Crusader Agreement *[Dol82] (where parties either output the same value v or $\bot$, and if they start unanimously they output that value) can be achieved in constant time. Specifically, parties run* $\mathsf{ChainKingConsensus}$ *and terminate at the end of first phase. If a party* P *sets her* `decide` *variable to* `true`*,* P *outputs* `val`*; otherwise she outputs* $\bot$*.*

3.4 Fast Sequential Composition

The chain-king consensus protocol presented in Section 3.3 is *one-shot*—i.e., parties start at the same time and terminate at (possibly) different phases. This non-simultaneous termination turns out to be problematic when $\mathsf{ChainKingConsensus}$ is invoked by a high-level protocol, such as MPC or SMR, and where parties need to decide on a series of outputs repeatedly. Given the non-simultaneous termination situation, after the first invocation, parties would not be able to return to the calling high-level protocol synchronously, and in subsequent invocations, $\mathsf{ChainKingConsensus}$ does not by itself provide any security guarantees if parties start at different phases[10]. Ideally, when the same protocol is invoked multiple times, the round complexity should be preserved—i.e., for ℓ sequential invocations, the total running time should be expected $O(\ell)$ rounds.

In the classical distributed computing and cryptographic protocols literature, this is studied as the sequential composition of BA protocols, with positive results: By using so-called "Bracha termination" [Bra84] and super-round expansion [CCGZ16], a BA protocol with probabilistic termination can asymptotically preserve the same round complexity while continuously deciding on a series of outputs.

In this section we show how to achieve fast sequential composition of multiple instances of $\mathsf{ChainKingConsensus}$ by first emulating the Bracha termination strategy on parallel chains, thus enabling parties to terminate in two neighboring phases; then, for later invocations, we introduce a novel "super-phase expansion" protocol that guarantees security under non-simultaneous start while preserving the expected-constant round complexity. Note that, our "super-phase expansion" works for any slack of constant number of rounds, hence Bracha termination is in fact not necessary. Nonetheless, we will first go through this strategy since it helps to achieve a more concise and (practically) efficient result.

Bracha termination. In our one-shot Chain-King Consensus protocol, honest parties might terminate at the end of different but adjacent iterations. We now show how to reduce this slack from one iteration (i.e., 3 phases) to one phase. The high-level idea follows Bracha's original suggestion [Bra84], but we adapt it to the PoW setting.

We first describe this approach in the classical setting (information-theoretic and assuming $n \geq 3t+1$). In Bracha's suggestion, as soon as a party decides on an output v or upon receiving at least $t+1$ messages $(decide, v)$ for the same value v, it sends $(decide, v)$ to all parties. Then, upon receiving $n-t$ messages $(decide, v)$ for the same value v, a party outputs v and terminates.

We now elaborate on our early termination strategy which tries to emulate Bracha's suggestion on parallel chains. Recall that in $\mathsf{ChainKingConsensus}$ the input-block content is its producer's output suggestion val. Here we extend it two types of messages: either output suggestion val, or decide suggestion $(decide, val)$. We say a chain $\mathbb{C}_j^{(i)}$ decides on val if more than half of the input-blocks included in the output generation stage report $(decide, val)$ for the same val. Note that when a chain does not decide on any val, the output extraction algorithm treats all $(decide, val)$ messages the same as val.

Thus, protocol $\mathsf{ChainKingConsensus}$ is modified with the following additional steps:

[10] We note that $\mathsf{ChainKingConsensus}$ can tolerate adversarial pre-mining for up to $\rho_{\mathsf{ref}} \ll \rho$ rounds, details see analysis in Appendix D.2.

- When P's internal variable `decide` is false, P includes only `val` in her input-blocks; when `decide` is true, P mines ($decide$, `val`).
- At the end of any phase, upon observing more than $m/2$ chains decide on val, P sets her `val` to val and `decide` to true.
- At the end of any phase, upon observing more than $3m/4$ chains decide on val, P sets her `val` to val and `exit` to 1.
- After setting `exit` to 1 in the previous step, P continues to mine ($decide$, `val`) for *one more phase* and then set `exit` to 0.

We present the new state update mechanism in Algorithm 6.

Algorithm 6 StateUpdateWithCloseTermination

```
▷ This algorithm is called once in each phase. It directly interacts with internal variables
  val, lock, decide and exit.
if r mod $\rho \neq 0$ then return                                   ◁ Not the end of a phase
if exit = 1 then exit ← 0 and return                          ◁ End of "extra mining phase"
$\vec{V}$ ← ExtractPhaseVector($\mathbb{C}_{\text{local}}$, r, phase)
Let $m$ denote the most frequent element in $V$ and $c$ its frequency
if $m = (decide, val)$ then                              ◁ Check decide for close termination
    if $c > m/2$ then val ← $val$, decide ← true
    if $c > 3m/4$ then val ← $val$, decide ← true, exit = 1
end if
if decide = true then return                  ◁ Stop updating val and lock if party decides
if phase mod 3 = 1 then                                                        ◁ Step 1
    if $c > m/2$ then val ← $val$
    if $c > 3m/4$ then decide ← true
else if phase mod 3 = 2 then                                                   ◁ Step 2
    if $c > m/2$ then val ← $val$
    if $c > 3m/4$ then lock ← true
else                                                                           ◁ Step 3
    if lock = false then val ← $\vec{V_1}$              ◁ Refer to the chain king (first chain)
    if lock = true then lock ← false
end if
```

Theorem 4. *There exist protocol parameterizations such that* ChainKingConsensus *modified with Algorithm 6 satisfies agreement, validity and 1-slack termination with expected-constant round complexity.*

Proof. Let i be the first phase such that there is at least one honest party P set her `decide` to true and `val` to v by observing more than $m/2$ chains reporting $(decide, v)$. We show that there is at least one honest party P′ set her `decide` to true and `val` to v at a phase $i' < i$. Suppose there is no such party P′, then no honest party mine $(decide, v)$ before the end of phase i. I.e., the adversary mines majority of the input-blocks on more than $m/2$ chains, which contradicts the fact that phase oblivious agreement is achieved on phase i (Theorem 2) where at most $m/4$ chains can report a majority of corrupted input-blocks. Combining this with Claim 1 and Theorem 3 we learn that our new protocol with Bracha-style termination, if all parties eventually terminate, achieves agreement and validity.

Next we argue that parties will terminate in neighbour phases using expected constant number phases. Let i be the first phase such that there is at least one honest party P set her `exit` to 1.

We show that all honest parties set `exit` to 1 either in phase i or $i+1$. Suppose — towards a contradiction — there is another party P' set her `exit` to 1 at some round $i' > i+1$ (or, never set `exit` to 1). I.e., P' saw less than $3m/4$ chains outputting $(decide, v)$ in phase $i+1$. Consider P, since P sets `exit` to 1, she saw more than $3m/4$ chains outputting $(decide, v)$ in phase i. Theorem 2 implies that all honest parties would see at least $(m/2+1)$ chains outputting $(decide, v)$ in phase i — i.e., they set `decide` to `true` and `val` to v and start to mine $(decide, v)$. Thus at the beginning of phase $i+1$ all honest parties mine $(decide, v)$. By Theorem 2 they will succeed on at least $(3m/4+1)$ chains and report majority of the input-blocks with $(decide, v)$. This contradicts the assumption that P' saw no more than $3m/4$ chains outputting $(decide, v)$ in phase $i+1$. I.e., the new protocol achieves 1-slack termination.

Finally, consider round complexity. Note that without the Bracha-style termination strategy, all honest parties will set their `decide` to `true` in expected-constant time which is the same as ChainKingConsensus. Upon all honest parties set `decide` to `true`, they will saw at least $(3m/4+1)$ chains outputting $(decide, v)$ and then set `exit` to 1. I.e., in the worst case, this new protocol terminates in one more phase than the number of phases that ChainKingConsensus needs to let all parties decide — which is in expected-constant time. Hence the round complexity of ChainKingConsensus with Bratch-style termination strategy is also expected-constant. □

Slack-tolerant sequential composition of ChainKingConsensus. Now we present how sequential composition works in the permissionless setting. We remark that this is not a straightforward emulation of the super-round expansion technique in the classical literature as in our setting, the adversary effectively has more power in "swinging" the decision of honest parties. We elaborate on the difference between classical round expansion and our novel "super-phase expansion."

In order to perform sequential composition, our protocol should be appropriately adjusted so that we have better quality of phase-oblivious agreement. Recall that Theorem 2 holds for any constant $\beta < 1$. While in one-shot ChainKingConsensus we have protocol parameterizations such that at least three quarters of the chains will reach phase-oblivious agreement, it is possible to achieve that an arbitrary (constant) fraction of chains reach oblivious agreement. One consequence is that we will get a slow-down on the length of a phase in terms of number of rounds; the asymptotic result (i.e., expected-constant number of rounds), however, is preserved.

Furthermore, consider any $n \in \mathbb{N}^+$ consecutive phases in an execution of the protocol. If β fraction of the chains have reached oblivious agreement in one phase, then at least $[1 - n(1-\beta)]$ fraction of chains reach oblivious agreement over all n phases. By appropriately choosing n and β, we get the following property: In any n consecutive phases at least three-quarters of the chains achieve phase-oblivious agreement over all phases. For example, when $\beta = 95\%$ and $n = 3$, for any 3 consecutive phases, honest parties obliviously agree on at least three quarters of the chains. As a result, we have the following corollary to Theorem 2:

Corollary 5 (Multi-phase oblivious agreement). *There exist protocol parameterizations such that the following properties hold. Consider $n \in \mathbb{N}^+$ consecutive phases $i, i+1, \ldots, i+n-1$, $i \geq 1$. Let $\mathbb{C}, \mathbb{C}'$ denote the parallel chains held by two honest parties P, P' at round r, r', respectively, after the $(i+n-1)$-th phase (i.e., $\min\{r, r'\} > (i+n-1)\rho$). Then there exists a subset $S \subseteq \{1, 2, \ldots, m\}$ of size $|S| > 3m/4$ such that for any $j \in S$ and any $k \in \{i, i+1, \ldots, i+n\}$, the following two properties hold on chains $\mathcal{C} = \mathbb{C}_j^{(k)}$ and $\mathcal{C}' = \mathbb{C}_j'^{(k)}$:*

