On Immutability of Blockchains

Esteban Landerreche
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
esteban@cwi.nl

Marc Stevens
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
marc.stevens@cwi.nl

ABSTRACT

Recently we presented a single-party cryptographic timestamping mechanism based on proof-of-sequential-work, which we proved secure in the universal composability framework [16]. This paper describes this construction and its security claims and uses it to construct a multi-party permissioned blockchain protocol and show that it achieves an immutability notion. Finally we discuss applications of this protocol, including unpermissioned blockchains, and how these may benefit.

ACM Classification Keywords

H.1.m. Models and Principles: Miscellaneous

Author Keywords

blockchain, cryptographic immutability, cryptographic timestamping, proof-of-sequential-work

INTRODUCTION

Blockchain immutability

The primary goal of a blockchain protocol is to achieve consensus over state between all participating nodes. This process is simplified by the append-only nature of the blockchain structure. Whenever new information needs to be added to the state, a new block is created and added to the chain. With the exception of temporary forks, which are resolved by erasing the latest blocks, the state is only updated by adding new blocks. After a certain point in time, a block is considered *immutable* as it becomes unfeasible for it to be erased or changed. This fact allows nodes to reach consensus on the whole state by agreeing only on the latest changes in the state.

Immutability in blockchains has mainly been studied as an element that allows for proof-of-work (PoW) consensus, but has not been studied widely as an individual characteristic. The first comprehensive paper in Bitcoin presented the abstraction of the Bitcoin blockchain and proved its security in a partially synchronous setting [9]. This paper was followed by numerous other papers investigating different aspects of Bitcoin. The same team followed up their work with a proof of Bitcoin with chains of variable difficulty in [10]. In [21], Bitcoin is proved secure in the asynchronous model and [1] presents a fully-composable treatment of Bitcoin.

All of these papers prove that consensus in Bitcoin (more broadly, proof-of-work consensus) works in part because of the immutability provided by the blockchain structured combined with proofs-of-work.

The proof-of-work based consensus of Bitcoin provides strong guarantees for immutability. Informally, adversaries that do not have a majority of the total hashing power invested in the proof-of-work can succeed at rewriting previous blocks in the network only with success probability that is exponentially small in the depth of rewritten blocks. Moreover, even against adversaries with a fraction $\alpha > 0.5$ of the total hashing power, the adversary is computationally restricted in how deep it can rewrite blocks in a certain amount of time, *i.e.* on average after time T the adversary can only successfully rewrite blocks in the network up to $T \cdot \alpha/(1-\alpha)$ time deep.

However, there exist various problems relating to the proof-of-work based consensus of Bitcoin. First, it requires a mining reward as an incentive to compute proofs-of-work and contribute to the security of Bitcoin [20], which is natural for cryptocurrencies, but is not a generic solution for all possible applications of blockchain. Second, the mining reward causes undesired effects threatening the aimed decentralization of Bitcoin, namely it causes centralization via mining pools (for miners to obtain frequent small rewards instead of very rarely obtaining large rewards) as well as via specialization (larger more efficient mining operations can reduce costs and increase profits) [12]. Third, Bitcoin does not seem to be sustainable given that the world-wide energy consumed in computing proofs-of-work for Bitcoin now exceeds that of entire countries[1].

In light of the above issues related to proof-of-work consensus, attention is turning towards other consensus mechanisms, such as proof-of-stake [3] and proof-of-space [8]. These non-PoW systems can ensure consensus for participants which are (almost) continuously connected to the protocol [7, 15, 19]. However, the consensus guarantees achieved in the protocol do not directly apply to parties that are mostly offline. For the same reasons, these systems are vulnerable to long-range attacks where parties can easily construct multiple blockchains that returning parties cannot distinguish from the real one [22]. Proof-of-work based systems do not have this problem, as creating valid blockchains requires an investment in computational work and time.

[1] https://digiconomist.net/bitcoin-energy-consumption

Landerreche, Esteban; Stevens, Marc (2018): On Immutability of Blockchains. In: W. Prinz & P. Hoschka (Eds.), Proceedings of the 1st ERCIM Blockchain Workshop 2018, Reports of the European Society for Socially Embedded Technologies

ERCIM-Blockchain 2018 May 08–09, 2018, Amsterdam, Netherlands

ISSN 2510-2591.

DOI: 10.18420/blockchain2018_04

When getting rid of proofs-of-work, the notion of immutability is maintained by punishing misbeahavior through incentives [3] or by constantly connecting to the the protocol in order to have an updated chain [11]. While this notion might be considered sufficient in certain contexts, it provides weaker guarantees than the PoW setting. We will call this notion *weak immutability* as it is not intrinsic and requires either incentives or monitoring to be achieved. In this paper we will seek immutability in a similar sense to the one found in PoW systems. We will say a blockchain has *strong immutability* if protection against malicious changes is obtained via a computationally hard problem. Note that, similar to a complexity class, strong immutability is parametrized by the difficulty of the computational problem, meaning that a blockchain is not necessarily more secure by having strong immutability. In particular, for a PoW blockchain to be secure it must maintain honest majority against the strongest possible adversary. In practice, this means that given one choice of a PoW function, at most one blockchain which uses that function can be considered secure.

