ULedger Technical Approach

Technical White Paper

Taulant Ramabaja, Josh McIver, Pete Anewalt, Dave Otander

November, 2017

ULEDGER

ULedger: Data Integrity through Blockchain

Bringing continuity, consensus, transparency, proof & identity to your data

Table of Contents

Technology Abstract

We provide system designs and methods by which any distributed and dynamically changing set of state machines (such as a blockchain data structure) can prove a relative order of events. The approach does not rely on a single global state such as is the case in Bitcoin, Ethereum, or more classical Byzantine Fault Tolerance (BFT) systems. Rather, we incorporate a combination of multiple time keeping methods at each change of the set of state machines to enable tracking of timing of the change. A reason behind our system design is a need for communication and computation in relativistic spacetime, such as in an inter-planetary context, as well as under inconsistent network conditions, such as in an IoT and Edge Computing context.

Stated differently, our proposed technologies are less concerned about absolute time and instead deal with time as a relative event ordering problem. This approach solves the issue of trusting other systems' internal clocks, network outages/disruptions, malicious clock changes or transaction withholding, and even time keeping under physical relativistic conditions.

Rather than being focused with solving the "double-spend" problem, ULedger's proposed technologies make possible proving misbehavior (including double spending) if any party in the network receives mutually exclusive or contradictory transactions. Our proposed technologies provide a way to efficiently prove:

1. The integrity of the data
2. The relative order in which the piece of data was received by all parties
3. The integrity and order of all data which is used as input to the current data.

Blockchain Generally

A blockchain is a data structure implemented as a continuously growing set of blocks, which can store data and are linked and secured using cryptography. Typically, each block in a blockchain includes a hash pointer as a link to a previous block and transaction data (or a payload). By design, blockchains are inherently resistant to modification of the data.

Further, blockchains are both a data structure and a time keeping mechanism for that data structure. This is achieved by combining merkelization of the to-be-verified data in a Merkle Direct Acyclic Graph, such as a Merkle Tree, and adding Proof of Work(PoW) to it. For the purposes of this paper, Blockchain technology includes PoW, meaning Blockchain proposals that do not have some sort of PoW are excluded.

Existing blockchains such as Bitcoin or Ethereum create a globally and transparently shared database of unidirectionally related transactions and blocks. The directionality is determined by the one-way hashing of the blocks as well as the related PoW. Anyone with the complete data of the blockchain can verify that a certain piece of data is included in the data of the blockchain.

Time keeping is a side effect of the need to measure network-wide PoW difficulty adjustments and keep network-wide difficulty always relatively the same. To timestamp any piece of data it needs to be included in the form of a transaction in a block. Because of the shared nature of all data the total amount of absolute timestamps which can be made this way is limited. Furthermore, because which transactions are included in what order in what block is at the full discretion of the block creators (i.e., miners) data cannot be granularly timestamped.

Brief Overview of ULedger

ULedger does not require storing other nodes' data to compute other nodes' transactions. Instead ULedger is a protocol which defines how proofs are exchanged and integrated into one's own datastructure. By doing so every node can prove to others when they **perceived/received** certain data, including other nodes' proofs.

ULedger's blockchain technology utilizes multiple time-keeping methods as each block is added to a blockchain data structure to maintain a relativistic space-time, without maintaining a global state at each blockchain node. The multiple time-keeping methods can include: 1) Vector clocks, 2) Assymetric Key Network Time Protocol, and 3) a proprietary cross-merkelization technology.

Vector clocks are a form of Logic Clocks. They are an algorithm by which transactions/events can be ordered in a distributed manner without synchronicity or real time assumptions. ULedger utilizes Vector Clock mechanisms for ordering Transactions/Events which are input-output dependent across nodes.

Existing Network Time Protocols (NTP) make use of symmetric keys for exchanging data and synchronizing clocks. This is due to historical and technical reasons. Newer Asymmetric Key based NTPs such as Google's Roughtime make use of Asymmetric Key Cryptography for verifying the content and source of data such as a timestamp. By hashing and chaining replies from individual nodes it becomes possible to compare two or more transactions/events from two or more different nodes to compare clocks roughly. An analogy would be a group of friends comparing clocks before splitting up to go somewhere with plans to meet back together at an appointed time. In this system you don't care as much about the real time as much as you care about being roughly synchronized with your peers. Furthermore, it becomes possible to prove to third parties that two or more timestamped transactions disagree on the actual time with one another.