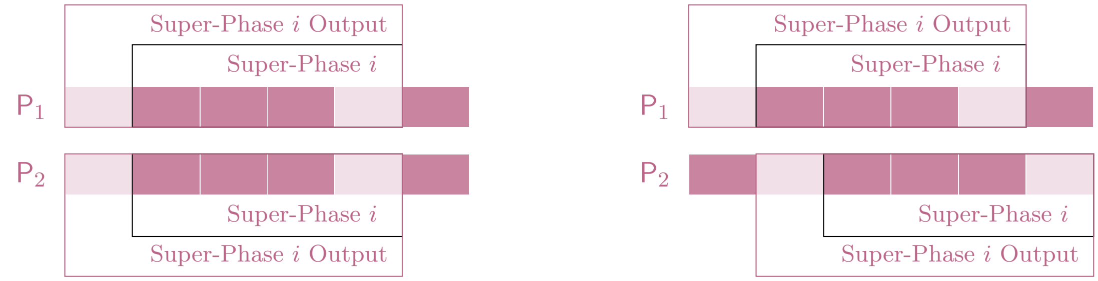
- **Agreement.** $\mathcal{C}^{\lceil \rho_{\mathsf{ref}}} = \mathcal{C}'^{\lceil \rho_{\mathsf{ref}}}$.
- **Honest input-block majority.** *For all input blocks included in the output generation stage of $\mathcal{C}$ and $\mathcal{C}'$, more than half of them are produced by honest parties.*

Proof. Fix n phases $\{i, i+1, \ldots, i+n-1\}$. Suppose the protocol is appropriately parameterized such that phase oblivious agreement is achieved on more than $\beta = 1 - 1/(4n) + \epsilon$ fraction of chains in each phase. Let A denote the (bad) event on two chains $\mathcal{C}$ and $\mathcal{C}'$ such that either $\mathcal{C}^{\lceil \rho_{\mathsf{ref}}} \neq \mathcal{C}'^{\lceil \rho_{\mathsf{ref}}}$ or more than half of the input blocks in $\mathcal{C}$ are produced by the adversary.

Suppose towards a contradiction, there exist a subset $S \subseteq \{1, 2, \ldots, m\}$ of size at least $m/4$ (i.e., $1/4$ fraction) such that $\exists k \in \{i, i+1, \ldots, i+n-1\}, j \in S$ and $\mathcal{C} = \mathbb{C}_j^{(k)}$ and $\mathcal{C}' = \mathbb{C}_j'^{(k)}$, A happens on $\mathcal{C}$ and $\mathcal{C}'$. Then, due to Pigeonhole principle, there exists $k^* \in \{i, i+1, \ldots, i+n-1\}$ such that for a subset $S' \subseteq \{1, 2, \ldots, m\}$ of size at least $(1/4)/n \geq 1/(4n)$ and $\mathcal{C} = \mathbb{C}_j^{(k^*)}$ and $\mathcal{C}' = \mathbb{C}_j'^{(k^*)}$, A happens on $\mathcal{C}$ and $\mathcal{C}'$. However, since $\beta = 1 - 1/(4n) + \epsilon$ according to Theorem 2, in each phase the fraction of chains such that oblivious agreement fails is bounded by $1/(4n)$, which contradicts the number of failing repetitions in the k-th phase. □

Regarding input messages, we also require that parties attach messages indicating the index of invocations and the index and steps of iterations in their input messages. That is, a valid input message in sequential composition would be of the form "This is the i-th invocation, j-th iteration and k-th phase, and my output suggestion is val." We omit the details of encoding such messages. Moreover, in some "dummy" phases, parties are allowed to send dummy suggestion $\perp$ that contains no information.

Given that parties can terminate and start within two neighboring phases, our super-phase expansion (which will be adopted in the second and subsequent invocations) replaces the original (aligned) phase to four (possibly unaligned) phases "input-input-input-dummy." I.e., parties report their suggested output during the first three phases in their local view, and leave the last phase dummy. See Figure 3 for an illustration of an aligned super-phase and an unaligned one.



(a) aligned super phases

(b) 1-slack super phases

Figure 3: *Illustration of the super-phase expansion and how parties extract the super-phase output.* ■ *represents the phase where a party mines input messages with her output suggestion, and* ▢ *is the dummy phase. The i-th super phase is represented by* □*; and the associated phases to extract output are depicted with* □.

The decision process works as follows. When a party P reaches the end of a super-phase (in her local view), she decides an output (a vector of size m) for this super-phase based on the output of *five* previous (normal) phases (i.e., starting from one normal phase before the current super-phase (see the illustration of "Super-Phase Output" in Figure 3). For each chain, P does the following. Recall that parties are allowed to report $\perp$. When there is a (normal) phase such that more than half of the input-blocks report $\perp$, then we say this phase reports $\perp$. Otherwise, pick the median value of all non-$\perp$ values (after sorting) as the output of this phase. The decisions are as follows:

- When there are more than two phases that output non-$\perp$ values, output the value in the *second* phase.

– When there is one phase outputting a value val, output val for this super-phase.

Next, we provide some intuition on why adding a dummy phase at the end of a super-phase is necessary. When honest parties do not start unanimously with the same value, the adversary can join forces with those late honest parties in their last phase so that the view of honest parties are not consistent (because the parties that terminate early should make a decision when other honest parties have not yet finished their current super phase). With dummy rounds, all honest parties share a consistent view under multi-phase oblivious agreement, hence guaranteeing agreement and validity.

Moreover, keeping including the output suggestion for 3 consecutive normal phases is also necessary. For a concrete example, suppose the underlying one-shot consensus protocol achieves 1-slack termination, and the honest computational power accounts for 60% of the total (i.e., the adversary owns 40%) and honest parties are equally divided into two subsets, starting from two neighbouring phases. In other words, we have parties starting and terminate early (resp., late) that accounts for 30% of computational power. Then, if parties include their output suggestion for only two phases, the adversary can refrain from mining in the first normal phase of the early parties, and join forces with the late parties in their second normal phase but inject a non-honest input. In such a case, even if parties start unanimously with v the output of this chain under multi-phase oblivious agreement will not be v (as 40% is greater than 30%), thus violating the validity property of consensus. With 3 consecutive mining normal phases, at least two of them will overlap, an adopting an output in the second non-$\perp$ phase will be safe.

The super-phase expansion can be easily adapted from a 1-slack non-simultaneous start to c-slack, for any constant c. We briefly describe the most naïve treatment. For c-slack termination, a super-phase consists of $(3c + 1)$ rounds where parties keep mining their suggested output for this super-phase in the first $(2c + 1)$ normal phases, and sending $\perp$ in the last c phases (i.e., c dummy phases in total). To extract the output of a chain, $(4c + 1)$ rounds are considered. Similar to the treatment in 1-slack termination, when there are more than $(c + 1)$ phases outputting a non-$\perp$ value, take the output from the $(c + 1)$-th phase; otherwise, output the value in the last phase. At a high level, with c-slack termination parties will share at least $(c + 1)$ overlapping phases (or any phase, as this implies some bad event happened so we do not expect an agreement). Under multi-phase oblivious agreement, the adversary can join forces with early honest parties for at most c phases. Hence, taking output from the $(c+1)$-th phase will result in a value from the overlapping normal phases where all honest parties mine their suggested values.

By adopting the 1-slack termination technique and super-phase expansion, we get the following theorem for sequential composition of ℓ invocations of ChainKingConsensus.

Theorem 6. *There exist protocol parameterizations such that the sequential composition of ℓ invocations of* ChainKingConsensus *satisfies agreement and validity on each invocation, and the round complexity is expected $O(\ell)$.*

Proof. Note that Theorem 4 implies that parties terminate at adjacent phases in the first invocation, we prove this theorem by induction.

Suppose that the i-th protocol invocation achieves agreement, validity and 1-slack termination in expected-constant time, we show these properties still holds in the $(i + 1)$-th invocation.

To argue for the protocol invocations using super-phase expansion, based on Claim 1, we consider the following two properties. First, when parties start unanimously with val, they saw more than $3m/4$ chains report val. Second, when parties refer to the king chain, with constant probability their view is consistent, and the majority of the input-blocks in king chain are produced by honest parties.

To prove the first property, suppose our protocol is well parameterized such that multi-phase oblivious agreement is achieved for any six consecutive phases $i, i+1, \ldots, i+5$ due to Corollary 5. Consider the j-th chain where typical execution holds for all six phases. We consider parties start in two neighbour normal phases $i+1$ and $i+2$. Since phase oblivious agreement holds in phase i and $i+5$, the adversary cannot let the j-th chain output a valid value in this super phase. Since all honest parties will join force in phase $i+2, i+3$, by adopting the output in the second phase (which outputs a valid value) in their local view, either all honest parties adopt the output of phase $i+2$ or all honest parties adopt the output of phase $i+3$. Since this holds on more than $3m/4$ chains, when parties start unanimously with val they saw more than $3m/4$ chains outputting val; and when there is an honest party saw more than $3m/4$ chains outputting val, all honest parties saw more than $m/2$ chains outputting val.

Regarding the second property, note that the probability that phase oblivious agreement holds on king chain for six consecutive rounds is more than 3/4. Using a similar argument we show that with constant probability, every honest party shares a consistent view of the king chain and the majority of input-blocks are produced by honest parties. □

4 Application: Fast State Machine Replication

We now show how to adapt the sequential composition approach in Section 3.4 to implement a state machine replication (SMR) protocol. Our resulting protocol achieves both Consistency and expected-constant-time Liveness for all types of transactions (including conflicting ones). Namely, for any transaction `tx`, when `tx` is diffused to all honest participants (miners), it takes in expectation a constant number of rounds to get settled into the immutable final ledger.

We first give our definition of SMR, and elaborate on why fast SMR protocol cannot be directly derived from the sequential composition of multi-valued Chain-King Consensus. Then, in Section 4.1 we propose a new method that introduces randomness to the output of the king chain and helps circumvent the above problem while preserving expected-constant settlement time for all types of transactions. Finally, in Section 4.2 we show how a third party observer, joining in the middle of the protocol, can catch up with honest parties and learn the state of the ledger.

SMR background. State machine replication (SMR) is the problem of distributing the operation of a state machine across a set of replicas so that the operation of the machine is resilient to failure of a subset of the replicas. This concept was originally described in [Lam78], and later further elaborated on by Schneider [Sch90] where a high-level description of SMR was provided. Blockchain protocols, and in particular Bitcoin's [Nak08] have renewed interest in SMR definitions and constructions, as they can be seen as a way to realize SMR in a setting where there is no predetermined set of replicas. This has been studied and formalized in a series of works (e.g., [GKL14, PSs17, GK20]).