Besides this distinction being theoretically interesting, we believe that it has consequences in the practical sense as well. Most of the users of a blockchain are not involved in the protocol as miners or maintainers of the chain. In practice, these parties connect sporadically to the network to read the blockchain in order to create new transactions or verify ones that they receive. Given this behavior, strong immutability is required.

Cryptographic timestamping

The immutability of blockchains is closely linked to the timestamping problem. In the seminal Bitcoin paper [20] the author presents the blockchain as a timestamp server which requires no trusted party. In particular, efficiently rewriting blockchains can be prevented by cryptographic timestamping.

Originally, Bayer, Haber and Stornetta considered timestamping digital documents as well as digital signatures [2, 14]. The security of proposed solutions relies on trusted parties and broadcasting blockchains where any malicious behaviour will be caught.

Other proposed solutions include encoding messages in blockchain transactions, in particular in Bitcoin transactions [5, 13], where security relies on the immutability of the underlying blockchain.

Recently we presented a single-party cryptographic timestamping mechanism based on proof-of-sequential-work, which we proved secure in the universal composability framework [16]. This paper will briefly review this construction and its security claims and apply it to achieve certain notions of immutability in the multi-party blockchain setting.

PRELIMINARIES

Time.

We consider a setting where time is essentially continuous, but it may be divided into intervals of time of a certain length which will be context-dependent. For instance, when a party computes a certain slow function at a rate of γ, then a time-step for this process will be $1/\gamma$ long, but for rounds of a (network) protocol this may be a pre-agreed length of time. Parties are equipped with synchronized clocks with at most an insignificant difference in time with respect to rounds of network protocols. We assume that timestamps can be described in bitstrings of length θ at a sufficient granularity.

Public-key signatures.

We assume a public-key infrastructure Σ for digital signatures that is existentially unforgeable. I.e., we assume that no attacker will ever be able to create any kind of forgery for a public-key where he does not know the corresponding private key. Given a public key pk message m and a signature s, any party may verify it by calling the function $\Sigma.\mathsf{verify}(pk, m, s)$.

Cryptographic hash function.

Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a collision-resistant cryptographic hash function, i.e. we assume no collisions will ever be found.

Merkle Trees.

Merkle Trees are balanced binary trees, where the ordered leaf nodes are each labeled with a bitstring, and where each non-leaf node has two child nodes and is labeled by the hash of its children's labels. The root hash of a Merkle Tree equals the label of the root node. Merkle Trees allow for short set membership proofs of length $O(log(N))$ for a set of size N. For convenience we define some interface functions that deal with Merkle Trees in some canonical deterministic way.

$\mathsf{MT.root}(T)$ computes the root hash h of the Merkle Tree for some ordered finite sequence $T \in (\{0,1\}^*)^*$ of bit strings and outputs $h \in \{0,1\}^\lambda$.

$\mathsf{MT.path}(T, v)$ outputs the Merkle path described as a sequence of strings $(x_0, \ldots, x_l)$ where $x_0 = v$, $x_l = \mathsf{MT.root}(T)$, $x_i \in \{0,1\}^\lambda$ and either $x_{i+1} = \mathsf{H}(x_i \| \mathsf{H}(x_{i-1}))$ or $x_{i+1} = \mathsf{H}(\mathsf{H}(x_{i-1}) \| x_i)$ for all $i > 0$.

$\mathsf{MT.verify}(P)$ given an input sequence $P = (x_0, \ldots, x_l)$ outputs accept if P is a valid Merkle path. It outputs reject otherwise.

With a slight abuse of notation we also use $\mathsf{MT.root}(T)$ recursively, *i.e.*, if one of the elements S of T is not a bitstring but a set or sequence, we use $\mathsf{MT.root}(S)$ as the bitstring representing S. E.g., if $T = (a, b, S)$ with bitstrings $a, b \in \{0,1\}^*$ and a set of bitstrings $S = \{c, d, e\}$, then $\mathsf{MT.root}(T) = \mathsf{MT.root}((a, b, \mathsf{MT.root}(S)))$. This similarly extends to $\mathsf{MT.path}(T, v)$, *e.g.*, where $v \in S$ in the previous example.

SINGLE-PARTY CRYPTOGRAPHIC TIMESTAMPING

Proofs of Sequential Work (PoSW).

Informally proofs of sequential work are proofs that some long and inherently sequential computation was performed, whereas any verifier can quickly verify the correctness of the proof. We use an extended notion of proof of sequential work to make it variable time, where one does not have to choose the strength in advance, but whose strength continuously increases with time spent computing it. More formally, we consider a non-interactive variable-time PoSW to be a triple of algorithms ($\mathsf{PoSW.gen}$, $\mathsf{PoSW.extend}$, $\mathsf{PoSW.verify}$) with security parameter μ and parameters $g, v \in \mathbb{N}$ as defined below.

