Securing Proof-of-Work Ledgers via Checkpointing

Securing Proof-of-Work Ledgers via Checkpointing

Dimitris Karakostas
University of Edinburgh and IOHK
dimitris.karakostas@ed.ac.uk

Aggelos Kiayias
University of Edinburgh and IOHK
akiayias@inf.ed.ac.uk

***Abstract*—Distributed ledgers based on Proof-of-Work (PoW) are typically vulnerable when mining participation is low. During these periods, an attacker with a mining majority can mount serious attacks, like double spending or transaction censorship. Our work explores mechanisms to secure a ledger against such adversaries and puts forth the first rigorous study of checkpointing as a protection from 51% attacks. The core idea is to employ an external set of parties to finalize blocks after their creation. This idea takes the form of checkpointing and timestamping, the former ensuring low latency in a federated setting and the latter being fully decentralized. Crucially, we identify and protect against a (previously undocumented) attack, "block lead", thus our scheme is the first to guarantee liveness.**

***Index Terms*—checkpoints, timestamping, Proof-of-Work**

I. Introduction

During the early '80s, the seminal work of Shostak, Pease, and Lamport introduced the consensus problem [20], [27]. 30 Years later, Bitcoin [23] accelerated research by introducing "Nakamoto consensus" and the blockchain structure. It is well-known that, to achieve consensus in any setting with an active adversary, (at least) a majority of participants need to be honest [10]. Proof-of-Work (PoW) systems, like Bitcoin, assume that over 50% of hashing power backs the correct protocol and, if honest majority is violated, dangers arise. Typically 51% attacks try to revert *transaction finality*. Finality ensures that published transactions are stable after some time, i.e. cannot be reversed (unless with negligible probability). Finality attacks are devastating, since they invalidate the ledger's immutability and problems like "double spending" arise; simply put, if the adversary can revert any transaction it wishes, then it can double spend the same assets by first issuing a payment and then reverting it, after presumed final by its counterparty. Additionally, blockchain systems may face transaction censorship attacks [29], which may also cause significant financial damage. Such attacks are hard to identify reliably, so it is important to ensure censorship resistance *by construction*; with foresight, this can be satisfied by guaranteeing liveness.

Our Contributions and Roadmap.[1] Our work provides, to the best of our knowledge, the first rigorous analysis of mechanisms to mitigate majority attacks in PoW ledgers. Our contributions are as follows: i) a provably secure, federated and efficient checkpointing mechanism; ii) block lead, a novel attack against liveness which affects all existing checkpointing implementations; iii) a provably secure and fully decentralized timestamping-based checkpointing mechanism. In detail, Section III defines $\mathcal{F}_{\mathsf{Checkpoint}}$, the checkpointing "ideal functionality". $\mathcal{F}_{\mathsf{Checkpoint}}$ defines a checkpointed chain resolution mechanism, which guarantees persistence and liveness with non-negligible probability. The analysis includes "block lead", a novel attack against liveness which, to the best of our knowledge, has not been previously discussed. We model the execution as an absorbing Markov chain and show that liveness is guaranteed after a sufficient period of time. Finally, we realize $\mathcal{F}_{\mathsf{Checkpoint}}$ via a protocol (Section III-B) with (optimal, on the security parameter) size $O(\kappa)$ of updatable state. Our second design is *timestamping-based checkpointing* (Section IV). The key idea is to use a second ledger, presumably more reliable than the one we seek to protect, to perform timestamping, which we show to be equivalent to checkpoints, thus achieving the highest level of decentralization.

Related Work. In their seminal paper on PBFT [4], Castro and Liskov use checkpoints in a replicated setting to bring a replica which is "left behind" up to date. In blockchains, checkpoints are often used to prevent network attacks and enhance performance, e.g. Bitcoin introduced centrally-issued checkpoints to speed up bootstrapping of network nodes and mitigate DoS and 51% attacks. Our checkpointing solution is similar, albeit federated instead of fully centralized and without being vulnerable to block lead. Literature has primarily considered checkpoints w.r.t. Proof-of-Stake (PoS) protocols to prevent threats like nothing-at-stake [8], [21], long range [1], and stake bleeding [12] attacks, in protocols like Ouroboros [18], Snow White [5], and Ouroboros Praos [6], in the latter also used to mitigate adaptive corruptions. Employing a committee to finalize the ledger's state has also seen extensive research. On the PoW side, notable works include hybrid consensus [25], Thunderella [26], and ByzCoin [19]; on the PoS side, Algorand [13] uses a Verifiable Random Function to elect a committee which runs a Byzantine Agreement protocol, while Casper [2], [3] defines checkpointing which, in conjunction with PoW, protects against block reversions by (financially) penalizing adversaries. Afgjort [22] describes a generic finality layer, run by a sub-committee and applied on top of a blockchain. Nonetheless, in contrast to our scheme, all these