We now give a concise definition of SMR. A number of n servers, a subset $\mathcal{H}$ of which is assumed to be non-faulty, maintain a log of transactions, denoted Log. The log of each server also timestamps each transaction. The notation $\mathsf{Log}_i[t]$ denotes the log of the server P_i up to time t. Furthermore, it is assumed that each server has a buffer for incoming transactions, denoted by $I_i[t]$, that are valid with respect to its view (invalid transactions are dropped). Finally, and for simplicity, assume that all well-formed transactions are admissible in the log. In SMR, the following two conditions must be satisfied:

- **Consistency:** $\forall \mathsf{P}_i, \mathsf{P}_j \in \mathcal{H}$ (where not necessarily $i \neq j$) and t, t' it holds that $\mathsf{Log}_i[t] \preceq \mathsf{Log}_j[t']$ or $\mathsf{Log}_j[t'] \preceq \mathsf{Log}_i[t]$.

- **Liveness:** There is a parameter $u \in \mathbb{N}$ for which the following holds: $(\forall \mathsf{P}_i \in \mathcal{H} : \mathtt{tx} \in I_i[t]) \implies \mathtt{tx} \in \mathsf{Log}_i[t+u]$.

Typically, the Liveness parameter u is a pre-defined value according to the protocol parameterization. It is natural to extend the notion and allow u to be a random variable with a distribution that depends on the specific parameterization. I.e., given a transaction tx appearing in all honest buffers at time t, the probability that it is included in all honest logs at time $t+u$ shares the same distribution with u. In our protocol, we achieve u with a geometric distribution, hence the time for tx to get installed in the immutable ledger is expected-constant.

Note that there are more properties of interest for SMR, such as *observability*, which is the requirement that a third party observer be capable of interpreting correctly the current state of the ledger by inspecting the logs of the servers.

4.1 From Sequential Composition to State Machine Replication

An SMR protocol accepts a batch of transactions as input. While we omit here the details on the particular form of the transactions, we note that the input domain is of exponential size. Thus, "strong validity" (i.e., the requirement that output is at least one honest input) is impossible even if the adversary only controls a tiny fraction of the computational power (cf. Remark 1). Also note that a unanimous start would rarely happen given that the adversary can collude with clients and send different or conflicting transactions to different parties. Therefore, if we follow the method from Section 3.3—e.g., to apply the median or plurality rule—to select the output on the king chain, as long as the adversary carefully selects the set of transactions, he can always make his input batch be selected as the output. By carefully constructing such transaction batches, the adversary will be able to indefinitely delay the confirmation of any honest transaction tx, even if tx has been provided to all honest participants.

Proof-of-Work as a lottery. We now present a new construction that helps preventing the adversarial control described above when parties do not start unanimously. In a nutshell, when a party P is still "confused" at the end of an iteration (i.e., her internal variable lock remains false), P adopts the output of the king chain as her new input, which is the (valid) input-block reported in the first chain, with the *smallest* block hash. When the honest parties obliviously agree on the king chain (which happens with constant probability), they will refer to the same block. Notice that honest parties make more RO queries than the corrupted parties. The following lemma shows that with probability (roughly) one half, the input-block with smallest block hash is produced by an honest party.

Lemma 7. *Let $h = \mathsf{poly}(\kappa)$ and $t = \mathsf{poly}(\kappa)$ denote the number of random oracle queries made by honest and corrupted parties, respectively. Under honest majority assumption ($h > t$), the probability that the smallest RO output is from an honest query is $1/2 - \mathsf{negl}(\kappa)$.*

Proof. Suppose $X_1, X_2, \ldots, X_h$ are h i.i.d. uniform random variables and let $X = \min\{X_1, X_2, \ldots, X_h\}$ and $X' = \min\{X_1, X_2, \ldots, X_t\}$ where $t < h$. Similarly, suppose $Y_1, Y_2, \ldots, Y_t$ are t i.i.d. uniform random variables and let $Y = \min\{Y_1, Y_2, \ldots, Y_t\}$. Let $s = \kappa/m = \omega(\log \kappa)$ denote the length of

RO output with respect to a single chain. We have $\mathbf{Pr}[X < Y] \geq \mathbf{Pr}[X' < Y]$.

$$\begin{aligned}\mathbf{Pr}[X < Y] &= \sum_{i=0}^{2^s-1} (\mathbf{Pr}[X \leq i] - \mathbf{Pr}[X \leq (i-1)]) \cdot \mathbf{Pr}[Y > i] \\ &< \sum_{i=0}^{2^s-1} \Big(\big[1 - (\mathbf{Pr}[X_1 > i])^t\big] - \big[1 - (\mathbf{Pr}[X_i > i-1])^t\big] \Big) \cdot \mathbf{Pr}[Y > i] \\ &= \sum_{i=0}^{2^s-1} \mathbf{Pr}[X' = i] \cdot \mathbf{Pr}[Y > i] = \mathbf{Pr}[X' < Y].\end{aligned}$$

Since X' and Y are two i.i.d. random variables, we have $\mathbf{Pr}[X' < Y] = \mathbf{Pr}[X' > Y]$. Recall that $t = \mathsf{poly}(\kappa)$, let C denote the event that a collision happens with $2t$ RO queries. We have $\mathbf{Pr}[C] \leq (2t)^2 2^{-s} = \exp(-\Omega(\mathsf{polylog}\kappa + \ln t)) = \mathsf{negl}(\kappa)$ — i.e., a collision happens with negligible probability. Notice that $X' = Y$ implies a collision, we have $\mathbf{Pr}[X' = Y] < \mathbf{Pr}[C]$. Hence $\mathbf{Pr}[X < Y] > \mathbf{Pr}[X' < Y] = \mathbf{Pr}[X' > Y] = 1/2 - \mathsf{negl}(\kappa)$. □

Fast state machine replication. We are now ready describe our SMR protocol. At a high level, it can be viewed as the sequential composition of Chain-King Consensus, equipped with a new phase output extraction algorithm, described as follows. When parties are extracting output in the first and second phase of an iteration, for each chain they will output v if the majority of input-blocks is v; otherwise they will output $\perp$ (in this way, the adversary cannot let parties decide on a batch of transactions that is not an honest input in the first two stages). When they are in the third phase (i.e., that's when the "confused" parties listen to the king chain) they will output the input-block with the smallest hash value.

Theorem 8. *There exist protocol parameterizations such that the sequential composition of Chain-King Consensus with the minimum-PoW king selection rule satisfies Consistency and expected-constant Liveness.*

Proof. Consistency is straightforward. For each invocation of chain-king consensus, the output is a batch of transactions. After linearization, these transactions are appended to the ledger and are considered as settled. These transactions will be at the same position in the ledger of all honest parties in the same or neighbouring phases.

Regarding Liveness, suppose a transaction `tx` is diffused to all honest parties at the onset of an invocation of chain-king consensus. In the first and second phase in an iteration, the adversary cannot let parties decide on a set of transactions that does not include `tx`, as `tx` is in all honest input and the adversary cannot produce the majority of the blocks in more than $m/2$ chains. With constant probability, every party shares a consistent view of the king chain and they will then decide. Further, with constant probability the input-block with smallest hash in the king chain is produced by an honest party, which includes `tx`. When the honest parties are unlucky to lose the race of producing smallest hash, `tx` remains in the pending transaction pool of all honest parties and is included in the input for the next invocation of chain-king consensus. Given that each invocation has expected-constant round complexity, and the probability that `tx` is in the output is roughly $1/2$, we learn that the time for `tx` to get settled in the ledger is also expected-constant. □

4.2 Bootstrapping from the Genesis Block

In this section, we focus on the *observability* property of our SMR protocol. Recall that in Section 3.2, we stated that a full agreement on all parallel chains in the previous phase is impossible, and parties that join at a specific phase cannot learn the previous execution by "tracing back" using cross-chain reference. Thus, it becomes challenging or even impossible for a passive observer to join the protocol in the middle of the execution. To solve this, in this section we slightly modify our Chain-King Consensus protocol and design a bootstrapping algorithm for fresh parties to synchronize state with all honest parties. Note that the design of a bootstrapping procedure to let fresh parties join is also an essential building block for protocols that support dynamic participation.

When a fresh party $\mathsf{P}_{\mathsf{new}}$ joins, $\mathsf{P}_{\mathsf{new}}$ has no knowledge about the protocol execution except for the CRS and global time (recall that we assume synchronous processors). To become synchronized and learn the ledger state, $\mathsf{P}_{\mathsf{new}}$ needs to bootstrap by passively listening to the protocol. We highlight that, in order for $\mathsf{P}_{\mathsf{new}}$ to synchronize with other honest parties (i.e., achieving phase oblivious agreement), $\mathsf{P}_{\mathsf{new}}$ needs to run a bootstrapping procedure which lasts for a constant number of rounds (precisely ρ rounds).

In order to let fresh parties join the protocol, we modify our Chain-King Consensus protocol as follows. In the i-th phase ($i > 1$), concatenated with the consensus-related input message, parties also include the fresh randomness extracted from their local chains in the $(i-1)$-th phase. More specifically, they extract the hash of the last block in the output generation stage on each chain in the $(i-1)$-th phase of $\mathbb{C}_{\mathsf{local}}$, assemble them as a κ-bit string and append it to the input-block content. For chains where a typical execution holds, honest parties adopt the same block hash. Next, in i-th phase, a Crusader Agreement is run on the block hash of each chain in the $(i-1)$-th phase (recall from Remark 2 that a single phase suffices to serve as a Crusader Agreement protocol). I.e., for the j-th chain with a typical execution, parties agree on a unique block hash that is the same as their local $\mathbb{C}_j^{(i-1)}$, and for other chains, all parties either output the same hash or $\perp$.

Thus, when a fresh party $\mathsf{P}_{\mathsf{new}}$ joins the protocol, she first passively listens to the protocol for ρ rounds so that she observes the end of a phase, say phase i. Our chain selection rule (Algorithm 3) guarantees that $\mathsf{P}_{\mathsf{new}}$ has parallel chains in phase i that obliviously agree with other honest parties on more than $3m/4$ chains (recall Theorem 2). Now, $\mathsf{P}_{\mathsf{new}}$ can "trace back" all the chains where typical execution holds by using the fresh randomness included in the current phase; and iterate them phase-by-phase. Specifically, when $\mathsf{P}_{\mathsf{new}}$ is at the end of phase i, she runs Algorithm 7 to extract the hashes of dense chains in the previous phase and use them to form her local chain $\mathbb{C}_{\mathsf{local}}^{(i-1)}$. For instance, consider the j-th chain in the $(i-1)$-th phase. If on more than $3m/4$ chains in phase i, a majority of the input blocks report fresh randomness that matches a chain $\mathcal{C} \in \mathtt{denseChains}[i-1][j]$, then $\mathsf{P}_{\mathsf{new}}$ will select $\mathcal{C}$ and add it as the j-th chain in $\mathbb{C}_{\mathsf{local}}^{(i-1)}$. If no such chain exists, $\mathsf{P}_{\mathsf{new}}$ will randomly pick a chain or just leave it empty.