$\mathsf{PoSW.gen}(x, s)$ is a slow cryptographic algorithm that for an input $x \in \{0,1\}^*$ and strength $s \in \mathbb{N}$ computes an output $(p, s) \in \{0,1\}^\mu \times \mathbb{N}$ in $s \cdot g$ parallel time steps.

$\mathsf{PoSW.extend}$ is a slow cryptographic algorithm that for inputs $x \in \{0,1\}^*$, $(p, s) = \mathsf{PoSW.gen}(x, s)$ and $s^* \in \mathbb{N}$ returns the output $(p^*, s+s^*)$, where $(p^*, s+s^*) = \mathsf{PoSW.gen}(x, s+s^*)$, in $s^* \cdot g$ parallel time steps.

$\mathsf{PoSW.verify}(x, p, s)$ is a fast cryptographic algorithm that for inputs $x \in \{0,1\}^*$, $p \in \{0,1\}^\mu$, and $s \in \mathbb{N}$ outputs accept if $(p, s) = \mathsf{PoSW.gen}(x, s)$, and reject otherwise, in at most $s \cdot v$ time steps.

We require perfect **correctness**:

$$\mathsf{PoSW.verify}(x, \mathsf{PoSW.gen}(x, s)) = \mathsf{accept}$$

for all $x \in \{0,1\}^*$ and $s \in \mathbb{N}$. The PoSW is called **secure** if no efficient adversary given a challenge x with sufficient min-entropy can compute values (s, p) in less than $s \cdot g$ parallel time steps for which $\mathsf{PoSW.verify}(x, p, s) = \mathsf{accept}$ with non-negligible probability. The **usability** of the PoSW is the factor g/v by which verification is faster than generation of the proof.

A candidate construction that satisfies this notion is the Sloth construction by Lenstra and Wesolowski [17] that iterates modular square root and (keyed) binary permutation functions.

In this work we will assume that every party and the adversary have access to certain computational resources (a CPU running at some clock speed) or some specific optimizations which implies that they each can compute proofs of sequential work at a certain (potentially distinct) rate γ. So for every party we model their capability to compute PoSW as a slow oracle $\mathcal{F}_\gamma^{\mathsf{PoSW}}$ as defined in Algorithm 1 that beneath interacts with a global random oracle PoSW.

Algorithm 1: Oracle $\mathcal{F}_\gamma^{\mathsf{PoSW}}$

Setting: The oracle is parametrized by a PoSW-rate $\gamma > 0$. Let $\mathsf{PoSW} : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^\mu$ be a global random oracle each oracle instance has access to. The oracle also has access to a global clock clock (to exactly measure time elapsed computing the proof of sequential work).

The oracle functions as follows:

Let $Q := \emptyset$ be the (initially empty) query log;
On input (start, x) *at time* t*:*
  Update $Q \leftarrow Q \cup \{(x, t)\}$;
On input (output, x) *at time* t_o*:*
  Let t_i be the earliest time such that $(x, t_i) \in Q$, return $\perp$ if there is no such t_i;
  Let $s := \lceil (t_o - t_i) \cdot \gamma \rceil$ be the strength of the resulting proof and $p := \mathsf{PoSW}(x, s)$;
  Return (p, s) at time $t_i + s/\gamma =: t_o + \varepsilon$, with $0 \le \varepsilon < 1/\gamma$;
On input $(\mathsf{verify}, x, p, s)$*:*
  Return accept if $\mathsf{PoSW}(x, s) = p$ and reject otherwise;

The SingleLipwig protocol

In this section we present our recent single-party SingleLipwig protocol [16] and its security claims.

Consider party $\mathcal{P}$ with public key pk that can compute PoSW at rate $\gamma > 0$, who thus has access to a functionality $\mathcal{F}_\gamma^{\mathsf{PoSW}}$.

Party $\mathcal{P}$ will run protocol SingleLipwig as described in Algorithm 2.

He will wait till he obtains some msg to output, at which point he will create his next $block$ containing a hash pointer to the previous block, the msg, a PoSW and a signature.

The output of party $\mathcal{P}$ running SingleLipwig can be verified by any (γ, ε)-SingleLipwig-verifier as described in Algorithm 3, where γ is the PoSW-rate γ of $\mathcal{P}$ and $\varepsilon \ge 0$ is the time $\mathcal{P}$ requires to execute steps 3–5 of SingleLipwig. We call ε the *PoSW-interrupt time* of $\mathcal{P}$.

Algorithm 2: SingleLipwig

Setting: Assume that party $\mathcal{P}$ with public key pk has access to $\mathcal{F}_\gamma^{\mathsf{PoSW}}$ for PoSW-rate $\gamma > 0$.