[1]Due to space constraints, we refer to the paper's extended version (https://eprint.iacr.org/2020/173.pdf) for: i) the full proofs of all theorems; ii) a liveness evaluation of Ethereum Classic; iii) a prototype implementation of the checkpointing protocol of Section III, which uses Raft [24] and issues checkpoints in less than a second; iv) two checkpointing variations, with randomized checkpointing interval unknown to $\mathcal{A}$ and when $\mathcal{A}$ is not rushing, both slightly improving liveness; v) an implementation description and evaluation of timestamping on Bitcoin and Ethereum.

978-0-7381-1420-0/21/$31.00 ©2021 IEEE

systems rely on honest majority. Finally, secure timestamping has been extensively researched. Specifically, timestamping has been implemented both centrally [15], using hashes and signatures, and in decentralized manner on Bitcoin [14], [16].

II. Preliminaries

We assume a synchronous setting, where the execution proceeds in rounds. The number of parties n is fixed for the duration of the execution (cf. the Bitcoin Backbone model [11]). We also assume a *diffuse* functionality, i.e. a gossip protocol which allows the parties to share messages without in a partially connected graph. Each party performs q queries to a *random oracle* (a hash function, in practice). The difficulty parameter p denotes the probability that a single query to the is successful, so the probability that a party produces a block at any given round is $q \cdot p$. $\mathcal{A}$ controls $\mu_{\mathcal{A}}$ of the network's mining power. We stress that it is possible that $\mu_{\mathcal{A}} > 0.5$, i.e. $\mathcal{A}$ might control the majority of the mining power. Additionally, $\mathcal{A}$ is "adaptive", i.e. corrupts parties on the fly, and "rushing", i.e. at each round retrieves all messages before deciding its strategy.

Our analysis relies on the ledger properties distilled in the Bitcoin Backbone model [11], *persistence* and *liveness*.

Definition 1 (Stable Transaction)**.** *A transaction is stable if every honest party reports it in the same ledger position.*

Definition 2 (Persistence)**.** *A transaction which is part of a block at least k blocks away from the ledger's head, i.e. a block which is part of the chain which results from removing the last k blocks of the current chain, is stable.*

Definition 3 (Liveness)**.** *Every transaction provided continuously as input to the parties is stable after u rounds.*

III. The Checkpointed Ledger

Our goal is to define a ledger which is resistant to attacks from an adversarial mining majority. In this section, we achieve this via the checkpointing functionality $\mathcal{F}_{\mathsf{Checkpoint}}$, which establishes *checkpoints*, i.e. irreversible chains. Upon retrieving a candidate chain, $\mathcal{F}_{\mathsf{Checkpoint}}$ decides whether to adopt it based on $\mathsf{maxvalid}(\cdot,\cdot)$, i.e. the chain decision rule [11]. When $\mathcal{F}_{\mathsf{Checkpoint}}$ adopts a chain which is k_c blocks longer than the latest checkpoint, it issues a new checkpoint and any chain which does not extend the latest checkpoint is automatically rejected. A larger k_c results in sparse checkpoints, while smaller k_c allows faster synchronization and restricts $\mathcal{A}$'s control over the chain's blocks (cf. Section III-A). Checkpoints also organize the execution in *epochs*, each beginning with a new checkpoint. Finally, to counter the "block lead" attack described next, every checkpoint contains an unpredictable nonce r; to model the ability of $\mathcal{A}$ to corrupt parties, $\mathcal{A}$ chooses a single nonce among a polynomial number of random nonces. Figure 1 defines the checkpointing functionality $\mathcal{F}_{\mathsf{Checkpoint}}$. $|C|$ denotes the length of a chain (in blocks), $|\mathbb{V}|$ the size of set $\mathbb{V}$, $||$ the concatenation of blocks, chains of blocks, and strings, $\prec$ the prefix operation, e.g. if $C = C'||\cdots$ then $C' \prec C$, $\setminus$ the difference of two chains, e.g. if $C = C'||B||\cdots$ then $C \setminus C' = B||\cdots$, and $tail(C)$ the last block of C.

