Executing large-scale processes in a blockchain

Mahalingam Ramkumar
Department of Computer Science and Engineering, Mississippi State University, Starkville, Mississippi, USA

Received 16 May 2018
Revised 16 May 2018
Accepted 1 September 2018

Abstract

Purpose – The purpose of this paper is to examine the blockchain as a trusted computing platform. Understanding the strengths and limitations of this platform is essential to execute large-scale real-world applications in blockchains.
Design/methodology/approach – This paper proposes several modifications to conventional blockchain networks to improve the scale and scope of applications.
Findings – Simple modifications to cryptographic protocols for constructing blockchain ledgers, and digital signatures for authentication of transactions, are sufficient to realize a scalable blockchain platform.
Originality/value – The original contributions of this paper are concrete steps to overcome limitations of current blockchain networks.

Keywords Cryptography, Blockchain, Scalability, Transactions, Blockchain networks

Paper type Research paper

1. Introduction

A blockchain broadcast network (Bozic *et al.*, 2016; Croman *et al.*, 2016; Nakamoto, 2008; Wood, 2014) is a mechanism for creating a distributed, tamper-proof, append-only ledger. Every participant in the broadcast network maintains a copy, or some representation, of the ledger. Ledger entries are made by consensus on the states of a process "executed" on the blockchain.

As an example, in the Bitcoin (Nakamoto, 2008) process:

(1) Bitcoin transactions transfer Bitcoins from a wallet to one or more other wallets.

(2) Bitcoins created by "mining" are added to the miner's wallet.

(3) Bitcoin process state is the unspent balance in each wallet.

The identity A of a Bitcoin wallet is a public key of an asymmetric digital signature scheme. The possessor of the corresponding private key can initiate signed transactions to transfer Bitcoins from her wallet to other wallets. For example, a Bitcoin transaction:

$$\mathbf{T}_i = [t_i, X, x, Y, y]_A \tag{1}$$

signed by A, and broadcast at time t_i, is for transferring (from wallet A), x Bitcoins to wallet X, and y Bitcoins to wallet Y.

The transaction $\mathbf{T}_i$ is deemed well-formed only if $x + y \leqslant a$, where a is the unspent balance in wallet A before the transaction. Only well-formed transactions are added to the Bitcoin-distributed ledger (ill-formed transactions are ignored). More specifically, a plurality of well-formed transactions is added to a block, and such blocks are added to the ledger, to create a "chain of blocks."

Journal of Capital Markets Studies
Vol. 2 No. 2, 2018
pp. 106-120
Emerald Publishing Limited
2514-4774
DOI 10.1108/JCMS-05-2018-0020

While the Bitcoin network is intended for a single fixed process, namely, tracking the state of Bitcoin wallets, more recent blockchain networks like Ethereum (Wood, 2014) are intended for running any number of flexible, software defined, processes. Ethereum provides a JavaScript-like programming language for implementing functions triggered by various types of transactions, along with a virtual machine for executing of such functions.

1.1 Auditing the ledger

A blockchain network can be justifiably regarded as a universally trusted platform for executing processes, as the trust is based solely on good cryptographic assumptions, namely, the quantifiable preimage and/or collision-resistance strength of a cryptographic hash function $h(\ )$. Blockchain ledger entries are a record of progression of states of the process. Protocols constructed using a standard cryptographic hash function $h(\ )$ make it possible to create an open, distributed, immutable (for past entries), append-only ledger that can be audited by anyone. In other words, the integrity guarantees regarding a process executed on a blockchain network stem from the fact that anyone can reliably audit the entire history of all process states.

However, the fact that anyone can audit a blockchain ledger does not imply that everyone will. To address this issue, some participants are explicitly incentivised to do so. In other words, blockchain participants can be seen as belonging to two broad categories:

(1) a small number of incentivised participants, who take an active part in making ledger entries; and

(2) a much larger number (typically) of passive participants, who do not take part in the process of making ledger entries, but can nevertheless audit the ledger, if they choose to do so.

Typically, every participant in a blockchain network maintains a copy of the ledger. Periodically, after a set of transactions have been processed, an incentivised user "makes a motion" to add a block to the blockchain. Most often, such a motion passes, and every participant updates their copy of the ledger. The reason that incentivised users do not make motions to add blocks with ill-formed transactions is due to the fact that they have a stake in the correctness of their ledger entries. Two common types of incentive mechanisms (Bentov *et al.*, 2014) include Proof-of-Stake (PoS) and Proof-of-Work (PoW).

In PoS-based (Kiayias *et al.*, 2017) incentive systems, incentivised participants are required to explicitly stake some amount on the correctness of their ledger entries. Any error (deliberate or otherwise) will result in loss of stake.

In PoW schemes (Nakamoto, 2008), the stake is in the form of expensive energy invested by incentivised participants ("miners" in Bitcoin) to solve a computationally intensive puzzle. In general, the puzzle itself has nothing to do with the process executed in the Blockchain. The incentive for miners to ensure correctness of entries is that a miner's expensive work may be rendered moot if they make an erroneous entry. More specifically, in Bitcoin, solving the puzzle has two purposes: to gain the privilege of adding a block to the Bitcoin ledger; and "mine" a certain number of new Bitcoins. The main shortcoming of PoW incentives stems from growing sustainability concerns (Digital Trends) – the annual energy cost for mining Bitcoins is estimated to be already over a billion dollars (Digiconomist).

While regular Bitcoin users may not solve puzzles, they are expected to audit the well-formedness of every ledger entry. Users who interact only intermittently with the blockchain network can still sync their copy of the ledger by downloading and examining all transactions that occurred since their last sync.