Initialize $prev := \mathsf{H}(pk)$, $t_0 := \mathsf{clock}()$;
Send $(\mathsf{start}, t_0 \| prev)$ to $\mathcal{F}_\gamma^{\mathsf{PoSW}}$;
for $i = 0, \ldots$ **do**
  Wait till message (record, msg_i);
  Retrieve (p_i, s_i) by querying $\mathcal{F}_\gamma^{\mathsf{PoSW}}(\mathsf{output}, t_i \| prev)$;
  Create signature sig_i for $\mathsf{MT.root}(data_i)$, where $data_i = (\mathsf{H}(pk), i, prev, ((t_i \| prev), (p_i \| s_i)), msg_i)$;
  Set $block_i \leftarrow (data_i, sig_i)$, $prev \leftarrow \mathsf{MT.root}(block_i)$, and $t_{i+1} \leftarrow \mathsf{clock}()$;
  Send $(\mathsf{start}, t_{i+1} \| prev)$ to $\mathcal{F}_\gamma^{\mathsf{PoSW}}$;
  Output $(block_i, t_{i+1})$;
end

Algorithm 3: (γ, ε)-SingleLipwig-verifier

Input: A sequence $C = ((block_0, t_1), \ldots, (block_l, t_{l+1}))$ for public-key pk output by SingleLipwig
Output: $\perp$ or a sequence $(msg_0, age_0), \ldots, (msg_l, age_l)$

Output $\perp$ if not all $block_i$ are correctly signed: $block_i = (data_i, sig_i)$ where sig_i is a valid signature of $\mathsf{MT.root}(data_i)$ for public key pk;
Output $\perp$ if not all $data_i$ are correctly formed: $data_i = (\mathsf{H}(pk), i, prev_i, posw_i, msg_i)$ where $prev_0 = pk$, $prev_i = \mathsf{MT.root}(block_{i-1})$, and $posw_i = ((t_i \| prev_i), (p_i, s_i))$ for some msg_i, p_i, s_i;
Output $\perp$ if not all $posw_i$ are correct PoSW: $posw_i = ((t_i \| prev_i), (p_i, s_i))$ where $p_i = \mathsf{PoSW}(t_i \| prev_i, s_i)$;
Output $\perp$ if not all PoSW are strong enough: $s_i \ge (t_{i+1} - t_i - \varepsilon) \cdot \gamma$;
Output $\{(msg_i, age_i = t_{l+1} - t_{i+1}) \mid 0 \le i \le l\}$;

The following properties of the algorithm SingleLipwig together with a (γ, ε)-SingleLipwig-verifier have been proven in the universal composability framework for sufficiently large security parameters κ, λ and μ.

THEOREM 3.1 ([16]). *(1) The output of an honest $\mathcal{P}$ running the algorithm* SingleLipwig *with PoSW-rate γ and PoSW-interrupt time ε will be accepted by any (γ, ε)-*SingleLipwig*-verifier.*

*(2) When a honest $\mathcal{P}$ is corrupted at time T_{corr} by an adversary with PoSW-rate $\gamma \cdot \alpha$ and PoSW-interrupt time ε/α with $\alpha \geq 1$, any sequence $C = ((block_0, t_1), \ldots, (block_l, t_{l+1}))$ output by the adversary at time T_{output} that is accepted by any (γ, ε)-*SingleLipwig*-verifier satisfies the following properties except with negligible probability:*

1. *Let $A = T_{output} - T_{corr}$ be the time passed since corruption, then either C contains all $block_i$ created by the honest $\mathcal{P}$ at least $A \cdot \alpha$ time ago or none.*
2. *Any $block_i$ created by the adversary has claimed age $age_i = t_{l+1} - t_{i+1}$ (cf. Algorithm 3) at most $A \cdot \alpha$.*
3. *Any $block_i$ created by the adversary at time T_i has claimed age age_i at most $(T_{output} - T_i) \cdot \alpha$.*

PERMISSIONED BLOCKCHAINS

Permissioned blockchains could be a solution to many practical problems faced by governments and enterprises. Cooperation between mutually mistrusting entities can only emerge when a party has assurances that power will not be abused. In practice, these cooperations can only happen through trust on a third party (most likely the government). These processes take considerable time, energy and money, elevating costs and affecting efficiency. Blockchains can solve these issues by providing a trusted ledger which provides the necessary assurances.

Permissioned blockchains should provide a ledger that any participant can add to according to preset rules. Distributed consensus means that no single entity has enough power to arbitrarily modify the ledger. There exist multiple efficient protocols for consensus in a permissioned network. Permissioned consensus requires a trust in identities (through a PKI or otherwise), trust that enough parties will not collude. Distributed consensus has existed for a long time, so what exactly do blockchains offer beyond distributed consensus?

Compared to classical consensus protocols, blockchains offer two primary advantages: the ability to function in an unknown network and cryptographic immutability. The former is irrelevant in the context of a permissioned network, especially because this comes at a large efficiency (and economic) cost. Therefore, the main feature of blockchains in this setting is immutability. Blockchains prevent any entity from arbitrarily rewriting anything written in it. The blockchain structure ensures that any change propagates throughout the whole chain. Changing a block becomes equivalent to recreating the entire succeeding section of the chain. If creating each block in the blockchain is *hard* enough, we can consider it immutable.