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

$\mathcal{F}_{\mathsf{Checkpoint}}$ interacts with a set of parties $\mathbb{V}$ and holds the local chain C and the checkpoint chain C_c, both initially set to ϵ. It is parameterized by k_c, which defines the number of blocks between two consecutive checkpoints, and the $\mathsf{maxvalid}(\cdot,\cdot)$ algorithm.

On receiving $(\textsc{CandidateCheckpoint}, C')$ from a party $\mathcal{V}$, if $C_c \prec C'$ set $C := \mathsf{maxvalid}(C, C')$. Next, if $|C \setminus C_c| = k_c$ compute a list R of $p(\kappa)$ random values as $r_j \xleftarrow{\$} \{0,1\}^{\omega}$ and send $(\textsc{Nonce}, R)$ to $\mathcal{A}$. On receiving from $\mathcal{A}$ a response $(\textsc{Nonce}, r_i)$, such that $r_i \in R$, return $(\textsc{Checkpoint}, tail(C)||r_i)$ to $\mathcal{V}$ and set $C := C_c := C||r_i$.

Fig. 1. The checkpointing ideal functionality.

Block lead. As covered in the introduction, violating liveness enables devastating attacks. Before proceeding with the security analysis, we describe block lead, an attack which breaks liveness *even in the presence* of standard checkpointing mechanisms. To the best of our knowledge, this attack has not been previously discussed or taken into consideration, thus all existing checkpointing mechanisms fail to provide *any* liveness guarantees and protect the systems against adversarial mining majorities. The core observation is that an adversary $\mathcal{A}$ with a mining majority produces blocks faster than honest parties. Thus, $\mathcal{A}$ gains an advantage by withholding newly-mined blocks until a competing chain is produced (similar to Selfish Mining [9], [28]). Eventually, $\mathcal{A}$'s chain is much longer and can compete with and discard every future honest block. Liveness can thus be guaranteed only if this advantage is constrained. We achieve this via the unpredictable nonce r, which "refreshes" the randomness and prevents $\mathcal{A}$ from retaining its advantage across epochs.

A. Security of the Checkpointed Ledger

We now show that the checkpointed ledger satisfies *persistence* and *liveness* w.r.t. the following parameters: 1) k: the persistence parameter, i.e. the number of blocks after which a transaction is stable; 2) u: the liveness parameter, i.e. the amount of time that a transaction needs to be continuously provided to all parties before it becomes stable; 3) k_c: the checkpointing interval, i.e. the epoch's length; 4) q: the number of queries to the hashing oracle that a party can make during a single round; 5) p: the block difficulty, i.e. the probability that a single query is successful in producing a block; 6) n: the number of parties; 7) t: the number of adversarial parties.

Persistence. Intuitively, every ancestor block to a checkpoint is stable, so persistence is satisfied for every block up to the latest checkpoint. Theorem 1 formally proves this intuition, where $C^{\lceil k}$ denotes the chain which is output by removing the k last blocks from C.

Theorem 1 (Persistence)**.** *The checkpointed chain resolution protocol of Section III satisfies persistence (cf. Definition 2) for parameter $k \geq k_c$.*

Liveness. Proving that checkpoints guarantee liveness is significantly more challenging, so our analysis proceeds in distinct steps. First, Theorem 2 shows that a transaction's liveness is guaranteed as long as an honest block is checkpointed.