1.2 Cryptographic hash functions

Central to blockchain networks is a cryptographic hash function $h(\ )$ for computing a succinct commitment to the ledger. The power of a cryptographic one-way hash function $h(\ )$ is deceptively simple – it is simply a mechanism to quantify certainty in the chronological order in which two events occurred. Specifically:

$$y = h(x) \Rightarrow x \in \{0,1\}^{*} \textit{ existed before } y \in \{0,1\}^{n}. \tag{2}$$

More specifically, given two sequences of bits (or bit-strings) x, y, where $h(\ )$ transforms a preimage $x \in \{0,1\}^{*}$ (bit-string of any length), to a digest $y \in \{0,1\}^{n}$ (bit-string of fixed length n), one can conclude with a very high degree of certainty that "x existed before y." More specifically, the uncertainty in such a claim is? (2^{-n}). In other words, for large enough n (say, $n \geqslant 128$ bits) it is impractical to choose a digest first, and determine a suitable preimage later.

In current blockchain networks the most commonly employed hash function $h(\ )$ is the hash standard SHA-2 with 256-bit digests. The specific utility of the hash function $h(\ )$ in a blockchain is that it enables computation of a single hash (a 256-bit SHA-2 hash) α, as the commitment to the entire ledger. More specifically:

(1) a Merkle (1987) hash tree is used to compute a commitment (a hash) v_i to each block; and

(2) a hash accumulator (Bayer *et al.*, 1993) is used to compute the commitment α to a chain of values $v_1, v_2, \ldots$.

The explicit consensus between all users (at any specific time) is on the precise value of α. Due to the properties of $h(\ )$ (and more complex constructions using $h(\ )$), an explicit consensus on α is an implicit consensus on every ledger entry.

1.3 Scalable blockchain processes

The practical utility of executing processes on a universally trusted platform (like a blockchain network) is that such as platform can eliminate, or substantially reduce the scope of, expensive infrastructures in the form of organizations like banks, insurance companies and even governments. However, real-world processes needed to replace such infrastructures may have possibly billions or even trillions of process states. For such large-scale processes, it is impractical for every blockchain participant to audit the complete history of all process states. More specifically, while some active participants may be incentivised (or paid) to audit every transaction, it is impractical to expect passive users, who may only participate intermittently, to do so.

Realizing scalable blockchains calls for strategies that permit passive users to selectively audit the correctness of any specific ledger entry. More specifically, scalable blockchains should strive to reduce the overhead necessary for passive and intermittent participants to perform selective audits.

Second, proactive strategies to eliminate ambiguities are essential for universal consensus, and thus, prevent forking of the blockchain. Consider a scenario where two miners A and B provide two different correct solutions to a puzzle (for adding a block). Such a state can cause forking of the blockchain, where different users may sync up with different forks. The practical implication of a forked ledger is that users following different forks have different interpretation of the truth, namely, the actual state of the process. For example, the state of wallet A and B will be different in both forks (A is the beneficiary of the mined Bitcoins in one fork and B in the other). If the reason for forking is due to an erroneous entry in one fork (e.g. inclusion of an ill-formed transaction), reducing the overhead for selective audits enables passive users to readily identify and follow the correct fork. Reasons for multiple forks without ill-formed transactions can be due to ambiguities in the order of transactions, and possibly even the interpretation of the process.

Third, it is essential to reduce the susceptibility of the blockchain broadcast network to clogging attacks (Oppliger, 1999). One common form of clogging attack involves an attacker sending a transaction with random bits as "signature." Only after performing expensive asymmetric cryptographic computations to verify the "signature," will verifiers realize that the signature is invalid. Clogging attacks are an especially serious concern in broadcast networks, as it takes very little effort for an attacker to send random bits, to expend computational resources of a large number of receivers.

This paper proposes multiple strategies aimed at addressing the three requirements above. Specifically:

(1) Toward reducing overhead for intermittent passive users to sync up with the current state of the ledger, a hash calendar (Buldas and Saarepera, 2014) is proposed as a better alternative to the hash accumulator.

(2) Toward reducing the overhead for selective audits, an ordered Merkle tree (OMT) (Ramkumar, 2014) is used to capture succinct commitments to process states.

(3) Two strategies are proposed for eliminating ambiguities that may lead to forking:

- the first is the use a separate timestamp ledger (TSL); and
- the second is to interpret any blockchain process as a finite state machine (FSM), where every blockchain transaction is associated with an unambiguous next-state function.

(4) Finally, to address clogging attacks, the use of expensive asymmetric cryptography-based digital signatures is eliminated.

The rest of this paper is organized as follows. Section 2 outlines cryptographic protocols for scalable blockchains. Section 3 outlines strategies for:

(1) introducing checkpoints in ledger entries to enable selective audits; and

(2) unambiguous description of (FSM) next-state functions as explicit predicates – in the form of:

- preconditions (predicates to commence the state transition); and
- postconditions (predicates on completion of the transition).

Section 4 outlines the architecture for a scalable blockchain that takes advantage of strategies outlined in Sections 2 and 3. Conclusions are offered in Section 5.

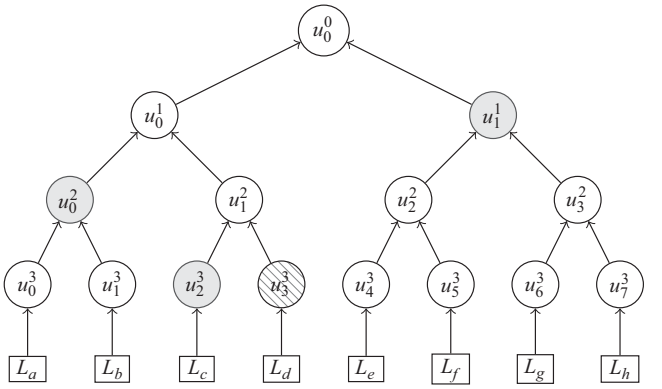
2. Cryptographic protocols

Hash chain based protocols discussed in this section include Merkle (1987) hash trees, OMT (Ramkumar, 2014), hash accumulators (Bayer *et al.*, 1993), hash calendar (Buldas and Saarepera, 2014) and the timed efficient stream loss-tolerant authentication (TESLA) broadcast authentication protocol (Perrig *et al.*, 2000).