Immutability is generally studied through its relationship with consensus. In non-PoW blockchains, it is shown that the adversary cannot disrupt consensus on the recent state of the chain. However, these systems are vulnerable to long-range attacks, as it is easy to emulate the execution of the protocol [22]. If a sufficient number of identities become adversarial at some point in time then in principle they can efficiently rewrite the entire ledger almost immediately. In a permissioned network consisting of a known (possibly small) number of known parties, the risk of an adversary corrupting enough parties is especially relevant (particularly if all parties use the same software, which should be expected) [23].

Bitcoin's PoW paradigm avoids this problem because rewriting a chain implies re-doing all the necessary work. Beyond hash collisions, rewriting a block is equivalent to disrupting consensus. In PoW-based systems, even if consensus breaks down because of an adversarial majority, the blockchain still offers some guarantees against rewriting. Alternatively, if a blockchain does not use proofs-of-work, an adversarial majority can, in principle, efficiently and arbitrarily rewrite everything whenever it gains control of a majority of the network. We postulate that this resistance to modification even with a majority adversarial corruption is the strongest advantage of a blockchain in a permissioned setting.

Unfortunately, proof-of-work consensus does not provide these guarantees in a permissioned setting. Security in proof-of-work blockchains comes from the amount of computational power that the adversary has access to, in comparison with the network. In large permissionless networks like Bitcoin and other cryptocurrencies, it is unfeasible for an adversary to have access to more computational power than the rest of the network combined. Therefore, a new blockchain must have a network with enough computational power to be resilient to attacks from the networks that maintain these cryptocurencies. For example, this is what causes merged-mining sidechains to be vulnerable to rewriting attacks [6].

Sensing this attack vector, multiple cryptocurrencies have adopted different proof-of-work functions that weaken these attackers. Bitcoin mining today relies on application-specific integrated circuits (ASICs); it is possible to choose a function where this specialized hardware provides little advantage. Multiple cryptocurrencies have chosen this road in order to sidestep this problem [4]. This solution does not work in a permissioned setting, as these networks are likely to be small. A malicious party could easily get access to enough computational power to rival that in the network. The only way to prevent this is to invest considerable amounts of computational power in the network, which is not cost-effective. As permissioned networks do not require proofs-of-work to achieve consensus and do not achieve any immutability from them, we believe that proofs-of-work have no place in the permissioned setting, but their role can be filled by proofs-of-sequential-work.

A Simple Protocol

Using proofs-of-sequential-work prevents arbitrary rewriting the content inside of a blockchain, even if this blockchain is maintained by only one party. However, this is not enough to ensure immutability and provide the trust guarantees that are expected from a blockchain. Proofs-of-sequential-work prevent against rewriting something in the history, but cannot prevent real-time forking. The creator of the chain could secretly maintain several forks of its blockchain and choose

which one to present depending on the situation. A corrupt agent cannot arbitrarily rewrite a chain but can still make modifications to it in certain conditions. Like any other blockchain protocol, we require multiple parties to cooperate to achieve a stronger immutability. What is more interesting is that a single honest party can prove whether or not a blockchain has been rewritten in a definitive way.

In our previous model there was only one party, which meant that the time spent between instances of the proof of sequential work (denoted by ε) was as small as possible.

We are interested in minimizing this time, but run into a problem in the multi-party setting where achieving consensus on the next block takes significant time. If we compute our PoSW over blocks in the chain, the time between a block being proposed and it being confirmed is time where PoSW cannot be computed. While getting a larger γ requires an investment to acquire a faster processor, an adversary can gain an advantage in ε by creating the block on its own. Even if in practice this process is almost instantaneous, it forces us to allow *weaker* PoSWs; proofs that do not span all the time needed to create the block. Computing PoSW over the blocks of the chain gives the adversary a strong advantage for rewriting blocks. Instead, we propose each participant will maintain their own personal PoSW chain and together maintain a ledger chain that interacts with the personal chains.

We will present a simplified model with favorable network conditions and access to an arbitrary consensus mechanism that has certain desirable properties as described in Algorithm 4. Our goal will not be to show that consensus is achieved but instead to show the immutability properties provided by the blockchain. Besides the advantages provided by the PoSWs to SingleLipwig, this protocol will additionally benefit from additional guarantees provided by signatures of each block. In this simplified construction, we will assume the existence of some protocol Consensus that achieves certain minimum security properties which determines the block created at each round of the protocol.

The protocol MultiLipwig, presented in Algorithm 5, will interact with other parties using Consensus to determine the next ledger block and locally use SingleLipwig to record its copy of the ledger block. So each party will record two distinct chains: its personal chain and the ledger chain. Blocks in the ledger chain will be represented by $block_i^L$, while blocks in party $\mathcal{P}_j$'s personal chain will be represented by $block_i^j$.