Theorem 2. *For any execution of a checkpointed chain resolution protocol which securely realizes $\mathcal{F}_{\mathsf{Checkpoint}}$ (cf. Section III), a transaction τ is stable (cf. Definition 1) if at least one honestly-generated block, which is mined after the creation of τ, is part of the checkpointed chain after u rounds since τ is diffused on the network.*

An epoch begins with the creation of a checkpoint and the accompanying unpredictable nonce r, which ensures that the execution is memoryless across epochs. Our analysis uses a (somewhat) simple absorbing Markov chain, parameterized by k_c, to express the checkpointed ledger's execution. Theorem 3 then shows that reaching the absorbing state translates into checkpointing an honest block, i.e. achieving liveness. Each state of the Markov chain is identified by (i, j). i denotes the number of blocks that the *honest parties* need to produce to reach the next checkpoint; similarly, j is the number of blocks that $\mathcal{A}$ needs to produce. Corollary 1 shows that, to violate liveness, $\mathcal{A}$ cannot adopt honest blocks, thus $\mathcal{A}$ mines separately from honest parties.

Corollary 1. *For every transaction τ and for every execution of a checkpointed chain resolution protocol which securely realizes $\mathcal{F}_{\mathsf{Checkpoint}}$ (cf. Section III), if $\mathcal{A}$ adopts an honest block which is produced after the creation of τ, then liveness is guaranteed for τ.*

Each epoch starts on state (k_c, k_c), where all parties need exactly k_c blocks to "reach" the next checkpoint. The absorbing state compounds all states of the form $(0, j)$ with $j > 0$. States (i, j) with $i > 0$ are transitional. Transitions represent the accumulation of honest and adversarial blocks in their respective chains. If an honest party produces multiple blocks in a single round, it diffuses only the first, so the only allowed transitions from state (i, j) are towards states $(i-a, j-b)$ with $a \in \{0, 1\}, b \in [0, j]$. We define the random variables:

- H: $H = 1$ if *at least one* honest party produces a block at a given round, else $H = 0$;
- $M^{(i)}$: $M^{(i)} = 1$ if *all adversarial parties* produce exactly i blocks at a given round, else $M^{(i)} = 0$;

for which the following hold:

- $\mathbb{E}(H) = h = 1 - (1 - p)^{q \cdot (n-t)}$;
- $\mathbb{E}(M^{(i)}) = m^{(i)} = \binom{q \cdot t}{i} \cdot p^i \cdot (1 - p)^{q \cdot t - i}$ for any i.

Lemma 1 shows that, when at least one honest block is created, *each* honest party's chain increases by one block; this result is adjacent to the *chain growth* property first implied in [11] and explicitly highlighted in [17].

Lemma 1. *If an honest party $\mathcal{V}$ has a chain of length l at round r, at round $r + 1$* every *honest party has a chain of length* at least *l.*

Lemma 2 defines the transition probability from a state (i, j) to $(i - a, j - b), a \in \{0, 1\}, b \in [0, j]$; we use the following notation: $\hat{m}_l = \sum_{\phi=0}^{l} m^{(\phi)}$, $\bar{h} = 1 - h$. These probabilities are minimal w.r.t. the honest chain growth, given that $\mathcal{A}$ publishes an l-long chain only if the longest honest chain is also l-long. Indeed, if $\mathcal{A}$ publishes its chain earlier, then the transition probabilities change in favor of the honest parties, since the honest parties converge quicker to the absorption state. Finally, Theorem 3 formalizes the liveness guarantees.

Lemma 2 (Transition Probabilities)**.** *For transitions from round (i, j), where $i > 1, j > 0$ and $b \in [0, j - 1]$, the following hold:*

- *transition to $(i, j - b)$ occurs with probability $\bar{h} \cdot m^{(b)}$;*
- *transition to $(i-1, j-b)$ occurs with probability $h \cdot m^{(b)}$.*

Additionally, the following special cases hold:

i) the state $(0, 0)$ is equivalent to the state (k_c, k_c);

ii) transition from round $(1, j)$, where $j > 0$, to the absorbing state occurs with probability $h \cdot \hat{m}_{j-1}$;

iii) from round (i, j), where $i, j > 0$, the following hold:
- *transition to $(i, 0)$ occurs with probability $\bar{h} \cdot (1 - \hat{m}_{j-1})$;*
- *transition to $(i - 1, 0)$ occurs with probability $\bar{h} \cdot (1 - \hat{m}_{j-1})$;*

iv) from round $(i, 0)$, where $i > 0$, the following hold:
- *transition to $(i - 1, 0)$ occurs with probability h;*
- *transition to $(i, 0)$ occurs with probability $\bar{h}$.*

Theorem 3 (Liveness)**.** *The Markov chain of Lemma 2 has the property that, whenever it reaches the absorbing state, an honest block is guaranteed to be checkpointed in the corresponding execution with error probability $L \cdot 2^{-\omega}$, L being the protocol execution length.*

Finally, we observe that, from every state, there exists a path which reaches the absorption state with non-zero aggregate probability. Therefore, for a sufficiently large number of steps absorption probability is non-negligible and, as the number of steps tends to infinity, the liveness probability tends to 1.