2.1 Merkle hash tree

A binary Merkle (1987) hash tree (Merkle, 1987) of depth d has 2^i nodes at each of the depths $0 \leqslant I \leqslant d$ nodes. Figure 1 depicts a tree with depth $d = 3$. The $N = 2^3 = 8$ nodes at depth $d = 3$ are leaf nodes. Internal nodes have two child nodes – a left child and a right child. Specifically, an internal node u_i^k at depth k is related to its two child nodes u_{2i}^{k+1} and u_{2i+1}^{k+1} at depth $k+1$ as follows:

$$u_i^k = h\left(u_{2i}^{k+1}, u_{2i+1}^{k+1}\right). \tag{3}$$



Notes: Gray shaded nodes u_2^3; u_0^2 and u_1^1 are complementary to (hatched) leaf node $u_3^3 = h(L_3)$

Figure 1.
A binary Merkle hash tree

Corresponding to every leaf (non-internal) node at depth d, are d verification objects (VOs), one in each of the levels $d, d-1, \ldots, 1$. The sets of d VOs $\mathbf{u}_i$ of a leaf node u_i^d are nodes complementary to u_i^d, as they are commitments to all leaf nodes except u_i^d.

Typically, a leaf node $u_i^d = h(L)$, where L is a leaf. Thus, for any leaf L there is a sequence of d hash operations $f_{bt}(\,)$, namely:

$$r = f_{bt}(h(L), \mathbf{u}_i), \tag{4}$$

which outputs the root r of the tree. For example, consider leaf L_d in Figure 1 with leaf node $u_3^3 = h(L_d)$. The three VOs of u_3^3 are $\mathbf{u} = \{u_2^3, u_0^2, u_1^1\}$. The sequence of $d = 3$ hash operations are as follows:

$$r = f_{bt}(u_3^3, \{u_2^3, u_0^2, u_1^1\}) = h(h(u_0^2, h(u_2^3, u_3^3)), u_1^1). \tag{5}$$

In Equation (4), the fact that the output of $f_{bt}(\,)$ is r is proof that "the leaf L (and the VOs $\mathbf{u}_i$ of L) should have existed before the root r was computed." In other words, given (a tree with) root r, this constitutes proof of existence of the leaf L and its VOs $\mathbf{u}_i$ in the tree.

In most blockchain networks, $N \geqslant 1$ well-formed transactions included in a block are leaves of a Merkle hash tree. The root of the tree v_i is a commitment to the entire block. Existence of the leaf L in the block can be demonstrated by providing a set of $\log_2 N$ VOs $\mathbf{u}$ that satisfy $f_{bt}(h(L), \mathbf{u} = v_i$.

In addition, the tree structure also permits efficient incremental updates to the leaves, which is a feature that is not taken advantage of in most blockchain networks. Specifically, given that $r = f_{bt}(h(L), \mathbf{u}$, proving existence of leaf L, two kinds of incremental updates are possible:

(1) leaf update: corresponding to an update of the verified leaf L to L', the new root is $r' = f_{bt}(h(L), \mathbf{u}$; and

(2) insertion/deletion of leaves: if a new leaf L_n is inserted to the right of existing leaf L, the new root is $r' = f_{bt}(h(h(L), h(L_n))), \mathbf{u}$; if root update $r \rightarrow r'$ can be demonstrated to be consistent with inserting a leaf, then an update $r' \rightarrow r$ is for deleting a leaf.

Thus, a Merkle hash tree permits efficient computation of a commitment (root) to a dynamic set of leaves with practically unrestricted cardinality. For a tree with a billion leaves, 30 hash operations using 30 VOs will be required to verify the existence of a leaf. An additional

30 hash operations (using the same VOs) will be required to update the leaf, or insert a new leaf, or delete a leaf.

2.2 Ordered Merkle tree

In an OMT (Ramkumar, 2014), proof of existence of a leaf can simultaneously convey existence of key-value pairs, nonexistence of key-value pairs and possibly highest/lowest keys.

Each leaf in an OMT is a three-tuple of the form $\{i, i_n, v_i\}$. Together, all leaves form collection of key-value pairs with unique keys. In a leaf $\{i, i_n, v\}$, i is the unique key in the collection, i_n is the next-key and v_i is the value of the item with key i. For a lone item in the collection with $i_n \leqslant i$, i and i_n are respectively the highest and lowest keys ($i = i_n$ for a collection with a single item).

An item with key j can be inserted only if no item with key j currently exists. Nonexistence of key j can be demonstrated by demonstrating existence of a leaf $\{i, i_n, v_i\}$ such that:

$$j \in [i, i_n[\Rightarrow \begin{cases} i < j < i_n & \text{if } i_n > i \\ j < i_n \leqslant i \text{ or } i_n \leqslant i < j & \text{if } i_n \leqslant i \end{cases}. \tag{6}$$

2.3 Hash accumulator

A hash accumulator (Bayer *et al.*, 1993; Ramkumar, 2014) is a dynamic commitment to a growing list of values $v_1 \ldots v_n$, and is computed as follows:

$$\alpha_1 = v_1, \alpha_2 = h(\alpha_1 \| v_2), \ldots, \alpha_n = h(\alpha_{n-1} \| v_n) \ldots. \tag{7}$$

Specifically, the accumulated hash α_i is a commitment to all values $v_1 \ldots v_i$ accumulated thus far, and the chronological order in which they were accumulated. From the properties of $h()$, it is impractical to determine any sequence of values different from $v_1 \ldots v_i$, for which the accumulated hash is α_i.

In most blockchains, values like $v_1, v_2, \ldots$ are commitments (Merkle tree roots) of blocks. The accumulated hash α is a commitment to the entire ledger.

Given the value of the current accumulated hash α, all values $v_1 \ldots v_n$ are necessary to determine if a specific v_j exists in the list.