Algorithm 7 JoiningProcedure

```
for $i = 1$ to $\rho$ do
    Fetch information and dentoe the incoming chains and input-blocks by $\langle \mathbb{C}, denseChains \rangle_1, \ldots, \langle \mathbb{C}, denseChains \rangle_n$ and $\mathtt{IB}_1, \ldots, \mathtt{IB}_{n'}$
    Add $\langle \mathbb{C}, denseChains \rangle_1, \ldots, \langle \mathbb{C}, denseChains \rangle_n$ to chainBuffer
    Add $\mathtt{IB}_1, \ldots, \mathtt{IB}_{n'}$ to IBBuffer
    $\mathbb{C}_{\mathsf{local}} \leftarrow \mathsf{UpdateLocalChain}(\mathbb{C}_{\mathsf{local}}, \langle \mathbb{C}, denseChains \rangle_1, \ldots, \langle \mathbb{C}, denseChains \rangle_n)$
    if $\mathtt{r} \mod \rho = 1$ then                          ◁ Retrieve all previous phase
        for $i$ from $\mathtt{phase} - 1$ to 1 do
            for $j = 1$ to $m$ do
                $\mathcal{C} \leftarrow \mathbb{C}_j^{(i+1)}$
                Initialize $\vec{h}$ to an empty array
                for $q = 1$ to $m$ do
                    Initialize $\vec{M}$ to an empty array
                    for $\mathcal{B} \in \mathcal{C}$ and $tp \cdot \rho - (\rho_{\mathsf{output}} + \rho_{\mathsf{ref}}) < \mathsf{TS}(\mathcal{B}) \leq tp \cdot \rho - \rho_{\mathsf{ref}}$ do
                        Parse chain pointer to last phase as $h'$
                        Append $[h']_{q\sim m}$ to $\vec{M}$
                    end for
                    Sort $\vec{M}$ then append $\mathsf{med}(\vec{M})$ to $\vec{h}$
                end for
                Let $h$ denote the most frequent element in $\vec{h}$ and $c$ its frequency
                if $c > 3m/4$ and $\exists \mathcal{C} \in \texttt{denseChains}[i][j]$ such that $[\text{head}(\mathcal{C}^{\lceil \rho_{\mathsf{ref}}})]_{j\sim m} = h$ then
                    Add $\mathcal{C}$ as the $j$-th chain in $\mathbb{C}^{(i)}_{\mathsf{local}}$
                else
                    Choose a chain $\mathcal{C} \in \texttt{denseChains}[i][j]$ randomly as the $j$-th chain in $\mathbb{C}^{(i)}_{\mathsf{local}}$ (or if no dense chain, select one from incoming $\mathbb{C}^{(i)}_{j}$ randomly)
                end if
            end for
        end for
    end if
end for
Call StateUpdate on each phase to build the ledger
Set synchronized with all honest parties
```

Note that the security of both Chain-King Consensus and Crusader Agreement only rely on the consistent view of chains where typical execution holds; hence, at the end of the joining procedure, $\mathsf{P}_{\mathsf{new}}$ achieves phase oblivious agreement with all honest parties. As a result, $\mathsf{P}_{\mathsf{new}}$ can reconstruct the entire execution and update her internal state to build the whole ledger.

Acknowledgements

Juan Garay's research has been supported in part by NSF grants no. 2001082 and 2055694, and by the Algorand Centres of Excellence programme managed by the Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation. He also thanks Karim Eldefrawy, Ben Terner and Vassilis Zikas for useful discussions on the topic. Yu Shen's research has been supported by Input Output (iohk.io) through their funding of the University of Edinburgh Blockchain Technology Lab.

References

[AD15] Marcin Andrychowicz and Stefan Dziembowski. PoW-based distributed cryptography with no trusted setup. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 379–399, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[BG89] Piotr Berman and Juan A. Garay. Asymptotically optimal distributed consensus (extended abstract). In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca, editors, *Automata, Languages and Programming*, pages 80–94, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 410–415, Research Triangle Park, NC, USA, October 30 – November 1, 1989. IEEE Computer Society Press.

[BKT$^+$19] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 585–602, London, UK, November 11–15, 2019. ACM Press.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[Bra84] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *3rd ACM Symposium Annual on Principles of Distributed Computing*, pages 154–162, Vancouver, BC, Canada, August 27–29, 1984. Association for Computing Machinery.

[Can00a] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

[Can00b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `https://eprint.iacr.org/2000/067`.

[CCGZ16] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 240–269, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[DEF$^+$22] Poulami Das, Lisa Eckey, Sebastian Faust, Julian Loss, and Monosij Maitra. Round efficient byzantine agreement from VDFs. Cryptology ePrint Archive, Report 2022/823, 2022. `https://eprint.iacr.org/2022/823`.

[Dol82] Danny Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[DRS90] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, Oct 1990.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[EFL17] Lisa Eckey, Sebastian Faust, and Julian Loss. Efficient algorithms for broadcast and consensus based on proofs of work. Cryptology ePrint Archive, Report 2017/915, 2017. `https://eprint.iacr.org/2017/915`.

[FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In Elizabeth Borowsky and Sergio Rajsbaum, editors, *22nd ACM Symposium Annual on Principles of Distributed Computing*, pages 211–220, Boston, MA, USA, July 13–16, 2003. Association for Computing Machinery.

[FGKR18] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, Report 2018/1119, 2018. `https://eprint.iacr.org/2018/1119`.

[FGKR20] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ledger combiners for fast settlement. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 322–352, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th Annual ACM Symposium on Theory of Computing*, pages 148–161, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[GK20] Juan A. Garay and Aggelos Kiayias. SoK: A consensus taxonomy in the blockchain era. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of *Lecture Notes in Computer Science*, pages 284–318, San Francisco, CA, USA, February 24–28, 2020. Springer, Heidelberg, Germany.

[GKL14] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Report 2014/765, 2014. `https://eprint.iacr.org/2014/765`.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 291–323, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[GKLP18] Juan A. Garay, Aggelos Kiayias, Nikos Leonardos, and Giorgos Panagiotakos. Bootstrapping the blockchain, with applications to consensus and fast PKI setup. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 10770 of *Lecture Notes in Computer Science*, pages 465–495, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.

[GKO$^+$20] Juan A. Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 129–158, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[GM93] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement in t+1 rounds. In *25th Annual ACM Symposium on Theory of Computing*, pages 31–41, San Diego, CA, USA, May 16–18, 1993. ACM Press.

[GMPY11] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. *Journal of Cryptology*, 24(4):615–658, October 2011.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, Jul 1978.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, Jul 1982.

[MR22] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2295–2308, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

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

[PS17] Rafael Pass and Elaine Shi. FruitChains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *36th ACM Symposium Annual on Principles of Distributed Computing*, pages 315–324, Washington, DC, USA, July 25–27, 2017. Association for Computing Machinery.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, apr 1980.

[PSs17] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science*, pages 403–409, Tucson, Arizona, November 7–9, 1983. IEEE Computer Society Press.

[Sch90] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, dec 1990.

[TC84] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

A Related Work (Cont'd)

In the naïve generalization from 2×1 PoW to $m\times 1$ PoW, in order to achieve independence among parallel mining procedures, the random oracle output is split into m non-overlapping segments and each segment is assigned to a unique procedure. We remark that in the case of parallel chains, the number of chains becomes the security parameter, and hence m should be chosen sufficiently large to provide security guarantees. In all existing parallel-chain schemes [BKT+19, FGKR20], a number $m = \Theta(\kappa)$ of parallel chains is adopted.

Notice that since the output of random oracle is a string of length κ, it is infeasible to directly run $m = \Theta(\kappa)$ chains in parallel, for two reasons. On one hand, with $m = \Theta(\kappa)$ repetitions, only a constant number of bits can be allocated to each chain thus upper-bounding the total number of participating parties; on the other hand, a constant number of bits implies a constant output space for the random oracle, where collisions can be found if the execution runs for $L = \mathsf{poly}(\kappa)$ steps.

To solve this, in [FGKR20] a new scheme for $m{\times}1$ PoW is proposed by partitioning the output string into two segments of $\kappa/2$ bits. The first segment indicates whether this query is successful; and the second segment decides on which chain this PoW message is valid. I.e., one query can succeed on at most one chain (while in the ideal scheme success on multiple chains is possible). This scheme achieves parallel chain sub-independence, and the statistical distance from the ideal parallel random oracles is bounded by the square of the success probability of a single random oracle query. Finally, the $m{\times}1$ scheme presented in [BKT^{+}19] checks if the numeric value of the random oracle output is within a specific range and hence decide on which chain it succeeds. As such, this scheme can only succeed in producing blocks on one chain and therefore does not provide full independence.

B Models and Preliminaries (Cont'd)

Clock functionality. We adopt $\mathcal{G}_{\text{CLOCK}}$ (cf. [KMTZ13]) to model synchronous processors. In a nutshell, $\mathcal{G}_{\text{CLOCK}}$ internally maintains a round variable τ which is only updatable when all parties send it the CLOCK-UPDATE command. Whenever a party P is activated, P sends a CLOCK-READ message to check the current round. When round proceeds, P executes the protocol and send CLOCK-UPDATE after it completes all computations; if not, P does nothing and wait for the next activation. By interacting with $\mathcal{G}_{\text{CLOCK}}$, parties are aware that they proceed in synchronized rounds.

Functionality $\mathcal{G}_{\text{CLOCK}}$

The functionality manages the set $\mathcal{P}$ of registered identities, i.e., parties $\mathcal{P} = (\text{pid}, \text{sid})$. It also manages the set F of functionalities (together with their session identifier). Initially, $\mathcal{P} \leftarrow \emptyset$ and $F \leftarrow \emptyset$.
For each session sid the clock maintains a variable τ_{sid}. For each identity $\mathsf{P} = (\text{pid}, \text{sid}) \in \mathcal{P}$ it manages variable d_{P}. For each pair $(\mathcal{F}, \text{sid}) \in F$ it manages variable $d(\mathcal{F}, \text{sid})$ (all integer variables are initially 0).

Synchronization:

- Upon receiving $(\text{CLOCK-UPDATE}, \text{sid}_C)$ from some party $\mathsf{P} \in \mathcal{P}$ set $d_{\mathsf{P}} \leftarrow 1$; execute *Round-Update* and forward $(\text{CLOCK-UPDATE}, \text{sid}_C, \mathsf{P})$ to $\mathcal{A}$.
- Upon receiving $(\text{CLOCK-UPDATE}, \text{sid}_C)$ from some functionality $\mathcal{F}$ in a session sid such that $(\mathcal{F}, \text{sid}) \in F$ set $d(\mathcal{F}, \text{sid}) \leftarrow 1$, execute *Round-Update* and return $(\text{CLOCK-UPDATE}, \text{sid}_C, \mathcal{F})$ to this instance of $\mathcal{F}$.
- Upon receiving $(\text{CLOCK-READ}, \text{sid}_C)$ from any participant (including the environment on behalf of a party, the adversary, or any ideal—shared or local—functionality) return $(\text{CLOCK-READ}, sid_C, \tau_{\text{sid}})$ to the requestor (where sid is the sid of the calling instance).