B. The Checkpointed Chain Resolution Protocol

Now we realize as a federated service distributed among parties. We now construct a federated protocol which securely implements $\mathcal{F}_{\mathsf{Checkpoint}}$ (cf. Theorem 4). The checkpointing protocol (Figure 2) is parameterized by a validation predicate $\mathsf{Validate}$, which identifies whether a chain is valid, e.g. verifies the signatures and the Proof-of-Work of the chain's blocks. Also it employs an interactive consistency protocol π_{IC} (cf. [7]), s.t. parties both reach agreement on the block to checkpoint and collectively produce the nonce r. After π_{IC} ends, each party outputs a list $[\langle B_1, r_1 \rangle, \ldots, \langle B_n, r_n \rangle]$ containing everybody's inputs; if a party aborts, a default value $\langle \perp, \perp \rangle$ is set as its input. The parties then pick the block with plurality

among the outputs, breaking ties lexicographically, and use a hash function H to compute the nonce $r = \mathsf{H}(r_1||\ldots||r_j)$.

Protocol $\pi_{\mathsf{Checkpoint}}$

A checkpointing party which runs $\pi_{\mathsf{Checkpoint}}$ is parameterized by the list $\mathbb{V}$ of n checkpointing parties, an interactive consistency protocol π_{IC}, a hash function H, a validation predicate $\mathsf{Validate}$, and k_c. It keeps a local checkpointed block, B_c, initially set to ϵ.
On receiving $(\textsc{CandidateCheckpoint}, C')$ from a party $\mathcal{V}$, check:

- $\exists i : C'[i] = B_c$ (i.e. *if C' extends the checkpoint*);
- $\mathsf{Validate}(C') = 1$ (i.e. *if C' is valid*);
- $|C'| - i = k_c$ (i.e. *if C' is long enough*).

If all hold do:

1) pick $r_j \xleftarrow{\$} \{0,1\}^\omega$;
2) execute protocol π_{IC} with the parties in $\mathbb{V}$ with input $\langle tail(C'), r_j\rangle$ and wait for its output $[\langle B_1, r_1\rangle, \ldots, \langle B_n, r_n\rangle]$;
3) find the block B_j which has plurality among the output blocks (breaking ties lexicographically) and set $B_c := B_j||\mathsf{H}(r_1||\ldots||r_n)$.

Finally, return $(\textsc{Checkpoint}, B_c)$ to $\mathcal{V}$.

Fig. 2. The protocol which is run by the checkpointing federation.

Theorem 4. *Protocol $\pi_{\mathsf{CheckpointBFT}}$ securely realizes the functionality $\mathcal{F}_{\mathsf{CheckpointBFT}}$, assuming a secure interactive consistency protocol π_{IC}, which successfully terminates, and a hash function* H.

To incorporate checkpoints, when a miner creates a new block, they submit it to all checkpointing parties via the $\mathsf{CandidateCheckpoint}$ interface of $\pi_{\mathsf{Checkpoint}}$. When the new checkpoint is issued, they accept it and, following, adopt a new chain only if it contains the newly-issued checkpoint.

IV. The Timestamped Ledger

Our second scheme aims to achieve the same security guarantees in a fully decentralized way. Figure 3 defines the *global* timestamping functionality, $\mathcal{F}_{\mathsf{Time}}$. A party can timestamp a string s by submitting the message $(\textsc{Timestamp}, s)$; afterwards, every party can verify it via the Verify interface. The timestamping functionality is *global*, so timestamps cannot be issued privately. Therefore, when a party timestamps a string, *every* other party can access both the string and its timestamp. Also the timestamp consists of both a counter and a random value, the latter mitigating the block lead attack.