Cross-Merkelization is the process by which multiple chains can mutually Merge Mine (aka Auxiliary Proof of Work - AuxPoW) one another, effectively increasing the security and immutability of both chains. Existing blockchains use only a single result of proof of work. For example Bitcoin uses only a single previous block hash on top of which nodes immediately start mining. If the network was split and multiple such hashes were mined, the nodes continue to mine on the longest chain, which has the most proof of work and still fulfills the Bitcoin Client validity rules. This means that the PoW that went into orphaned blocks is simply discarded, making the network overall less secure and immutable. This also represents a limitation as to how fast block creation can be set.

ULedger includes a form of AuxPoW with Multiple Parent hashes. Instead of only referencing a single previous parent block hash ULedger blocks reference multiple parent hashes. Due to the nature of one-way hashing it becomes possible for each chain owner to know the relative order of blocks with which its chain interacted.

ULedger also makes use of a modified GHOST Protocol. While the regular hash calendar blocks are created at arbitrarily set intervals by the system administrator, interactions with other chains happen on a per-transaction level. Effectively outputs from one chain can be used as inputs for another chain. For doing this cross-referencing both chains sign off on each other's interrelated transactions via the ULedger Event-Chain protocol and merge-mine the subsequent blocks were these transactions were included.

Detailed Description of Technology

A. Blockchain Generally

Based on our narrow definition of Blockchain technology which requires a Merkelized Direct Acyclic Graph datastructure with some sort of Proof of Work at functionally predictable intervals, Bitcoin is the first blockchain ever created.

Bitcoin's main requirements were censorship resistance and pseudonymous identities for transactions and state creation because previous attempts at digital cash were halted due to legal reasons rather than technical ones. Therefore, the primary technical approach to censorship resistance was to create a *decentralized* state machine that keeps track of state changes in an independently verifiable manner, and that choses the next leader in an unpredictable but verifiable manner to append state to the overall state. Previous analysis is referring to this as a Dynamic Membership Multi-Party Signature (DMMS)[1].

To achieve this technically, all state must be shared and stored with all parties globally. Bitcoin also solved the governance and identity problem in anonymous systems by adding a game-theoretic incentive system for parties to participate in block creation. Bitcoin's decentralized/distributed nature can therefore be regarded as a political strategy more so than a pure technical solution. While Bitcoin's approach is maybe close to ideal for its very specific requirements, newer blockchain systems are copying Bitcoin's technical concepts and architecture without thinking about their fundamental difference in requirements.

Blockchain technologies rely heavily cryptographic hashes which are uni-directional in nature, as well as on Merkle Trees and more recently more generalizable Merkelized Direct Acyclic Graphs (MDAGs)[2]. Cryptographic hashes of data are used both for integrity checks of the data as well as Universally Unique Identifiers (UUIDs). Merkle Trees are hierarchically layered and unidirectionally referencing hashes of data. Merkle Trees can be used for rapid and secure verification of large files.

[1] https://blockstream.com/sidechains.pdf

[2] https://github.com/jbenet/random-ideas/issues/20

Immutability of files committed to a MDAG is ensured by requiring the root (single combined hash of all files) to be combined with an unpredictable nonce until the combination of both generates a hash with sufficient nonces (zeros) in front of it, as determined by the software [3]. These "cryptographic puzzles" can only be solved by brute force and can therefore be roughly predicted how much computation went into generating those hashes.

By further referencing in the next block the hash of the previous block in a chain of blocks, each block includes a certain amount of Proof of Work that has been generated. If any single piece of data referenced in any of the blocks were to be modified, all subsequent blocks' hashes and PoW would not match anymore. In order to successfully alter an individual block, an attacker would need to not just run the PoW of the individual block but also of all subsequent blocks.

This, combined with the data replication/decentralization across almost any number of nodes makes the state of the chain highly redundant and secure from attacks.

The foregoing advantages and benefits are not the only considerations, however. There are drawbacks to a blockchain architecture, particularly when the architecture is used outside of its intended use, namely avoiding the double-spend problem. We will analyze the drawbacks of Bitcoin's architecture in the light of industrial and contractual circumstances.

One example of a drawback of blockchain is Bitcoin has no way of dealing with and interpreting data from outside it's blockchain. All state being tracked and modified in Bitcoin originates in Bitcoin in the form of its Coinbase. All transactions solely reference previous transactions back to when the coins and the chain was created originally. Bitcoin can store small amounts of external data in each transaction, but it is barely enough for some hashes and short messages. It has also no way of interacting with this stored data in any programmatic fashion.

Bitcoin is therefore a single-purpose limited decentralized/distributed state machine.