In order to link the personal blockchains with the ledger chain, we will add Merkle pointers to the blocks. For simplicity, we will act as if the participants simply copy the entire blocks. Each personal block will contain the ledger block created that round. The new ledger block output by Consensus will contain the personal blocks created in the previous round[2], in order to provide the ledger chain with the security provided by the proofs-of-sequential-work. While the PoSW are computed over the personal chains and not the ledger chain, every personal block contains the ledger block. Therefore, we can create a Merkle path between the root of the ledger block up to the root of a personal block, which is the input to the PoSW. Because we assume that no collisions can be found for the hash function underlying our Merkle trees, the query to $\mathcal{F}_\gamma^{\mathsf{PoSW}}$ could only come after the ledger block was created. This allows the ledger chain to *borrow* the proof-of-sequential-work from the personal chains.

[2]In practice, personal chains will contain a simple Merkle root while the ledger chain will contain Merkle paths to the PoSW, the pointer to the ledger block and the signature.

Algorithm 4: Consensus for n parties

Setting: Let $\mathcal{P}_j$ running this protocol in a network of n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with corresponding public keys $pk_1, \ldots, pk_n$. The desired consensus protocol is parametrized by a minimum number $n/2 < Thr \leq n$ of contributing parties and a maximum run time T.

Input: Ledger chain $C^L = (block_1^L, \ldots, block_{i-1}^L)$ and SingleLipwig chain $C^j = ((block_0^j, t_1^j), \ldots, (block_{i-1}^j, t_i^j))$.

Result:
The protocol runs within time T and either outputs $(block_i^L, sig_{i,j}^L)$ or abort. The new block $block_i^L$ is of the following form:

$$\left(pkH, i, prev_i, \{block_i^k \mid k \in P_i\}, msg_i\right)$$

where $pkH = \mathsf{MT.root}(pk_1, \ldots, pk_n)$, $prev_1 = pkH$ for $i = 1$ or $prev_i = \mathsf{MT.root}(block_{i-1}^L)$ otherwise, $|P_i| \geq Thr$ and $block_i^k$ is the last block in the SingleLipwig chain C^k of $\mathcal{P}_k$. The content msg_i is also decided by the protocol, however the format and validity of msg_i is application specific and therefore left unspecified here.
If there are at least *Thr* honest parties then all honest parties receive the same output $block_i^L$ and every honest party $\mathcal{P}_k$ receives a set $sig_{i,k}^L$ of signatures on $block_i^L$ of at least *Thr* distinct parties.
In all cases, even if all other parties are corrupt, if $\mathcal{P}_j$ is honest then it signed at most one candidate $\widetilde{block_i^L}$ that must be valid in the above form including a valid $\widetilde{msg_i}$.

The Consensus algorithm must wait until the threshold of participants submit their blocks. We require the threshold to be at more than half the players to allow for external verification. After that, it waits for an random amount of time before outputting a block to all participants, containing all the blocks for the round that it received. Note that if the block is not correctly formatted, does not contain a pointer to the latest block Consensus created or the signature is not valid, then it is not added to the block.

As we did for SingleLipwig, we will define a (γ, ε)-MultiLipwig-verifier to verify chains output by MultiLipwig in Algorithm 6. This verifier will certify the ledger chain (found encoded in the personal chain) and call the SingleLipwig-verifier on the personal chain.

Theorem 4.1. *(1) If there are at least Thr honest parties, then the output of any $\mathcal{P}_j$ running* MultiLipwig *with PoSW-*

Algorithm 5: MultiLipwig

Setting: Let $\mathcal{P}_j$ running this algorithm in a network of n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with corresponding public keys $pk_1, \ldots, pk_n$. Assume each has the same PoSW-rate γ and PoSW-interrupt time ε.

$\mathcal{P}_j$ starts running SingleLipwig;
Retrieve $(block_0^j, t_1^j) \leftarrow$ SingleLipwig(record, $\perp$);
Set $C^j \leftarrow ((block_0^j, t_1^j))$, $C^L = \varnothing$;
for $i = 1, \ldots$ **do**
  Call Consensus(C^L, C^j) to obtain $(block_i^L, sig_{i,j}^L)$;
  Call SingleLipwig(record, $(block_i^L, sig_{i,j}^L)$) to obtain $(block_i^j, t_{i+1}^j)$;
  Append $block_i^L$ to C^L;
  Append $(block_i^j, t_{i+1}^j)$ to C^j;
  Output $(block_i^j, t_{i+1}^j)$;
end

Algorithm 6: (γ, ε)-MultiLipwig-verifier

Input: An MultiLipwig sequence $C^j = ((block_0^j, t_1^j), \ldots, (block_{l+1}^j, t_{l+2}^j))$
Output: $\perp$ or a sequence $(msg_0, age_0), \ldots, (msg_l, age_l)$