2.4 Hash calendar

A hash calendar (Buldas and Saarepera, 2014) can be used to compute a dynamic commitment to growing list of values $v_1 \ldots v_n \ldots$, if new values are added at a constant rate (e.g. v_i is added at time $i\Delta + \tau$ where τ is the calendar start-time, and Δ is a fixed interval).

The hash calendar is implemented as a Merkle hash tree where leaves $v_1 \ldots v_n \ldots$ are added from left to right. In general, when n is not a power of 2, the tree may be seen consisting of up to $\log_2 n$ complete subtrees. For example, after 18 intervals ($n = 18$), the tree can be seen as consisting of two complete subtrees – a tree with 16 leaves, and a tree with 2 leaves. Note that the number of ones in the binary representation of n is the same as the number of complete subtrees. If $n = 22 = 10{,}110_b$ the tree will have three complete subtrees with 16, 4 and 2 leaves, respectively.

The dynamic commitment γ to a calendar is the accumulated hash of the roots of all (maximum of $\log_2 n$) complete subtrees.

2.5 TESLA

In the TESLA (Perrig *et al.*, 2000) broadcast authentication protocol, the sender A of a broadcast stream chooses a random value (say) K_0^A, and creates a hash chain $\{K_0^A \cdots K_L^A\}$, where:

$$K_1^A = h\left(K_0^A\right), K_2^A = h\left(K_1^A\right), \ldots, K_L^A = h\left(K_{L-1}^A\right).$$

From the properties of $h(\,)$, given K_i^A it is trivial to compute $K_{j \geqslant i}^A$ (by repeated hashing $j-i$ times), but impractical to compute $K_{j<i}^A$.

The tail value of the chain K_L^A is the "public key" of A. It is associated with two additional values: an absolute value of time T and a time interval Δ. The interpretation of this association is that "K_{L-i}^A will remain A's secret until time $T + i\Delta$."

A hashed message authentication code (HMAC) for a message M and secret K, namely, $\mu = \text{HMAC}(M, k)$, is typically a token accompanying a message M. The receiver with knowledge of K on verifying that HMAC(M, k) is the same as the token accompanying the message can safely conclude that the token μ could have been computed only by an entity with the knowledge of the key K.

Consider a scenario where an HMAC $\sigma_M = HMAC(M, K_{L-i}^A)$ for a message M using value K_{L-i}^A from the hash chain was seen before time $t = T + i\Delta$. Later, after time $T + i\Delta$, the value K_{L-i}^A from the chain is disclosed, satisfying $\sigma_M = HMAC(M, K_{L-i}^A)$. This is proof that σ_M could have been computed only with the knowledge of K_{L-i}^A (and only the creator of the chain A could have had knowledge of K_{L-i}^A before time $t = T + i\Delta$).

In other words, as long as it is possible for everyone to establish that σ_M was "seen" before time t, everyone can be convinced that the message was sent by A. In such a scenario, σ_M is a digital signature for the message M by A. As we shall see later, one possible approach to ensure that "σ_M was seen before time t" is by employing a timestamping service (TSS) to timestamp the value σ_M. Thus, in conjunction with a TSS, TESLA broadcast authentication becomes a non-reputable digital signature scheme.

In theory, digital signatures based on asymmetric cryptographic primitives do not need to rely on an additional infrastructure for timestamping. In practice, timestamps for signatures are required in any case in scenarios where we need to cater for revocation of keys. Specifically, timestamping is required to ensure that a signature was computed before the key was revoked. Another advantage of asymmetric cryptography-based digital signatures is that they are instantly verifiable. Note that TESLA signatures have to be computed when the key used for computing the HMAC was a secret that can be verified only after the key used for HMAC is made public (along with the signed message/transaction).

This disadvantage of TESLA is perhaps more than offset by its advantages. First, clogging attacks can be effectively addressed. Second, in several evolving blockchain-based application scenarios, transactions may be measurements from sensors or severely resource-limited Internet of Things devices (Xu *et al.*, 2016) that may not be capable of performing asymmetric computations. Third, in the emerging post-quantum computing (Chen *et al.*, 2016) world, asymmetric cryptography may no longer be a viable option anyway.

3. Blockchain processes as an FSM

The FSM model of process $\mathcal{P}$ with dynamic process states $\mathcal{S}$ can be represented as $\delta: \mathbf{I} \times \mathcal{S} \mapsto \mathcal{S}$ where δ represents a next-state function triggered by unconstrained input $\mathbf{I}$. In practice, any process $\mathcal{P}$ can be defined as using a set of (say) m next-state functions $f_1(\,) \ldots f_m(\,)$.

In a blockchain, inputs that trigger execution of next-state functions are broadcast transactions of m different types. Specifically, execution of a next-state function $f_j(\,)$ is triggered by a transaction T_i^j of type j, broadcast at some time t_i^j. Function $f_j(T_i^j)$ is executed only if the transaction is well-formed, and results in a change in the process state $\mathcal{S}$. The progression of states of process $\mathcal{P}$ due to a sequence of transactions $T_1^{j_1}, \ldots, T_n^{j_n}$ can be represented as follows:

$$\mathcal{S}_0 \xrightarrow{f_{j_1}\left(T_1^{j_1}\right)} \mathcal{S}_1 \xrightarrow{f_{j_2}\left(T_2^{j_2}\right)} \mathcal{S}_2 \cdots \mathcal{S}_{n-1} \xrightarrow{f_{j_n}\left(T_n^{j_n}\right)} \mathcal{S}_n, \ldots j_i \in \{1 \ldots m\}. \tag{8}$$

In Equation (8), $\mathcal{S}_0$ is the initial state of the process, and $T_i^{j_i}$ is the ith "well-formed" transaction of type $j_i \in \{1 \ \ldots \ m\}$. Given the initial state $\mathcal{S}_0$, and descriptions of functions $f_1(\,) \ldots f_m(\,)$, the state $\mathcal{S}_n$ is completely determined by the sequence of transactions $T_1, T_2, \ldots, T_n$. Consequently, for purposes of auditing the correctness of process states, it is sufficient for the ledger entries $e_1, e_2, \ldots$ to be a list of transactions in their chronological order, or more specifically:

$$e_1 = (t_1, T_1), e_2 = (t_2, T_2), \ldots e_i = (t_i, T_i), \ldots. \tag{9}$$

As an example, if the process executed by a blockchain represents a bank, the types of transactions may be `OpenAccount()`, `CloseAccount()`, `Transfer()`, etc. A transaction `Transfer()` to transfer an amount x from an account A to an account B will be deemed well-formed only if it is authorized (using a digital signature) by A, and sufficient balance exists in account A.