Procedure Round-Update: For each session sid do: If $d(\mathcal{F}, \text{sid}) = 1$ for all $\mathcal{F} \in F$ and $d_{\mathsf{P}} = 1$ for all honest parties $P = (\cdot, \text{sid}) \in \mathcal{P}$, then set $\tau_{\text{sid}} \leftarrow \tau_{\text{sid}} + 1$ and reset $d(\mathcal{F}, \text{sid}) \leftarrow 0$ and $d_{\mathsf{P}} \leftarrow 0$ for all parties $\mathsf{P} = (\cdot, \text{sid}) \in \mathcal{P}$.

Random oracle functionality. By convention, we model the hash function used to generate PoW as a random oracle; this is captured by the functionality $\mathcal{F}_{\text{RO}}$. $\mathcal{F}_{\text{RO}}$ internally maintains an updatable table H with output length the same as security parameter κ. Upon receiving a query $(\textsc{eval}, \mathrm{sid}, x)$, if no pair of the form $(x, \cdot)$ is in H, a value y is chosen uniformly at random from $\{0,1\}^\kappa$ and returned to the party ($\mathcal{F}_{\mathrm{RO}}$ also updates $H(x) = y$). If $H(x) \neq \bot$ (i.e., x has been queried before), the corresponding y is returned.

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

The functionality is parameterized by the security parameter κ. It maintains a dynamically updatable function table H where $H[x] = \bot$ denotes the fact that no pair of the form $(x, \cdot)$ is in H. Initially, $H = \emptyset$.

- Upon receiving $(\textsc{eval}, \mathrm{sid}, x)$ from some party $\mathsf{P} \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted P), do the following:
  1. If $H[x] = \bot$ sample a value y uniformly at random from $\{0,1\}^\kappa$ and set $H[x] \leftarrow y$.
  2. Return $(\textsc{eval}, \mathrm{sid}, x, H[x])$ to the requestor.

Note that with regards to bounding access to real-world resources, functionality $\mathcal{F}_{\mathrm{RO}}$ as defined fails to limit the adversary on making a certain number of queries per round. Hence, we adopt a functionality wrapper [BMTZ17, GKO$^+$20] $\mathcal{W}(\mathcal{F}_{\mathrm{RO}})$ that wraps the corresponding resource to capture such restrictions.

Functionality $\mathcal{W}(\mathcal{F}_{\mathrm{RO}})$

The wrapper functionality is parameterized by a set of parties $\mathcal{P}$, and an upper bound t which restricts the $\mathcal{F}$-evaluations of all corrupted party per round. The functionality manages the variable τ (positive integer or $\bot$) and the current set of corrupted miners $\mathcal{P}'$. It also manages variable $t_{\mathcal{A}}$. Initially, $\tau = \bot$.

General: The wrapper stops the interaction with the adversary as soon as the adversary tries to exceed its budget of t queries per nominal round.

Relaying inputs to the random oracle:

- Upon receiving $(\textsc{eval}, \mathrm{sid}, x)$ from $\mathcal{A}$ on behalf of a corrupted party $P \in \mathcal{P}'$, if $\tau = \bot$, forward the request to $\mathcal{F}_{\mathrm{RO}}$ and return to $\mathcal{A}$ whatever $\mathcal{F}_{\mathrm{RO}}$ returns. Otherwise, first execute *Round Reset*. Then, set $t_{\mathcal{A}} \leftarrow t_{\mathcal{A}} + 1$ and only if $t_{\mathcal{A}} \leq t_\tau$ forward the request to $\mathcal{F}_{\mathrm{RO}}$ and return to $\mathcal{A}$ whatever $\mathcal{F}_{\mathrm{RO}}$ returns.
- Upon receiving $(\textsc{Retrieved})$ from $\mathcal{F}_{\mathrm{CRS}}^{\mathcal{D}}$, set $\tau = 1$.
- Any other request from any participant or the adversary is simply relayed to the underlying functionality without any further action and the output is given to the destination specified by the hybrid functionality.

Corruption Handling: Upon receiving $(\textsc{corrupt}, \mathrm{sid}, \mathsf{P})$ from the adversary, set $\mathcal{P}' \leftarrow \mathcal{P}' \cup \mathsf{P}$.

Procedure Round-Reset: Send $(\textsc{clock-read}, \mathrm{sid}_C)$ to $\mathcal{G}_{\textsc{Clock}}$ and receive $(\textsc{clock-read}, \mathrm{sid}_C, \tau')$ from $\mathcal{G}_{\textsc{Clock}}$. If $|\tau - \tau'| > 0$, then set $t_{\mathcal{A}} \leftarrow 0$ for the adversary and set $\tau \leftarrow \tau'$.

Diffusion functionality. We model the synchronous communication by $\mathcal{F}_{\textsc{Diffuse}}$ [BMTZ17]. Note that we present $\mathcal{F}_{\textsc{Diffuse}}^{\Delta}$ which is parameterized by the network delay Δ, and the synchronous variant can be easily derived by setting $\Delta = 1$.

Functionality $\mathcal{F}^{\mathit{\Delta}}_{\textsc{Diffuse}}$

The functionality is parameterized with a set possible senders and receivers $\mathcal{P}$. Any newly registered (resp. deregistered) party is added to (resp. deleted from) $\mathcal{P}$.

- **Honest sender diffusion.** Upon receiving $(\textsc{diffuse}, \mathrm{sid}, m)$ from some $\mathsf{P} \in \mathcal{P}$, where $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ denotes the current party set, do: (i) choose n new unique message-IDs $\mathrm{mid}_1, \ldots, \mathrm{mid}_n$; (ii) initialize $2n$ new variables $D_{\mathrm{mid}_1}, D^{MAX}_{\mathrm{mid}_1}, \ldots, D_{\mathrm{mid}_n}, D^{MAX}_{\mathrm{mid}_n}$ to 1; (iii) set $\vec{M} \leftarrow \vec{M} \parallel (m, \mathrm{mid}_1, D_{\mathrm{mid}_1}, \mathsf{P}_1) \parallel \ldots \parallel (m, \mathrm{mid}_n, D_{\mathrm{mid}_n}, \mathsf{P}_n)$; and (iv) send $(\textsc{diffuse}, \mathrm{sid}, m, \mathsf{P}, (\mathsf{P}_1, \mathrm{mid}_1), \ldots, (\mathsf{P}_n, \mathrm{mid}_n))$ to the adversary.
- **Adversarial sender diffusion.** Upon receiving $(\textsc{diffuse}, \mathrm{sid}, (m_{i_1}, \mathsf{P}_{i_1}), \ldots, (m_{i_\ell}, \mathsf{P}_{i_\ell}))$ from the adversary, where $\{\mathsf{P}_1, \ldots, \mathsf{P}_{i_\ell}\} \subseteq \mathcal{P}$, do: (i) choose ℓ new unique message-IDs $\mathrm{mid}_{1_\ell}, \ldots, \mathrm{mid}_{i_\ell}$; (ii) initialize 2ℓ new variables $D_{\mathrm{mid}_{i_1}}, D^{MAX}_{\mathrm{mid}_{i_1}}, \ldots, D_{\mathrm{mid}_{i_\ell}}, D^{MAX}_{\mathrm{mid}_{i_\ell}}$ to 1; (iii) set $\vec{M} \leftarrow \vec{M} \parallel (m_{i_1}, \mathrm{mid}_{i_1}, D_{\mathrm{mid}_{i_1}}, \mathsf{P}_{i_1}) \parallel \ldots \parallel (m_{i_\ell}, \mathrm{mid}_{i_\ell}, D_{\mathrm{mid}_{i_\ell}}, \mathsf{P}_{i_\ell})$; and (iv) send $(\textsc{diffuse}, \mathrm{sid}, m, \mathsf{P}, (m_{i_1}, \mathsf{P}_{i_1}, \mathrm{mid}_{i_1}), \ldots, (m_{i_\ell}, \mathsf{P}_{i_\ell}, \mathrm{mid}_{i_\ell}))$ to the adversary.
- **Honest party fetching.** Upon receiving $(\textsc{fetch}, \mathrm{sid})$ from $\mathsf{P} \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of P if P is corrupted): For all tuples $(m, \mathrm{mid}, D_{\mathrm{mid}}, \mathsf{P}) \in \vec{M}$, set $D_{\mathrm{mid}} \leftarrow D_{\mathrm{mid}} - 1$. Let $\vec{M}^{\mathsf{P}}_0$ denote the subvector $\vec{M}$ including all tuples of the form $(m, \mathrm{mid}, D_{\mathrm{mid}}, \mathsf{P})$ with $D_{\mathrm{mid}} = 0$ (in the same order as they appear in $\vec{M}$). Then, delete all entries in $\vec{M}^{\mathsf{P}}_0$ from $\vec{M}$ and send $\vec{M}^{\mathsf{P}}_0$ to P.
- **Adding adversarial delays**. Upon receiving $(\textsc{delays}, \mathrm{sid}, (T_{\mathrm{mid}_{i_1}}, \mathrm{mid}_{i_1}), \ldots, (T_{\mathrm{mid}_{i_\ell}}, \mathrm{mid}_{i_\ell}))$ from the adversary do the following for each pair $(T_{\mathrm{mid}_{i_j}}, \mathrm{mid}_{i_j})$: if $D^{MAX}_{\mathrm{mid}_{i_j}} + T_{\mathrm{mid}_{i_j}} \leq \mathit{\Delta}$ and mid_{i_j} is a message-ID registered in the current $\vec{M}$, set $D_{\mathrm{mid}_{i_j}} \leftarrow D_{\mathrm{mid}_{i_j}} + T_{\mathrm{mid}_{i_j}}$ and set $D^{MAX}_{\mathrm{mid}_{i_j}} \leftarrow D^{MAX}_{\mathrm{mid}_{i_j}} + T_{\mathrm{mid}_{i_j}}$; otherwise, ignore this pair.
- **Adversarially reordering messages**. Upon receiving $(\mathsf{swap}, \mathrm{sid}, \mathrm{mid}, \mathrm{mid}')$ from the adversary, if mid and mid′ are message-IDs registered in the current $\vec{M}$, then swap the triples $(m, \mathrm{mid}, D_{\mathrm{mid}}, \cdot)$ and $(m, \mathrm{mid}', D_{\mathrm{mid}'}, \cdot)$ in $\vec{M}$. Return $(\textsc{swap}, \mathrm{sid})$ to the adversary.