Call (γ, ε)-SingleLipwig-verifier for C^j;
if *Verifier outputs* $(((block_1^L, sig_{1,j}^L), age_1), \ldots, ((block_{l+1}^L, sig_{l+1,j}^L), age_{l+1}))$ **then**
  Output $\perp$ if not all $block_i^L$ are correctly formed: $block_i^L = \left(pkH, i, prev_i, \{B_i^k \mid k \in P_i\}, msg_i\right)$ where $pkH = \mathsf{MT.root}(pk_1, \ldots, pk_n)$, $prev_1 = pkH$, $prev_i + 1 = \mathsf{MT.root}(block_i^L)$, $|P_i| \geq Thr$ and **if** $j \in P_i$ **then** $B_i^j = block_{i-1}^j$;
  Output $\perp$ if $|sig_{i,j}^L| < Thr$ for some i;
  Output $\perp$ if there are invalid signatures in some $sig_{i,j}^L$ There exists $s \in |sig_{i,j}^L|$ for some i such that for all $k \in [n]$, $\Sigma.\mathsf{verify}(pk_k, block_i^L, s) = \mathsf{reject}$;
  Output $(((msg_1), age_1), \ldots, (msg_{l+1}), age_{l+1}))$
else
  Output $\perp$
end

*rate γ and PoSW-interrupt time ε will be accepted by any (γ, ε)-*MultiLipwig*-verifier. Furthermore, the output of any adversarial party that is accepted by any (γ, ε)-*MultiLipwig*-verifier contains the same ledger chain C^L as the output of* all *honest parties.*

*(2) If there are less than Thr honest parties but also less than Thr corrupted parties, then the output of any adversarial party that is accepted by any (γ, ε)-*MultiLipwig*-verifier contains a ledger chain C_i^L where each $block_i^L$ is verified and signed by* some *honest party. No consensus is guaranteed.*

*(3) Suppose that Thr parties fall under adversarial control at time T_{corr}. Given an adversary with PoSW-rate $\gamma \cdot \alpha$ and PoSW-interrupt time ε/α with $\alpha \geq 1$, any sequence $C = ((block_0, t_1), \ldots, (block_l, t_{l+1}))$ output by the adversary at time T_{output} that is accepted by any (γ, ε)-*MultiLipwig*-verifier satisfies the following properties except with negligible probability:*

1. *Let $A = T_{output} - T_{corr}$ be the time passed since corruption, then every $block_i^L$ in C of at least $A \cdot \alpha$ time ago is verified and signed by* some *$\mathcal{P}$ that was honest at that point in time.*
2. *Any $block_i^L$ created by the adversary has claimed age $age_i = t_{l+1} - t_{i+1}$ at most $A \cdot \alpha$.*
3. *Any $block_i^L$ created by the adversary at time T_i has claimed age age_i at most $(T_{output} - T_i) \cdot \alpha$.*

Proof. Part (1) follows from correctness of consensus and SingleLipwig. Part (2) follows from the fact at least *Thr* signatures are required by the MultiLipwig-verifier, requiring participation of at least one honest party. Part (3) follows from Theorem 3.1 and part (2). □

Note that while Consensus requires an honest *Thr*-majority to ensure agreement, without a corrupted *Thr*-majority the adversary still cannot create new ledger blocks without the verification and signature by some honest party. Thus the adversary must actually control a *Thr*-majority of parties to start rewriting blocks without any verification by an honest party. But most importantly, even if the adversary completely corrupts the entire permissioned network then still it is limited by how many blocks it can rewrite over a period of time that would be accepted by any external verifier.

Our construction of the verifier assumes that every ledger block contains a certificate of correctness (the set of signatures) which can be forged the moment the adversary gains control of enough parties. Different consensus mechanisms may have different certificates of correctness, like the block hash in proof-of-work systems. The verifier can be modified for different consensus mechanisms, but in that case the adversary must not only corrupt enough parties but also be able to create valid certificates. The need for certificates can be avoided by requiring the verifier to certify that the personal blocks contained in each ledger block are correctly constructed and contain both a signature and a pointer to the previous ledger block. Signatures in personal blocks can be considered a commitment to the ledger block that they point to. The alternative construction implies a higher verification cost in exchange for a possibly cheaper consensus mechanism.

APPLICATIONS

We have extended our single party protocol to a permissioned blockchain. We now briefly present possible applications for our constructions, including some outside the classical realm of blockchains.

Semi-private Databases

Advantages of blockchain technology can be also applied to a centralized setting, in cases where a party is partially trusted. Assume that an entity (for example a government) maintains a database of important records (for example, a land registry) for which temporality is important. This entity is interested in maintaining the entirety of this database private, while making entries available to the right parties. In the case of a land registry, if a party wants to know the particular status of a piece of land, they can request this information and get a response. As the database is not public, this party has no direct way to verify that the information is actually in the registry. There are ways to prove that the record is in the database, but it is harder to prove *when* the record was added.

In parts of Mexico City there are residential areas where land must only be used for housing unless the land was used for something other than a house from before this law was passed. In order to use these residences as commercial units, bureaucrats are bribed to forge documents that state that the property has always been used for commercial purposes [18] . Having a history as a commercial unit allows the owner to "legally" rent it as a commercial space. If there was a PoSW-secured database containing these records, it would be possible to prove that the forged documents are not as old as they claim to be, preventing this instance of corruption.