A TSS (Bayer *et al.*, 1993) attributes a "seen-at-time" t to a value v to be timestamped. If the blockchain is used to implement a TSS, where the state of the process is merely receipt of timestamp requests, no further processing is necessary. A blockchain TSS merely needs to maintain a ledger with entries of the form:

$$e_1 = (t_1, v_1), e_2 = (t_2, v_2), \ldots e_i = (t_i, v_i), \ldots, \tag{10}$$

where an entry (t_i, v_i) states that v_i was seen-at-time t_i (or more specifically, v_i was submitted for timestamping at time t_i).

In general, blockchain transactions like $T_1 \ \ldots \ T_i$ need to be signed because we need to know the source of the broadcast. Values like $v_1 \ \ldots \ v_i$ submitted for timestamping need not be digitally signed, as a TSS says nothing about who sent v_i – it just says that v_i existed at time t_i. Most often, a timestamp is for a document. The creator of a document D can compute a hash $v = h(D)$ and submit it to the TSS for timestamping. Later, existence of a timestamp (t, v) is proof that a document D satisfying $v = h(D)$ existed before time t (as it is impractical for anybody to create the document D after the timestamp for v was obtained). This mechanism is useful for resolving copyright issues.

3.1 Checkpoints

As we saw in the previous section, it is sufficient for ledger entries to be a sequence as transactions as in Equation (9). With this information, while anyone can determine the state of the system $\mathcal{S}_n$ following n transactions, the overhead may be prohibitive for regular users, especially for large-scale systems with billions of process states.

If it is possible to introduce "checkpoints" corresponding to process states like $\mathcal{S}_i$, $\mathcal{S}_{i+1}$ before and/or after each transaction, then verification of correctness of any specific transaction will involve verification of correctness of the single state change:

$$\mathcal{S}_i \xrightarrow{f_{j_i}\left(T_i^{j_i}\right)} \mathcal{S}_{i+1}. \tag{11}$$

An OMT is useful for creating such checkpoints. Specifically, if all process states are considered as leaves of an OMT, then two values s_i and s_{i+1} can be seen as commitment to process states before and after the transaction. More specifically, s_i is the root of an OMT whose leaves represent process state $\mathcal{S}_i$; s_{i+1} is the root of an incrementally updated OMT, whose leaves represent process states $\mathcal{S}_{i+1}$.

In the proposed approach, process states, and all nodes of the OMT with process states as leaves, need to be maintained only by incentivised users. Ledger entries corresponding to every transaction include OMT roots like s_i and s_{i+1}.

As long as regular users have reliable access to any ledger entry, and if it is possible for users to determine which specific process states were implicated (read/updated or inserted/deleted) in the transaction, they can check the correctness of the state change, by obtaining $\mathbb{O}(\log_2 N)$ VOs from incentivised users.

3.2 An example

As an example, consider a ledger entry:

$$[t, \mathbf{T}, s, s'] \tag{12}$$

where $\mathbf{T}$ is a transaction that occurred at time t. According to the ledger entry, transaction $\mathbf{T}$ required the system state (OMT root) to be updated from s to s'. Assume that the purpose of this transaction $\mathbf{T} = [B, x]_A$ was to transfer an amount x from an account A to account B. The process states (leaves of a tree) implicated in this transaction are:

(1) a leaf $L_A = \{A, A_n, a\}$ showing current balance a in account A, and A_n as the next account number; and

(2) a leaf $L_B = \{B, B_n, b\}$ showing current balance b in account B.

A state-change function is $f(\ )$, where $s \xrightarrow{f(\mathbf{T})} s'$ checks if the transaction is well-formed. According to $f(\ )$, transaction $\mathbf{T} = [B, x]_A$ is considered well-formed only if it is signed by A, and if the balance in account A, $a \geqslant x + m + f$, where m is a minimum balance requirement, and f is a transaction fee.

Following the transaction, the two leaves need to be updated to (A, a') and (B, b'), respectively, where $a' = a - x - m - f$ and $b' = b + x$. In other words, using:

(1) values A, B included in the transaction T; and

(2) auxiliary values $\mathbf{I} = \{a, A_n, b, B_n, \text{VOs}\}$ provided by an incentivised user (from leaves and nodes of the OMT maintained by the incentivised user).

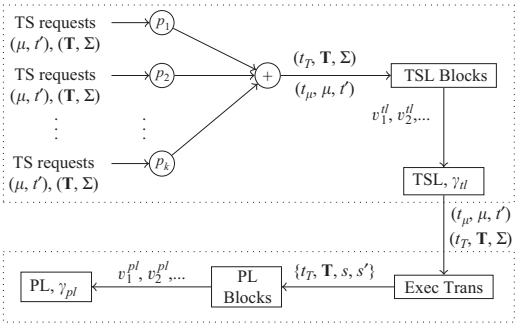
Any regular user can determine that the state change $s \rightarrow s'$ is correct, only if the following predicates are true, namely:

(1) leaves $L_A = \{A, A_n, a\}$ and $L_B = \{B, B_n, b\}$ exist in a tree with root s; and

(2) leaves $L'_A = \{A, A_n, a'\}$ and $L'_B = \{B, B_n, b'\}$ exist in a tree with root s'.