A miner timestamps a new block B by submitting it to $\mathcal{F}_{\mathsf{Time}}$; $B.\tau$ denotes the timestamp of the block B. When a miner receives a candidate chain, it compares it with the local chain. Starting from genesis, it parses both chains until it finds the *timestamped* position where the two diverge, i.e. the oldest

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

$\mathcal{F}_{\mathsf{Time}}$ keeps: i) $T_{[]}$: an initially empty list of timestamped strings; ii) τ: a counter initially set to 0.
On receiving $(\textsc{Timestamp}, s)$, if $\forall(s', \cdot) \in T_{[]} : s' \neq s$, set $\tau := \tau + 1$. Then compute a list R of $p(\kappa)$ random values as $r_j \xleftarrow{\$} \{0,1\}^\omega$ and send $(\textsc{Nonce}, R)$ to $\mathcal{A}$. On receiving a response $(\textsc{Nonce}, r_i)$, such that $r_i \in R$, add (s, τ, r_i) to $T_{[]}$.
On receiving $(\textsc{Verify}, s, \tau)$, if $\exists(s, \tau) \in T_{[]}$ then return $(\textsc{VerifyTimestamp}, \top)$.

Fig. 3. The timestamping ideal functionality.

timestamped block in each chain which does not exist in the other. If such point exists then it adopts the chain with the oldest diverging block, else it employs $\mathsf{maxvalid}$.

Protocol $\pi_{\mathsf{TimeMiningRes}}$

A party that runs $\pi_{\mathsf{TimeMiningRes}}$ holds the local chain C, initially set to ϵ, and is parameterized by $\mathsf{maxvalid}(\cdot,\cdot)$.
On receiving $(\textsc{CandidateChain}, C')$, send to $\mathcal{F}_{\mathsf{Time}}$ $(\textsc{Verify}, B, B.\tau)$ for each timestamped block $B \in C'$ and wait for $(\textsc{VerifyTimestamp}, \top)$. Next:

i) set $i := 0$;
ii) while $C[i] = C'[i]$ do $i := i + 1$;
iii) set $i' := i, c := i - 1$;
iv) while $C[i]$ is not timestamped and $i < |C|$ do $i := i + 1$;
v) while $C'[i']$ is not timestamped and $i' < |C'|$ do $i' := i' + 1$;
vi) if $i = |C|$ and $i' = |C'|$ set $C := \mathsf{maxvalid}(C \setminus C[:c], C' \setminus C'[:c])$,
vii) else if $i = |C|$ or $C'[i'].\tau < C[i].\tau$ then set $C := C'$.

On receiving $(\textsc{Read},)$ return $(\textsc{Chain}, C)$.

Fig. 4. Timestamped chain resolution for miners.

Theorem 5 shows the security of $\pi_{\mathsf{TimeMiningRes}}$. An important caveats should be stressed though. If only part of the block is timestamped, e.g. its hash or its headers, $\mathcal{A}$ could keep a timestamped block's content secret, resulting in a DoS, as honest miners halt until the block is revealed. In a centralized setting, this can be prevented by ensuring that the service timestamps only fully available blocks; in the paper's full version, we show how to prevent this attack in the decentralized setting via consecutive timestamps.

Theorem 5 (Timestamping)**.** *The timestamped resolution protocol $\pi_{\mathsf{TimeMiningRes}}$ and the timestamping functionality $\mathcal{F}_{\mathsf{Time}}$ of Section IV guarantee persistence and liveness with parameter $k_c = 1$ (cf. Theorems 1 and 3).*

References

[1] V. Buterin, "On stake," 2014.

[2] V. Buterin and V. Griffith, "Casper the friendly finality gadget," 2017.

[3] V. Buterin, D. Reijsbergen, S. Leonardos, and G. Piliouras, "Incentives in ethereum's hybrid casper protocol," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul, Korea (South), May 14-17, 2019*. IEEE, 2019, pp. 236–244. [Online]. Available: https://doi.org/10.1109/BLOC.2019.8751241

[4] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, M. I. Seltzer and P. J. Leach, Eds. USENIX Association, 1999, pp. 173–186. [Online]. Available: https://dl.acm.org/citation.cfm?id=296824

[5] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, I. Goldberg and T. Moore, Eds., vol. 11598. Frigate Bay, St. Kitts and Nevis: Springer, Heidelberg, Germany, Feb. 18–22, 2019, pp. 23–41.