Permissionless Blockchains

On the other side of the spectrum, a permissionless blockchain can also benefit from *immutability blockchains* created by SingleLipwig or MultiLipwig, in particular ones without proofs-of-work.

A non-intrusive method is to use such an immutability blockchain that collects hashes of blocks from the permissionless blockchain. It can act as a sort of *immutability beacon* providing time-lock guarantees about the permissionless chain to any verifier interacting with the immutability blockchain, for instance during bootstrapping.

In a more intrusive manner, any permissionless blockchain can *borrow* the immutability from the permissioned blockchain by embedding its blocks into the permissionless blockchain, much in the same way that the permissioned ledger chain *borrows* the immutability from the personal blockchains [16].

In both cases, this requires minimal commitment and trust of the set of parties executing the immutability blockchain. As long as this immutability blockchain continues to include hashes of blocks and correctly compute proofs-of-sequential work, both easily verifiable, any party can benefit from the proofs of age provided by it. Moreover, it is easy to start a new trusted permissioned group that does the same and that can actually use the entire immutability chain of any previous group up to that point in time[3], making it easy to switch between immutability blockchains if desired.

In both cases, it can also aid light-weight clients and in fast bootstrapping, since the immutability chain is significantly smaller in data size than the permissionless chain and is quickly verified. Then light-weight and/or bootstrapping parties can more easily rely on the validity of the history and focus on verification of recent blocks.

REFERENCES

1. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. 2017. Bitcoin as a Transaction Ledger: A Composable Treatment.. In *Annual International Cryptology Conference*. Springer, 324–356.
2. Dave Bayer, Stuart Haber, and W Scott Stornetta. 1993. Improving the efficiency and reliability of digital time-stamping. *Sequences II: Methods in Communication, Security and Computer Science* (1993), 329–334.
3. Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies without proof of work. In *International Conference on Financial Cryptography and Data Security*. Springer, 142–157.
4. Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. 2015. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 104–121.
5. Jeremy Clark and Aleksander Essex. 2012. CommitCoin: Carbon Dating Commitments with Bitcoin - (Short Paper). In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers (Lecture Notes in Computer Science)*, Angelos D. Keromytis (Ed.), Vol. 7397. Springer, 390–398. DOI: http://dx.doi.org/10.1007/978-3-642-32946-3_28
6. Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, and others. 2016. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*. Springer, 106–125.
7. Phil Daian, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919. (2016). http://eprint.iacr.org/2016/919.
8. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2015. Proofs of Space. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II (Lecture Notes in Computer Science)*, Rosario Gennaro and Matthew Robshaw (Eds.), Vol. 9216. Springer, 585–605. DOI: http://dx.doi.org/10.1007/978-3-662-48000-7_29

[3]This requires only a minimal change to MultiLipwig and its verifier where the genesis ledger block includes this borrowed chain which has to be verified in a recursive manner.

9. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2014. The Bitcoin Backbone Protocol: Analysis and Applications. Cryptology ePrint Archive, Report 2014/765. (2014). http://eprint.iacr.org/2014/765.
10. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The bitcoin backbone protocol with chains of variable difficulty. In *Annual International Cryptology Conference*. Springer, 291–323.
11. Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Stake-Bleeding Attacks on Proof-of-Stake Blockchains. Cryptology ePrint Archive, Report 2018/248. (2018). https://eprint.iacr.org/2018/248.
12. Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. 2018. Decentralization in Bitcoin and Ethereum Networks. *arXiv preprint arXiv:1801.03998* (2018).
13. Bela Gipp, Norman Meuschke, and André Gernandt. 2015. Trusted Timestamping using the Crypto Currency Bitcoin. *iConference 2015 Proceedings* (2015).
14. Stuart Haber and W. Scott Stornetta. 1991. How to time-stamp a digital document. *Journal of Cryptology* 3, 2 (01 Jan 1991), 99–111. DOI: http://dx.doi.org/10.1007/BF00196791
15. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*. Springer, 357–388.
16. Esteban Landerreche, Marc Stevens, and Christian Schaffner. 2018. Cryptographic timestamping through sequential work. Preprint. (2018). https://marc-stevens.nl/research/papers/preprint_LSS18_Cryptographic-timestamping.pdf.
17. Arjen K. Lenstra and Benjamin Wesolowski. 2015. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366. (2015). http://eprint.iacr.org/2015/366.
18. Patricia López Moreno. 2017. Irrregularidades en los Procesos y Autorizaciones de las Manifestaciones de Construcción. (February 2017). http://www.impunidadcero.org/uploads/app/articulo/23/archivo/1486526151A57.pdf Impunidad Cero.
19. Silvio Micali. 2016. Algorand: The efficient and democratic ledger. *arXiv preprint arXiv:1607.01341* (2016).
20. Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
21. Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 643–673.
22. Andrew Poelstra. 2014. Distributed consensus from proof of stake is impossible. (2014).
23. Emin Gün Sirer. 2017. What Could Go Wrong? When Blockchains Fail. (2017). http://events.technologyreview.com/video/watch/emin-gun-sirer-cornell-when-blockchains-fail/ Business of Blockchain.