4. A scalable blockchain platform

In the broad outline of a scalable blockchain platform in Figure 2 for executing any number of possibly unrelated large-scale processes, two separate ledgers are maintained – a TSL and a process ledger (PL). Blockchain broadcasts of two types, namely, transactions and TESLA MACs (which are signatures for transactions to be broadcast at a later time) are timestamped and entered in the TSL. In practice, a small number of portals may exist for users to submit broadcast packets for timestamping. Transactions in the TSL are then processed in a chronological order to create PL entries.



Notes: Two types of timestamp requests are timestamped by portal operators $p_1 \ldots p_n$. Timestamp requests are collated and used for creating timestamp ledger (TSL) blocks, where each block corresponds to an interval of length Δ_{tl}. The commitments to TSL blocks v_i^{tl}, v_2^{tl},... are TSL entries made at a constant rate $(1/\Delta_{tl})$ to create a hash calendar with commitment γ_{tl}. Execution of transactions $(t_T, \mathbf{T}, \Sigma)$ in the TSL (signed using a MAC entry $(t, \mu, t' \approx t_T)$, also in the TSL) triggers state changes, resulting entries of the form $\{(t_T, \mathbf{T}, s, s')\}$ added to in uniformly spaced process ledger (PL) blocks (corresponding to intervals of length Δ_{pl}); commitments v_i^{pl}, v_2^{pl},... to PL blocks are PL entries made at a constant rate $(1/\Delta_{pl})$ to create a hash calendar with commitment γ_{pl}

Figure 2.
A scalable blockchain for executing large-scale processes

4.1 Timestamp ledger

Two types of broadcasts that are timestamped include:

(1) $[\mu, t']$: a TESLA signature (MAC) μ along with an expected time t' when a transaction (for which μ is a signature) will be sent; and

(2) $[\mathbf{T}, \sum]$: a signed transaction $\mathbf{T}$ along with signature $\sum$; in the signature $\sum = \{S, K, j, n, t_{\text{mac}}\}$, K is jth value in the nth TESLA chain of sender S; t_{mac} is the time at which the MAC was timestamped earlier.

The operator of each portal batches the requests into regular intervals. The operator timestamps each request, and computes an accumulated hash of all (now timstamped) requests received during the current interval. At the end of the current interval, the operator signs the accumulated hash as proof of acceptance of all requests during the interval. The portal broadcasts the stream of all timestamp requests received during the interval along with the accumulated hash and signature for the interval.

The process for creating the TSL involves collating all timestamp requests from possibly multiple portals, and creating blocks corresponding to uniform intervals of time, say Δ_{tl}. Each entry in a block (a Merkle tree leaf) can be a timestamped TESLA MAC (t_μ, μ, t'), or a timestamped transaction $(t_T, \mathbf{T}, \sum)$.

For creation of the TSL one entity is declared to have the power to resolve ambiguities in consensus. Merkle tree roots $v_1^{tl}, v_2^{tl}, \ldots$ of blocks created in this fashion (at a fixed rate) are used to construct a hash calendar with dynamic commitment γ_{tl}.

4.2 Blockchain processes and PL

Timestamped signed transactions (t_T, **T**, $\sum$) in the TSL are processed by incentivised users to create PL entries. The first step toward this is the verification of signature $\sum = \{S, K, j, n, t_{\text{mac}}\}$, using TSL entry ($t_{\text{mac}}$, μ, $t' \approx t_T$). Incentivised users, who continuously monitor the broadcast channel, may simply cache broadcasts like (t_T, **T**, $\sum$), (t_{mac}, μ, t') that are intended to be added to TSL to avoid the overhead of actually accessing the TSL. The process for signature verification using $\sum$ and μ is outlined later in Section 4.3.

Transactions with good signatures are processed to create PL entries. Every process has a unique identifier Θ. Every process is defined by a set of m state-change functions $\delta_1^{\Theta}(\,) \ldots \tau_m^{\Theta}(\,)$. Broadcast transactions **T** that trigger a state-change function $\delta_i^{\Theta}(\,)$ explicitly identify the process Θ, function index i and the sender (who signed the transaction), in addition to other process specific values.

The unique process identifier Θ is the root of a static hash tree. The leaves of the static tree are static descriptions of state-change functions $\delta_1^{\Theta}(\,) \ldots \tau_m^{\Theta}(\,)$. Each $\delta_i^{\Theta}(\,)$ can be seen as a mapping:

$$\delta_i^{\Theta} : \{t_T, \mathbf{T}, s, \mathbf{I}\} \Rightarrow \{\text{Preconditions, Postconditions, } s'\}, \tag{13}$$

where:

(1) **T** is the triggering transaction with a timestamp t_T;

(2) **I** are auxiliary values that are not included in the transaction, but are available to incentivised users who maintain all OMT leaves and nodes;

(3) the OMT root s is the commitment to the process states before the transaction **T**; and

(4) the OMT root s' is the commitment to the process states after the transaction **T**.

Successful execution of a state-change function results in the creation of an entry in the PL. More specifically, the PL consists of blocks created at regular intervals of time, where each block consists of multiple entries of the form:

$$\{t_T, \mathbf{T}, s, s'\}. \tag{14}$$

All well-formed transactions occurring in a specific interval of time are added to a single block as leaves of a hash tree. The sequence of roots of such trees $v_1, v_2, v_3, \ldots$ (one corresponding to each block) is added to the ledger at regular intervals of time using a hash calendar.

Incentivised users may choose to maintain an additional log of auxiliary values **I** necessary to verify preconditions and postconditions for every transaction.

Given a ledger entry, $\{t_T, \mathbf{T}, s, s'\}$ any user can verify the validity of the signature $\sum$ by fetching the timestamped MAC μ from the TSL. The user can then proceed to verify the correctness of the state change in exactly the same manner as an incentivised user, except that auxiliary values **I** necessary to verify that preconditions and postconditions are demanded from an incentivised user.