FIG. 1 below provides a simplified representation of the progression of the Bitcoin Blockchain through time and in the network. The rectangles represent individual blocks while the connections represent one-way hashing relationships (using the previous block as a parent and building on top of it).

[3] https://bitcoin.org/bitcoin.pdf

"Orphaned Blocks" are blocks that were mined and continued by part of the network before that part of the network discovered that a longer chain already exists. This can happen due to network splits, blocks being created faster than they can propagate to every miner, client software forking, or other issues.

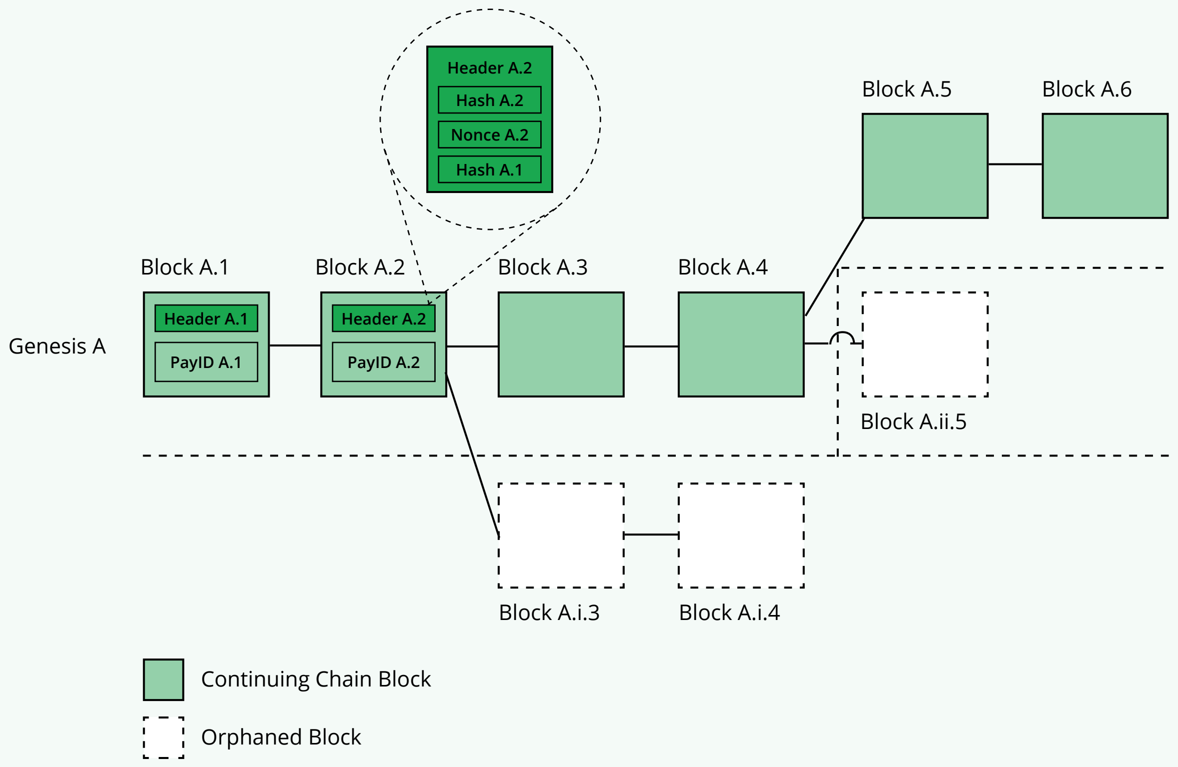

FIGURE 1

One can see that the total PoW security is also a function of how many orphaned forks happen. The more orphaned blocks, the less PoW the main-chain has, and the easier it becomes for an attacker to overwhelm the network overall. These and other dynamics have been analyzed extensively in several papers already[4][5][6][7]. The end-result is that Bitcoin and any similar architecture cannot scale indefinitely. This is primarily due to its *stateful nature* in which the correctness/legality of a transaction is dependent of the correctness of all the previous state of Bitcoin. Scalable protocols always are mostly *stateless* in nature.

[4] https://eprint.iacr.org/2013/881.pdf
[5] https://btc-hijack.ethz.ch/files/btc_hijack.pdf
[6] https://www.cs.cornell.edu/~ie53/publications/btcProcFC.pdf
[7] http://ieeexplore.ieee.org/document/7728010/

B. Uledger technology - Cross-merkelization

We have recognized that the above-mentioned technologies and approaches used in Bitcoin and blockchain generally can be of value in other applications, industries, and contexts as well, with appropriate architecture adaptations for different or new requirements.