Common reference string functionality. We model a public setup by the CRS functionality $\mathcal{F}^{\mathcal{D}}_{\mathrm{CRS}}$. This functionality is parameterized with some distribution $\mathcal{D}$ with sufficiently high entropy. Upon receiving $(\textsc{Retrieve}, \mathrm{sid})$ from any party for the first time, $\mathcal{F}^{\mathcal{D}}_{\mathrm{CRS}}$ generates a string $d \leftarrow \mathcal{D}$ as the common reference string.

Functionality $\mathcal{F}^{\mathcal{D}}_{\mathrm{CRS}}$

When activated for the first time on input $(\textsc{Retrieve}, \mathrm{sid})$, choose a value $d \leftarrow \mathcal{D}$, and send $(\textsc{Retrieve}, d)$ back to the activated party; also send $(\textsc{Retrieved}, \mathrm{sid})$ to $\mathcal{W}(\mathcal{F}_{\mathrm{RO}})$. In each other activation return the value d to the activated party.

C Algorithms Omitted in the Main Body

Chain validation check. The following functions help us simplify the validation process. First, we adopt $\mathsf{ValidContent}$ to validate the block content. When the input is $\mathtt{IB}$ and $\mathsf{ValidContent}$ extracts its associated block content and returns true only when the block content is a valid type of input in our consensus protocol; when the input is $\mathcal{B}$, ValidContent returns true only when the associated block content contains only block headers of input-blocks and other types of valid transactions (we omit the details as it is irrelevant to our consensus protocol).

Next, we use $\mathsf{ValidBlock}^T$ to verify if a (chain-)block is a successful PoW on the i-th chain (that is, the nonce ctr is valid and the block hash — i-th segment of the RO output is less than target T).

$$\mathsf{ValidBlock}^T(\langle ctr, r, h, st, h', val\rangle, i) = [H(\langle ctr, r, h, st, h', val\rangle)]_{i\sim m} < T \wedge ctr < 2^{32}$$

Similarly, we use use $\mathsf{ValidInputBlock}^T$ to verify if an input-block is a successful PoW on the i-th chain by checking the reverse of the string segment.

Let isDenseChain denote the a predicate that returns true iff. a chain $\mathcal{C}$ is a dense chain. Slightly abusing the notations, when ExtractInputFreshness is called with a single chain $\mathcal{C}$ we let this denote the fresh randomness for input-blocks extracted from this chain (note that this can be computed without parallel chains).

```
Algorithm 8 IsValidChain($\mathbb{C}$, $denseChains$, $r$)
▷ This algorithm has five internal Boolean variables goodHash, goodNonce, goodTime, goodContent and goodRef, all initialized as true.
for $i$ from 1 to phase do
    for $j$ from 1 to $m$ do
        Call isValidChain($\mathbb{C}, denseChains, r, i, j, \mathbb{C}_j^{(i)}$)
        for $\mathcal{C} \in denseChains[i][j]$ do
            Call isValidChain($\mathbb{C}, denseChains, r, i, j, \mathcal{C}$)
            if not isDenseChain($\mathcal{C}$) then goodRef ← false
        end for
    end for
end for
return goodHash ∧ goodNonce ∧ goodTime ∧ goodContent ∧ goodRef

procedure isValidChain($\mathbb{C}, denseChains, r, phase, chain, \mathcal{C}$)
    $r^* \leftarrow \min\{r, phase \cdot \rho\}$
    for $k$ from len($\mathcal{C}$) to 1 do
        Parse $\mathcal{C}_k$ as $\mathcal{B} = \langle ctr, r, h, st, h', val\rangle$
        if $k > 1$ then
            Parse $\mathcal{C}_{k-1}$ as $\mathcal{B}^*$
            if $[h]_{chain\sim m} \neq [H(\mathcal{B}^*)]_{chain\sim m}$ then goodHash ← false
        else if phase = 1 then
            if $h \neq 0^\kappa$ then goodHash ← false
        end if
        if not $\mathsf{ValidBlock}^T(\mathcal{B}, j)$ then goodNonce ← false
        if not $(phase - 1)\rho < \mathsf{TS}(\mathcal{B}) < r^*$ then goodTime ← false
        if not ValidContent($\mathcal{B}$) then goodContent ← false
        Run IsValidInputBlocks($\mathcal{B}, \mathcal{C}, phase, chain$)
        $r^* \leftarrow \mathsf{TS}(\mathcal{B})$
        if $phase > 1$ then                                  ◁ Check cross-chain reference
            $denseCnt \leftarrow 0$
            for $q$ from $1$ to $m$ do
                Parse $\mathcal{C} = \mathbb{C}^{(phase-1)}_{chain}$
                if $[\text{head}(\mathcal{C}^{\lceil \rho_{\mathsf{ref}}})]_{q\sim m} = [h']_{q\sim m}$ and $\mathsf{isDenseChain}(\mathcal{C})$ then
                    $denseCnt \leftarrow denseCnt + 1$
                end if
                if $\exists \mathcal{C}' \in denseChains[phase][chain]$ and$[\text{head}(\mathcal{C}'^{\lceil \rho_{\mathsf{ref}}})]_{q\sim m} = [h']_{q\sim m}$ then
                    $denseCnt \leftarrow denseCnt + 1$
                end if
            end for
            if $denseCnt < \beta \cdot m$ then $\mathsf{goodRef} \leftarrow \mathsf{false}$
        end if
    end for
end procedure

procedure IsValidInputBlocks($\mathcal{B}, \mathcal{C}, phase, chain$)
    for each $\mathtt{IB} = \langle ctr, r, aux, h', val \rangle \in \mathcal{B}$ do
        if $\exists \mathtt{IB}' \in \mathcal{B}' \in \mathcal{C}$ and $[H(\mathtt{IB}')]^{\mathsf{R}}_{chain\sim m} = [H(\mathtt{IB})]^{\mathsf{R}}_{chain\sim m}$ then
            $\mathsf{goodHash} \leftarrow \mathsf{false}$
        end if
        if not $\mathsf{ValidInputBlock}^{T}(\mathtt{IB}, chain)$ then $\mathsf{goodNonce} \leftarrow \mathsf{false}$
        if not $(phase - 1)\rho < r < \mathsf{TS}(\mathcal{B})$ then $\mathsf{goodTime} \leftarrow \mathsf{false}$
        if not $\mathsf{ValidContent}(\mathtt{IB})$ then $\mathsf{goodContent} \leftarrow \mathsf{false}$
        $h^* = \mathsf{ExtractInputFreshness}(\mathcal{C})$
        if $[h']_{chain\sim m} \neq [h^*]_{chain\sim m}$ then $\mathsf{goodRef} \leftarrow \mathsf{false}$
    end for
end procedure
```

D Proofs Omitted in the Main Body

The following mathmatical facts are of interest.

Theorem 9 (Chernoff bounds). *Suppose $\{X_i : i \in [n]\}$ are mutually independent Boolean random variables, with $\mathbf{Pr}[X_i = 1] = p$, for all $i \in [n]$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = pn$. Then, for any $\delta \in (0, 1]$, it holds that*

$$\mathbf{Pr}[X \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2} \text{ and } \mathbf{Pr}[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/3}.$$

Also, for all $t > 0$,

$$\mathbf{Pr}[X \geq \mu + t] \leq e^{-2t^2 n}.$$

We summarize all protocol parameters and their explanation in Table 2 in Appendix E.1.

D.1 Typical Execution on Parallel Chains

In this section we present a formal analysis of the security of parallel chains by using the typical execution analytical framework from [GKL14, GKL17].

Notations and preliminaries. We write h as the number of honest RO queries and t as the number of corrupted RO queries. To illustrate the honest advantage against the adversary, we adopt an additional parameter $\delta \in (0, 1]$ such that $t \leq (1 - \delta)h$. In order for our protocol to work, the following conditions should be satisfied.

$$3f + 3\epsilon < \delta \leq 1. \tag{C1}$$

For the purpose of estimating the number of blocks acquired by honest parties during a sequence of rounds, we define the following random variables X, X', Y, Z, Z' with respect to round r. First, if in round r *at least* one honest party successfully solves a PoW with respect to chain-block, then $X_r = 1$ (we call r a successful round); otherwise $X_r = 0$. If in round r *at least* one honest party successfully solves a PoW with respect to input-block, then $X'_r = 1$; otherwise $X'_r = 0$. If in round r *exactly* one honest party solves a PoW w.r.t. chain-block, then $Y_r = 1$ (we call r a unique successful round); otherwise $Y_r = 0$. Regarding the adversary, let Z_r (Z'_r resp.) denote the total number of successful PoWs with respect o chain-blocks (input-blocks resp.) that the adversary gets at round r. Note that Z_r (Z'_r resp.) can be viewed as the sum of Bernoulli random variable Z_{rij} (Z'_{rij} reps.) denote the i-th query of the j-th corrupted party in round r. For a set of rounds S, let $X(S) = \sum_{r \in S} X_r$ and similarly define $X'(S), Y(S), Z(S)$ and $Z'(S)$.

The following mathematical facts with respect to $X_r, X'_r, Y_r, Z_r, Z'_r$, derived from the q-bounded model [GKL14] to one query per party per round, can be useful in later proofs.

$$\begin{aligned}(1 - f)ph < f =& \mathbb{E}[X_r] = \mathbb{E}[X'_r] = 1 - (1 - p)^h < ph,\\ & \mathbb{E}[Y_r] \geq ph(1 - p)^{h-1} > ph[1 - ph] \geq f(1 - f),\\ & \mathbb{E}[Z_r] = \mathbb{E}[Z'_r] = pt = \frac{t}{h}ph < \frac{t}{h}\frac{f}{1 - f} < (1 + \frac{\delta}{2}) \cdot f \cdot \frac{t}{h}.\end{aligned}$$

Typical executions. We formally define typical executions. An execution is typical if for any set S of at least k consecutive rounds, it holds that (i) for all random variables $X(S), X'(S), Y(S), Z(S)$ and $Z'(S)$, the deviation from their expected value is bounded by a concentration quality parameter ϵ; and (ii) no bad events with respect to random oracle (collisions) happen during these rounds.

Regarding bad events with respect to random oracle (cf. [GKL14]), an *insertion* occurs when, given a chain $\mathcal{C}$ with two consecutive blocks $\mathcal{B}$ and $\mathcal{B}'$, a block $\mathcal{B}^*$ created after $\mathcal{B}'$ is such that $\mathcal{B}, \mathcal{B}^*, \mathcal{B}'$ form three consecutive blocks of a valid chain; a *copy* occurs if the same block exists in two different positions; a *prediction* occurs when a block extends one which was computed at a later round.