4.3 Infrastructure for digital signatures

Timestamped transactions of the form (t_T, **T**, $\sum$) in the TSL are first verified for a consistent signature $\sum = (S, K, j, n, t_{\text{mac}})$. The process for verifying a signature utilizes another TSL entry – a timestamped MAC of the form (t_{mac}, μ, t') where $t' \approx t_T$ was the expected timestamp of the transaction **T**. Conveying the expected time t' is merely to permit incentivised used to cache MAC μ in anticipation. Regular users, however, have the additional overhead for fetching a TSL entry.

The signature $\sum = (S, K, j, n, t_{\text{mac}})$ for $\mathbf{T}$ is deemed legitimate only if the following predicates are true:

(1) the nth chain of user S, associated with a commitment c, start-time T and interval Δ, has not been revoked;

(2) $\mu = h(\mathbf{T}, K)$ and $t_{\text{mac}} < T + j\Delta$, where $(t_{\text{mac}}, \mu, t')$ exists in the TSL; and

(3) the result of hashing K repeatedly, j times, is c.

The infrastructure for maintaining parameters associated with different chains of users is itself a blockchain process/application with a unique identifier, say $\Theta_{\sum}$.

4.3.1 Process $\Theta^{\sum}$ states. Every user is associated with a long-term public identity, which is the same as the commitment to the first hash chain created by the user. The user can later create any number of hash chains with different start times and intervals, and bind commitments to such chains to the user's long-term identity. Three different transaction types for this application are employed by users to create their long-term identity, certify new chains and revoke chains.

The states of the process for maintaining hash chain parameters are represented using leaves that represent key-value pairs of the form:

$$\left(\text{key} = [S \| i], \text{value} = [c \| \Delta \| T \| R \| t_r]\right), \tag{15}$$

where item keys are concatenations of a long-term key S and a chain index i. The value of an item with key $S \| i$ is a concatenation of five values:

(1) a commitment c to chain i (where $c = S$ for $i = 1$);

(2) interval Δ;

(3) start-time T;

(4) a value R, whose preimage needs to be disclosed by the user S to revoke the chain; and

(5) the time t_r at which the chain was revoked (which is set to 0 for an unrevoked chain).

To join the blockchain network:

(1) a user generates her first chain $K_0 \ldots K_l$ of any length, and chooses appropriate values of interval Δ and start-time T; $S = K_l$ will ultimately become the user's long-term identity;

(2) the user chooses a random R_0 and computes $R = h(R_0)$, and stores R_0 and K_0 in a secure location; and

(3) the user computes a MAC $\mu = h(K_n, \Delta, T, R, K_i)$ where K_i is a secret from the hash chain (usually K_{l-1}), and submits the value $[\mu, t']$ for timestamping while K_i is a secret. The value t' is an estimate of time at which the certificate (for which μ is the signature) will be submitted.

Let the timestamp obtained for μ be t_{mac}. Around time t', the user submits a transaction $\mathbf{T}$ (indicating process $\Theta_{\sum}$ and function index $i = 1$ for creating the first chain) along with a signature $\sum_s$, where:

$$\mathbf{T} = [\Theta_{\Sigma}, 1, S, \Delta, T, R], \Sigma_S = (S, K_i, j, 1, t_{\text{mac}}). \tag{16}$$

As no entry exists for user S yet, the verification of the signature for this transaction is considered as a special case. If entry $(t_{\text{mac}}, \mu = h(\mathbf{T}, K_i))$ exists in TSL, and if $t_{\text{mac}} < T + j\Delta$, and if hashing K, j times, yields S, the signature is accepted as valid.

4.3.2 Process Θ_{Σ} transactions. The predicates (preconditions and postconditions) for different Θ_{Σ} transactions are as follows.

Transaction type 1 is for creating the first chain for a new user. Type 2 is for creating the nth chain where $n > 1$ chains. Type 3 is for revoking the nth chain. If the OMT roots of the process Θ_{Σ} are s and s', respectively (before and after a transaction), the preconditions and postconditions for each transaction can be expressed as follows:

$$\begin{array}{lll}
1. & t_T, \mathbf{T} = [\Theta_{\Sigma}, 1, S, \Delta, T, R]_S, \mathbf{I} = \{x, y, v_x\} \\
\text{Pre} & \{x, y, v_x\} \in s, S\|1 \in [x, y[\\
\text{Post} & \{x, v, v_x\} \in s', \{S\|1, y, S\|\Delta\|T\|R\|0\} \in s' \\
2. & t_T, \mathbf{T} = [\Theta_{\Sigma}, 2, c, \Delta, T, R]_S, \mathbf{I} = \{n, v_{n-1}, x\} \\
\text{Pre} & \{S\|n{-}1, x, v_{n-1}\} \in s, S\|n \in [S\|n{-}1, x[\\
\text{Post} & \{S\|n{-}1, S\|n, v_{n-1}\} \in s', \{S\|n, x, c\|\Delta\|T\|R\|0\} \in s' \\
3. & t_T, \mathbf{T} = [\Theta_{\Sigma}, 3, n, R_0]_S, \mathbf{I} = \{c, \Delta, T, R, x\} \\
\text{Pre} & \{S\|n, x, c\|\Delta\|T\|R\|0\} \in s, h(R_0) = R \\
\text{Post} & \{S\|n, x, c\|\Delta\|T\|R\|t_T\} \in s'
\end{array} \tag{17}$$

Timestamped transactions of the form (t_T, **T**) in the TSL with valid signatures (from a sender S) are triggers to state changes. The precondition for a type 1 transaction from a user S is that no entry should exist for key $S\|1$, which is demonstrated by the existence of an item for key x with next-key y, such that $S\|1 \in [x, y]$. The postcondition is that an entry for index $S\|1$ is added with values Δ, T, R included in the transaction.

The precondition for adding the nth chain (transaction type 2) for S is the existence of the $n{-}1$th chain for S, with next index x satisfying $S\|n \in [S\|n{-}1, x]$. The postconditions include modification of the next index x in the existing item, and introduction of a new item for key $S\|n$ with next index as x.