[6] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Advances in Cryptology – EUROCRYPT 2018, Part II*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10821. Tel Aviv, Israel: Springer, Heidelberg, Germany, Apr. 29 – May 3, 2018, pp. 66–98.

[7] P. Diamantopoulos, S. Maneas, C. Patsonakis, N. Chondros, and M. Roussopoulos, "Interactive consistency in practical, mostly-asynchronous systems," in *21st IEEE International Conference on Parallel and Distributed Systems, ICPADS 2015, Melbourne, Australia, December 14-17, 2015*. IEEE Computer Society, 2015, pp. 752–759. [Online]. Available: https://doi.org/10.1109/ICPADS.2015.99

[8] Ethereum, "Proof of stake faqs," 2018, https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs.

[9] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *FC 2014: 18th International Conference on Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, N. Christin and R. Safavi-Naini, Eds., vol. 8437. Christ Church, Barbados: Springer, Heidelberg, Germany, Mar. 3–7, 2014, pp. 436–454.

[10] M. Fitzi, "Generalized communication and security models in byzantine agreement," Ph.D. dissertation, ETH Zurich, 2002.

[11] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology – EUROCRYPT 2015, Part II*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 26–30, 2015, pp. 281–310.

[12] P. Gaži, A. Kiayias, and A. Russell, "Stake-bleeding attacks on proof-of-stake blockchains," Cryptology ePrint Archive, Report 2018/248, 2018, https://eprint.iacr.org/2018/248.

[13] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," Cryptology ePrint Archive, Report 2017/454, 2017, http://eprint.iacr.org/2017/454.

[14] B. Gipp, N. Meuschke, and A. Gernandt, "Decentralized trusted timestamping using the crypto currency bitcoin," 2015.

[15] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, no. 2, pp. 99–111, Jan. 1991.

[16] T. Hepp, P. Wortner, A. Schönhals, and B. Gipp, "Securing physical assets on the blockchain: Linking a novel object identification concept with distributed ledgers," in *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, CRYBLOCK@MobiSys 2018, Munich, Germany, June 15, 2018*. ACM, 2018, pp. 60–65. [Online]. Available: https://doi.org/10.1145/3211933.3211944

[17] A. Kiayias and G. Panagiotakos, "Speed-security tradeoffs in blockchain protocols," Cryptology ePrint Archive, Report 2015/1019, 2015, http://eprint.iacr.org/2015/1019.

[18] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology – CRYPTO 2017, Part I*, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds., vol. 10401. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 20–24, 2017, pp. 357–388.

[19] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *USENIX Security 2016: 25th USENIX Security Symposium*, T. Holz and S. Savage, Eds. Austin, TX, USA: USENIX Association, Aug. 10–12, 2016, pp. 279–296.

[20] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[21] W. Li, S. Andreina, J.-M. Bohli, and G. Karame, "Securing proof-of-stake blockchain protocols," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 297–315.

[22] B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, "Afgjort: A partially synchronous finality layer for blockchains," Cryptology ePrint Archive, Report 2019/504, 2019, https://eprint.iacr.org/2019/504.

[23] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[24] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*, G. Gibson and N. Zeldovich, Eds. USENIX Association, 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[25] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," Cryptology ePrint Archive, Report 2016/917, 2016, http://eprint.iacr.org/2016/917.

[26] ——, "Thunderella: Blockchains with optimistic instant confirmation," Cryptology ePrint Archive, Report 2017/913, 2017, http://eprint.iacr.org/2017/913.

[27] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980.

[28] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *FC 2016: 20th International Conference on Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, J. Grossklags and B. Preneel, Eds., vol. 9603. Christ Church, Barbados: Springer, Heidelberg, Germany, Feb. 22–26, 2016, pp. 515–532.

[29] F. Winzer, B. Herd, and S. Faust, "Temporary censorship attacks in the presence of rational miners," Cryptology ePrint Archive, Report 2019/748, 2019, https://eprint.iacr.org/2019/748.