Definition 4 (Typical execution). *An execution is (ϵ, k)-typical, for $\epsilon \in (0, 1)$ and constant integer k if, for any set S of at least k consecutive rounds, the following hold.*
(a) $(1 - \epsilon)\mathbb{E}[X(S)] < X(S) < (1 + \epsilon)\mathbb{E}[X(S)]$ *and* $(1 - \epsilon)\mathbb{E}[X'(S)] < X'(S) < (1 + \epsilon)\mathbb{E}[X'(S)]$
(b) $(1 - \epsilon)\mathbb{E}[Y(S)] < Y(S)$.
(c) $Z(S) < \mathbb{E}[Z(S)] + \epsilon\mathbb{E}[X(S)]$ *and* $Z'(S) < \mathbb{E}[Z'(S)] + \epsilon\mathbb{E}[X'(S)]$.
(d) No insertions, no copies, and no predictions occurred.

With typical executions and random variables defined above, we prove the following lemma.

Lemma 10. *For any $\beta < 1$, there exist protocol parameterizations such that an execution running for a constant number ρ of rounds is typical with probability at least β.*

Proof. Regarding Property (a), (b) and (c), consider any k consecutive rounds. Let badX denote the event that either $X(S) \geq (1 + \epsilon)\mathbb{E}[X(S)]$ or $X(S) \leq (1 - \epsilon)\mathbb{E}[X(S)]$; let badY denote the event that $Y(S) \leq (1-\epsilon)\mathbb{E}[Y(S)]$; and let badZ denote the event that $Z(S) \geq \mathbb{E}[Z(S)] + \epsilon\mathbb{E}[X(S)]$. Following the Chernoff Bounds (Theorem 9), we have

$$\begin{aligned}\mathbf{Pr}[\mathsf{badX}] = \mathbf{Pr}[\mathsf{badX}'] &\leq \exp(-\epsilon fk/3) + \exp(-\epsilon fk/2),\\ \mathbf{Pr}[\mathsf{badY}] &\leq \exp(-\epsilon fk/2),\\ \mathbf{Pr}[\mathsf{badZ}] = \mathbf{Pr}[\mathsf{badZ}'] &\leq \exp(-2\epsilon^2 f^2 k^3).\end{aligned}$$

The probability that neither one of them happens is lower-bounded by

$$\mathbf{Pr}\big[\neg\mathsf{badX} \wedge \neg\mathsf{badX}' \wedge \neg\mathsf{badY} \wedge \neg\mathsf{badZ} \wedge \neg\mathsf{badZ}'\big] \geq 1 - 7\exp\big(-\epsilon fk/3\big).$$

Notice that $k \mapsto 1 - 7\exp(-\epsilon fk/3)$ is increasing in $(0, +\infty)$, and $\lim_{k\to\infty} 1 - 7\exp(-\epsilon fk/3) = 1$, for any $\beta' < 1$ there exists k such that $\mathbf{Pr}[\neg\mathsf{badX} \wedge \neg\mathsf{badX}' \wedge \neg\mathsf{badY} \wedge \neg\mathsf{badZ} \wedge \neg\mathsf{badZ}'] \geq \beta'$.

We then consider the execution running for ρ rounds. Without loss of generality, assume $\rho = c \cdot k$ for a constant $c \in \mathbb{N}^+$. The number of set of at least k consecutive rounds in ρ is $(1/2)(ck-k+2)(ck-k+1) < c^2k^2$. Hence, the probability that an execution running for $\rho = c \cdot k$ steps and Property (a)(b)(c) holds on any set of k consecutive rounds is lower-bounded by

$$[1 - 7\exp(-\epsilon fk/3)]^{c^2k^2} \geq 1 - 7c^2k^2\exp(-\epsilon fk/3) = 1 - 7\exp(-\epsilon fk/3 + 2\ln c + 2\ln k).$$

Notice that there exists constant c such that $k \mapsto 1 - 7\exp(-\epsilon fk/3 + 2\ln c + 2\ln k)$ is increasing in $(c, +\infty)$ and $\lim_{k\to\infty} 1 - 7\exp(-\epsilon fk/3 + 2\ln c + 2\ln k) = 1$, for any $\beta < 1$ there exists k such that the probability that an execution running for $\rho = c \cdot k$ steps and Property (a)(b)(c) holds on any set of k consecutive rounds is at least β.

Regarding Property (d), the RO queries made by all parties in ρ rounds is $Q = (h+t) \cdot \rho$. Recall our $m\times 1$ PoW construction in Section 3.1, we consider RO output of length $\mathsf{polylog}\kappa$. A collision happens with probability $Q^2/2^{\mathsf{polylog}\kappa} = \exp(-\Omega(\mathsf{polylog}\kappa + \log\rho)$ which is negligible. (Note that Property (d) actually holds on an execution of $L = \mathsf{poly}(\kappa)$ steps.) □

Recall that we run $m = \Theta(\log^2\kappa)$ chains in parallel using $m\times 1$ PoW, and all parallel executions are mutually independent of each other, we show that the above probability on a single execution translates to the fraction of typical executions among m parallel ones.

Theorem 1. *For any $\beta < 1$, running $m = \Theta(\log^2\kappa)$ parallel chains as described above for a constant number ρ of rounds, results in at least a β fraction of them being typical with overwhelming probability in κ.*

Proof. Fix $\beta < 1$. We define random variable $Q_i (i \in [m])$ as follows. If the i-th execution among m parallel repetitions, running for a constant number of ρ steps is typical, $Q_i = 1$; otherwise $Q_i = 0$. Also let $Q = \sum Q_i$. From Lemma 10 we learn that there exists protocol parameterizations such that $\forall i \in [m], \mathbf{Pr}[Q_i = 1] = (1+2\epsilon)\beta$. Hence $\mathbb{E}[Q] = (1+2\epsilon)\beta m$. Since our $m\times 1$ PoW construction achieves full independence over all parallel chains, we have

$$\mathbf{Pr}[Q \leq \beta m] < \mathbf{Pr}[Q \leq (1-\epsilon)(1+2\epsilon)\beta m] = \mathbf{Pr}[Q \leq (1-\epsilon)\mathbb{E}[Q]] \leq \exp(-\epsilon^2(1+2\epsilon)\beta m).$$

The first inequality holds because $\epsilon < 1/2$, and the last one is due to the Chernoff bounds. Since $m = \Theta(\log^2\kappa)$, this probability is negligible in terms of κ. I.e., there exist protocol parameterizations such that, with overwhelming probability, at least β fraction of the executions are typical. □

D.2 Analysis of Phase Oblivious Agreement

In a typical execution, the random variables satisfy the following properties.

Lemma 11 (cf. [GKL14]). *The following hold for any set S of at least k consecutive rounds in a typical execution. Note that the relations on $X(S)$ ($Z(S)$ resp.) also applies on $X'(S)$ ($Z'(S)$ resp.).*
(a) $(1-\epsilon)f|S| < X(S) < (1+\epsilon)f|S|$ *and* $(1-\delta/3)f|S| < Y(S)$.
(b) $Z(S) < (1-2\delta/3)f|S|$.
(c) $Z(S) < (1-\delta/2)X(S)$ *and* $Z(S) < Y(S)$.

To achieve phase oblivious agreement, certain conditions on the length of phases and stages should be satisfied which we summarize as follows.

$$\rho_{\mathsf{ref}} \geq \frac{8k}{\delta}; \rho_{\mathsf{view}} \geq (\frac{3}{\delta}-2)\rho_{\mathsf{ref}} + (\frac{3}{\delta}-1)k \text{ and } \rho_{\mathsf{output}} \geq \frac{12}{\delta}(\rho_{\mathsf{view}}+k) \tag{C2}$$

Also, we require that the density parameter τ is set such that it is roughly the same as block generation rate f.

$$\tau = (1-\epsilon)(1-\frac{2k}{\rho_{\mathsf{ref}}})f. \tag{C3}$$

We now consider two lemmas — Lemma 12 and Lemma 13 in the first phase where parties start simultaneously with CRS. Notice that neither lemma can be applied unconditionally in the second and later phases, as in these phases the adversary can pre-mine for a bounded amount of time. We will treat them carefully in the proof of Theorem 2.

First, Lemma 12 considers two basic blockchain properties — common prefix and existential chain quality. A detailed proof can be found in [GKL14].

Lemma 12. *The following two properties hold in a typical execution.*

- Common prefix. *Let $\mathcal{C}_1, \mathcal{C}_2$ denote two chains held by two honest parties $\mathsf{P}_1, \mathsf{P}_2$ respectively at round r. It holds that $\mathcal{C}_1^{\lceil k} = \mathcal{C}_2^{\lceil k}$*
- Existential chain quality. *For any set S of k consecutive rounds, there is at least one block $\mathcal{B} \in \mathcal{C}$ such that the timestamp of $\mathcal{B}$ is in S and $\mathcal{B}$ is produced by an honest party.*

Proof(sketch). To show common prefix, we consider a contradiction. Suppose $\mathcal{C}_1^{\lceil k} \neq \mathcal{C}_2^{\lceil k}$, and let $\mathcal{B}$ be the last honest block on the common part of $\mathcal{C}_1$ and $\mathcal{C}_2$ and r^* its timestamp. Let $S = \{r^*, \ldots, r\}$. Since $\mathcal{C}_1^{\lceil k} \neq \mathcal{C}_2^{\lceil k}$ we have $Z(S) > Y(S)$ (by pairing every unique successful round with an adversarial block) which contradicts Lemma 11(c).

Regarding existential chain quality, since the adversary cannot create a fork more than k rounds it implies he cannot revert all honest blocks in a consecutive k rounds. □

Next, since chain selection rule asks for cross-chain reference that links to sufficiently many dense chains, we show that when the execution is typical in the first phase, honest parties can prepare a dense chain by themselves.

Lemma 13. *In a typical execution, all honest parties hold a dense chain at the end of the first phase.*

Proof. Let $\mathcal{C}$ denote a chain held by an honest party P at the end of i-th phase. Consider a set $S = \{u, \ldots, v\}$ consecutive rounds where $|S| > \rho_{\mathsf{ref}}$. By Lemma 12, there exists at least one honest block $\mathcal{B}$ with timestamp $r \in [u, u+k)$ and at least one honest block $\mathcal{B}'$ with timestamp $r' \in (v-k, v]$. Let $S' = \{r, r+1, \dots, r'\}$. We learn that the number of blocks in S is lower-bounded as follows.

$$X(S') \geq (1-\epsilon)f|S'| \geq (1-\epsilon)(1-\frac{2k}{\rho_{\mathsf{ref}}})f|S| \geq \tau|S|.$$

The first inequality follows Lemma 11(a); the second is because $|S'| \geq |S| - 2k = (1-(2k/|S|))|S| \geq (1-(2k/\rho_{\mathsf{ref}}))|S|$; and the last one follows Condition (C3). □

The following lemma proves an upper-bound on pre-mining in phase $i+1$, given that phase i achieves phase oblivious agreement.