The precondition for revoking the nth chain is that the value R_0 included in the transaction should be a preimage of a value R in the certificate of the unrevoked nth chain for S. The postcondition is that the time of revocation is updated from 0 (for an unrevoked chain) to the transaction time t_T.

4.4 Selective audits by regular users

Let T_b be the start-time of the both blockchain ledgers (TSL and PL). As users know the current time, they are aware of the current number of blocks (say n_t, n_p) in both TSL and PL.

If, for example, $n = 274 = 100{,}010{,}010_b$ (where bits 9, 5 and 2 are one) the user expects a calendar hash γ_n to be accompanied by two values r_5 and r_9, satisfying:

$$\gamma_n = h(r_9, h(r_5, r_2)). \tag{18}$$

If the next time the same user utilizes the ledger is at a time when $n' = 308 = 100{,}110{,}100_b$ the user expects γ'_n along with values r'_3, r'_5, r'_6 and r'_9 where $r'_9 = r_9, r'_5 = r_5$, and $r_5 = r'_5$ and r_3 should be nodes in a tree with root r'_6. VOs for verifying this fact can also be demanded from active users.

Thus, using a calendar hash permits intermittent users to check the validity of the current calendar hash against previously known values. Consequently, passive users store

only minimal state information (past commitments to $\log_2 n$ calendar hashes), and are not required to fetch every single block to verify the validity of the current calendar hash γ_n.

The trust in the correctness of γ_n can be leveraged verifying the commitment v_i to any block in the past, by requesting up to $\log_2 n$ VOs. With trust in the correctness of v_i, the user may demand the entire block, or a specific entry, or the last m entries, etc., along with VOs for existence proofs in a tree with root v_i.

Once users have authentic copy of a specific leaf in a specific PL block, namely, a PL entry $\{t, \mathbf{T}, s, s'\}$, they can proceed to check the correctness of the state change $s \rightarrow s'$ triggered by transaction $\mathbf{T}$ as described in Section 4.2.

5. Conclusions

Blockchain networks offer a universally trusted computing platform, which can be leveraged to replace (or, at the minimum, limit the role of) high-overhead infrastructures like banks, insurance and governments.

Utilizing the full ability of blockchain networks calls for explicit mechanisms to reduce/eliminate ambiguities, to promote universal consensus (to avoid forking); low overhead strategies for selective audits, to ensure that even intermittent users can readily follow the correct fork (in the event a fork occurs); and mechanisms to reduce susceptibility of the broadcast network to clogging attacks.

An architecture for a scalable blockchain platform and cryptographic protocols underlying the architecture were proposed to address three major concerns that affect scalability of blockchains.

References

Bayer, D., Haber, S. and Stornetta, W.S. (1993), "Improving the efficiency and reliability of digital time-stamping sequences", *Sequences II: Methods in Communication, Security and Computer Science*, Vol. 2, Springer-Verlag, pp. 329-334.

Bentov, I., Charles, L., Mizrahi, A. and Rosenfeld, M. (2014), "Proof of activity: extending Bitcoin's proof of work via proof of stake", *ACM SIGMETRICS Performance Evaluation Review*, Vol. 42 No. 3, pp. 34-37.

Bozic, N., Pujolle, G. and Secci, S. (2016), "A tutorial on blockchain and applications to secure network control-planes", Smart Cloud Networks & Systems (SCNS). IEEE, Dubai.

Buldas, A. and Saarepera, M. (2014), "Document verification with distributed calendar infrastructure", US Patent No. 8,719,576, May 6.

Chen, L., Jordan, S., Moody, D., Liu, Y.-K., Peralta, R., Perlner, R. and Smith-Tone, D. (2016), "Report on post-quantum cryptography", Internal Report No. 8105, US Department of Commerce, National Institute of Standards and Technology, April.

Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., Song, D. and Wattenhofer, R. (2016), "On scaling decentralized blockchains", *International Conference on Financial Cryptography and Data Security, Springer*, pp. 106-125.

Digiconomist, "Bitcoin energy consumption", available at: https://digiconomist.net/bitcoin-energy-consumption (accessed July 7, 2017).

Digital Trends, "The world's cryptocurrency mining uses more electricity than Iceland", available at: www.digitaltrends.com/computing/bitcoin-ethereum-mining-use-significant-electrical-power/ (accessed July 7, 2017).

Kiayias, A., Russel, A., David, B. and Oliynykov, R. (2017), "Ouroboros: a provably secure proof-of-stake blockchain protocol", *Annual International Cryptology Conference, Springer*, *Cham*.

Merkle, R.C. (1987), "A digital signature based on a conventional encryption function", Advances in Cryptology, CRYPTO '87, Lecture Notes in Computer Science 293, Santa Barbara, CA.

Nakamoto, S. (2008), "Bitcoin: a peer-to-peer electronic cash System", October 31, available at: http://nakamotoinstitute.org/bitcoin/ (accessed October 21, 2018).

Oppliger, R. (1999), "Protecting key exchange and management protocols against resource clogging attacks", in Preneel, B. (Ed.), *Secure Information Networks*, Springer, Boston, MA, pp. 163-175.

Perrig, A., Tygar, J.D., Song, D. and Canetti, R. (2000), "Efficient authentication and signing of multicast streams over lossy channels", *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, *May 14-17*, p. 56.

Ramkumar, M. (2014), *Symmetric Cryptographic Protocols*, Springer, Dubai.

Wood, G. (2014), "Ethereum: a secure decentralised generalised transaction ledger", Ethereum Project Yellow Paper, available at: https://github.com/ethereum/wiki/wiki/White-Paper

Xu, K., Qu, Y. and Yang, K. (2016), "A tutorial on the Internet of Things: from a heterogeneous network integration perspective", *IEEE Network*, Vol. 30 No. 2, pp. 102-108.

Further reading

ECDSA (2013), National Institute of Standards and Technology (NIST), Federal Information Processing Standard (FIPS) 186-4, July, pp. 19-26.

Corresponding author
Mahalingam Ramkumar can be contacted at: ramkumar@cse.msstate.edu