Lemma 14. *If parties achieves phase oblivious agreement in phase i, the adversary cannot mine valid blocks in phase $i+1$ before round $i \cdot \rho - \rho_{\mathsf{ref}} - k$.*

Proof. Without loss of generality, for the sake of a contradiction, suppose the i-th phase is the earliest phase such that parties mine valid blocks in the $(i+1)$-th phase before round $i \cdot \rho - \rho_{\mathsf{ref}} - k$. I.e., they prepare at least $(3m/4+1)$ dense chains before this round. We consider three types of adversarial strategies.

First, the adversary stick to the typical chains in phase i and start to mine blocks in phase $i+1$ after he learns sufficiently many block hashes on dense chains. Consider an arbitrary chain in phase i such that typical execution holds. Lemma 12 implies that there is an honest block with timestamp r in $(i \cdot \rho - \rho_{\mathsf{ref}} - k, i \cdot \rho - \rho_{\mathsf{ref}}]$ which is produce at round r. If the adversary starts earlier than $i \cdot \rho - \rho_{\mathsf{ref}} - k$, it implies a prediction which contradicts the execution being typical.

Second, the adversary stick to the typical chains for a while and at some point $r > (i-1)\rho + \rho_{\mathsf{view}}$ he creates a fork from the honest chain. In order for this fork to be dense, the adversary should prepare at least $\tau \cdot (i \cdot \rho - r)$ blocks on his own in rounds $S = \{r, r+1, \dots, i \cdot \rho - \rho_{\mathsf{ref}} - k\}$. However, we get the following contradiction.

$$Z(S) < (1-\frac{2\delta}{3})f|S| < (1-\frac{2k}{\rho_{\mathsf{ref}}})(1-\epsilon)f|S| < \tau \cdot |S + \rho_{\mathsf{ref}} + k|.$$

The first inequality comes from Lemma 11(b); the second one follows Condition (C1) and the inequality between k and ρ_{ref} in Condition (C2); and the last one is achieved by substituting the density parameter τ.

Third, the adversary builds a chain on his own as soon as he learns the fresh randomness in phase $i-1$ and for all blocks in this private chain he insert fake timestamps in the last two stages. I.e., he needs to prepare $\tau \cdot (\rho_{\mathsf{output}} + \rho_{\mathsf{ref}})$ blocks on his own in $\rho + k$ rounds. We get the following contradiction.

$$Z(\rho+k) < (1-\frac{2\delta}{3})f|\rho+k| < (1-\frac{2k}{\rho_{\mathsf{ref}}})(1-\epsilon)(1-\frac{\delta}{12})f|\rho+k| < \tau \cdot |\rho_{\mathsf{output}} + \rho_{\mathsf{ref}}|.$$

The first inequality comes from Lemma 11(b); the second one follows Condition (C1) and the inequality between k and ρ_{ref} in Condition (C2); and the last one is achieved by substituting the density parameter τ and the inequality between ρ_{view} and ρ_{output} in Condition (C2). □

We now restate and formally prove Theorem 2 in Section 3.2.

Theorem 2 (Phase oblivious agreement). *There exist protocol parameterizations such that the following properties hold. Let $\beta \in (3/4, 1)$ and consider a phase i. Let $\mathbb{C}, \mathbb{C}'$ denote the parallel chains held by two honest parties P, P' at rounds r, r' after phase i (i.e., $\min\{r, r'\} > i\rho$), respectively. Then there exists a subset $S \subseteq \{1, 2, \dots, m\}$ of size larger than $\beta \cdot m$ such that for all $j \in S$, the following two properties hold on chains $\mathcal{C} = \mathbb{C}_j^{(i)}$ and $\mathcal{C}' = \mathbb{C}_j'^{(i)}$.*

- **Agreement:** $\mathcal{C}^{\lceil \rho_{\mathsf{ref}}} = \mathcal{C}'^{\lceil \rho_{\mathsf{ref}}}$.
- **Honest input-block majority:** *For all input-blocks included in the output generation stage of $\mathcal{C}$ and $\mathcal{C}'$, more than half of them are produced by honest parties.*

Proof. We prove by induction.

Consider the first phase. Theorem 1 shows that typical execution holds on at least β fraction of the chains. Consider the j-th execution on j-th chain where typical execution holds. At the end of round ρ, let $\mathcal{C}$ and $\mathcal{C}'$ be chains held by two honest parties respectively. By common prefix (Lemma 12) it holds that $\mathcal{C}^{\lceil k} = \mathcal{C}'^{\lceil k}$ hence $\mathcal{C}^{\lceil \rho_{\mathsf{ref}}} = \mathcal{C}'^{\lceil \rho_{\mathsf{ref}}}$.

Moreover, existential chain quality (Lemma 12) implies that there is at least one honest block with timestamp in $(\rho_{\mathsf{view}} - k, \rho_{\mathsf{view}}]$, and there is at least one honest block with timestamp in $(\rho_{\mathsf{view}} - 2k, \rho_{\mathsf{view}} - k]$. Thus, for the last block in the view convergence phase it is produced in at round $r^* > \rho_{\mathsf{view}} - 2k$ where r^* is the earliest time that the adversary can produce valid input-blocks on chain j. Similarly there is at least one honest block with timestamp in $(\rho_{\mathsf{view}} + \rho_{\mathsf{output}} - k, \rho_{\mathsf{view}} + \rho_{\mathsf{output}}]$. We learn that the honest parties can collect input-blocks mined in a set H of consecutive rounds of length at least $\rho_{\mathsf{output}} - k$; and the adversary can collect input-blocks mined in a set of consecutive rounds of length no more than $|H| + 3k$ (where $2k$ rounds comes from pre-mining and k rounds comes from post-mining). We learn that the majority of input-blocks included in the output generation stage of the first phase are produced by honest parties from the following inequality.

$$Z'(|H| + 3k) < (1 - \frac{2\delta}{3})f(|H| + 3k) \leq (1 - \frac{2\delta}{3})(1 + \frac{2\delta/3 - \epsilon}{1 - 2\delta/3})f|H| < (1 - \epsilon)f|H| < X'(H)$$

The first inequality is from Lemma 11(b); the next one is by the inequality between ρ_{view} and k in Condition C2; and the last inequality is by Lemma 11(a).

We also note that all honest parties hold at least $(3m/4 + 1)$ dense chains by Lemma 13.

Now, suppose the i-th phase satisfies phase oblivious agreement, we show that this also holds for the $(i + 1)$-th phase.

First, assume the execution on j-th chain is typical and let $\mathcal{C}$ denote a chain held by an honest party after the view convergence phase in phase $i + 1$. We show that the adversarial advantage from pre-mining can only revert all honest blocks up to time $i\rho + \rho_{\mathsf{view}} - k$. Let $S = \{i\rho - \rho_{\mathsf{ref}} - k, i\rho + \rho_{\mathsf{view}} - k\}$ and $S' = \{i\rho, i\rho + \rho_{\mathsf{view}} - k\}$. If the adversary reverts all honest blocks before round $i\rho + \rho_{\mathsf{view}} - k$, we have $Z(S) \geq Y(S')$. We get the following contradiction.

$$Z(S) < (1 - \frac{2\delta}{3})f|S| \leq (1 - \frac{2\delta}{3})(1 + \frac{1}{3/\delta - 2})f|S'| < (1 - \frac{\delta}{3})f|S'| < Y(S').$$

The first inequality is from Lemma 11(b); the next one is by the inequality between ρ_{view} and ρ_{ref} in Condition C2; and the last inequality is again by Lemma 11(a).

Since the adversarial pre-mining can revert all honest blocks up to time $i\rho + \rho_{\mathsf{view}} - k$, we learn that there must exist one honest block $\mathcal{B}$ in $\mathcal{C}$ with timestamp $r \in (i\rho + \rho_{\mathsf{view}} - 2k, i\rho + \rho_{\mathsf{view}} - k]$ and at least one honest block $\mathcal{B}'$ with timestamp $r' \in (i\rho + \rho_{\mathsf{view}} - k, i\rho + \rho_{\mathsf{view}}]$. Note that if the adversary choose not to revert an honest block at an earlier time before $i\rho + \rho_{\mathsf{view}} - k$, since the unpredictability of honest blocks he will lose all advantage. And the existence of $\mathcal{B}$ and $\mathcal{B}'$ can be argued by existential chain quality. Following a similar argument we conclude agreement and honest input-block majority in phase $i + 1$. □

Variable	Description
κ	Security parameter; length of the random oracle output.
m	The number of parallel chains ($m = \Theta(\log^2 \kappa)$).
ρ	The length of a phase in number of rounds, it consists of 3 different stages: (i) view convergence of length ρ_{view} rounds; (ii) output generation of length ρ_{output} rounds; and (iii) reference convergence of length ρ_{ref} rounds.
τ	The density parameter that asks for at least $\tau \cdot \rho_{\mathsf{ref}}$ blocks in any consecutive rounds of length at least ρ_{ref} in the output generation and reference convergence stage.
T	The target to successful solve a PoW.
h_r	Number of RO queries made by honest parties at round r.
t_r	Number of RO queries made by corrupted parties at round r.
δ	Advantage of honest parties ($t \leq (1-\delta)h$).
f	Block generation rate; the probability at least one honest party succeeds in finding a PoW in a round.
ϵ	Quality of concentration of random variables in typical executions, cf. Definition 4.

Table 2: Main parameters of ChainKingConsensus.

E Glossary

E.1 Main Parameters of ChainKingConsensus

E.2 Main State Variables of ChainKingConsensus Participants

Variable	Description
`r`	Party P's local time.
`phase`	Party P's local phase.
$\mathbb{C}_{\mathsf{local}}$	Party P's local parallel chain.
`denseChains`	Party P's local recorded dense chains.
`chainBuffer`	The buffer that stores all incoming new parallel-chains and their associated dense chains.
`IBBuffer`	The buffer that stores all incoming new input-blocks.
`val`	Party P's suggestion for protocol output.
`lock`	Whether P should adopt the output from chain-king.
`decide`	Whether P has decided to output `val`.
`exit`	Whether P should stop mining. `exit` $= \infty$ if P have not yet decided; `exit` $= 1$ implies that P will terminate mining in the next iteration and P stops mining when `exit` $= 0$.

Table 3: Main state variables in ChainKingConsensus.