We have recognized that what is needed is a way by which any piece of data / state / event / transaction (used interchangeably herein) is:

1. Arbitrary: data can be used and associated with a real-world entity
2. Independently verifiable data
3. Has an associated independently verifiable rough time
4. Its relative order as compared to other events is independently verifiable
5. Misbehaviors of entities can be independently verified when conflicting events are caught.

We propose a protocol for verifying in a fully distributed way the time integrity of hash calendars and similar linked timestamp data structures. Today, while it is possible to prove the integrity of a datagram by using hashing, and even proving its integrity in time and relative order as compared to other datagrams by using linked timestamps such as hash calendars, there is currently no good way of verifying that the hash calendar presented to a third party is the only one generated. It is currently relatively trivial to generate an alternative hash calendar to "prove" any arbitrary order of events. We propose a system by which it is possible for any state machine interacting with the so proven data to forward this proof to state machines downstream without having to do any sort of interactive proof with the originator of the data. Non-interactivity is extremely important in relativistic spacetime and distances which are measured in terms of the speed of light rather than human scale, as well as under inconsistent network connectivity.

Current solutions to this problem in a planetary context include but is not limited to:

- Centralized 3rd party trust (a TTP) (e.g. Guardtime)
- Nakamoto Consensus such as the Bitcoin Network
- Deferred/Anchored Nakamoto Consensus such as OpenTimestamps, Factom, and Tieron, etc.

Our solution uses none of the above methods. ULedger is a truly P2P protocol without the need for consensus on a common global state, hence removing several bottlenecks of Nakamoto Consensus and removing the need for trust as is the case with Trusted Third Parties (TTPs).

Trusted Third Parties (TTPs) for Timestamping (such as Hash Calendars as implemented in Guardtime) do not work. TTPs are censorable. Furthermore, alternative orders of events can be generated and shown to different parties or even the same party in different interactions. This is like a company keeping two or more different accounting books which it selectively shows to different parties.

The above problem of multiple books is generally called the “double spend” problem. Bitcoin solved this by keeping all state shared between all nodes, and having that state verifiable both as standalone state (the input-output chains), as well as in time (timestamps, Merkle Tree, and PoW). This approach is good for Dynamic Membership Multiparty Signatures (DMMS) which are shared state machines, but does not scale to verifying an arbitrary amount of data being generated by centralized state machines interacting with one another. Most state machines in the real world are centralized and/or non-dynamic.

Current semi-blockchain approaches include using SSL for generating proofs of APIs and then uploading/anchoring those proofs to a globally shared ledger such as Ethereum. This approach has multiple issues such as not being scalable because all state needs to be stored and shared by everyone, ordering is not precise as it depends on when the stat hits a random miner/node, and depends on the shared ledger subjective status (e.g. forked or not forked), proofs are dependent on the existing SSL system, etc.

To accomplish verifying in a fully distributed way the time integrity of hash calendars and similar linked timestamp data structures, ULedger utilizes multiple time-keeping methods as each block is added to a blockchain data structure to maintain a relativistic space-time, without maintaining a global state at each blockchain node. The multiple time-keeping methods can include proprietary modified forms of: 1) Vector clocks, 2) Assymetric Key Network Time Protocol, and 3) a cross-merkelization technology.

i. Relative Order - Vector Clocks

Vector clocks are an algorithm through which causality violations within a timeline of events can be detected. To do so the algorithm creates partial ordering of events across interacting nodes.

Independently provable order of events is indispensable for distributed state machines. If all nodes are known entities and security is not a consideration (such as node takeover), Vector Clocks can function by simply adding an integer based counting system. This way a higher integer event is assumed to have happened after a lower integer event.

In ULedger a modified version of Vector Clocks is used for precisely ordering directly interdependent events across nodes. Our modified Roughtime protocol then ensures rough ordering of interrelated event timelines relative to one another across mutually known nodes. For nodes which do not know nor trust one another the Cross-Merkelization protocol then fascilitates rough ordering of interrelated event timelines across mutually fully or partially unknown nodes.

While internal cross-merkelized calendars are used for keeping track of events internally as well as externally, Event-chains are used for keeping track of the absolute order of events within an event-chain.

Event-Chains are Vector Clocks presented in the form of a Merkle DAG. In ULedger each latest Event in the even-chain also includes Merkle Audit Proofs to the Merkle DAG of the State Machine which generated it. This way if the event-chain forks due to network issues, all participating state machines can prove based on the Merkle Audit Proofs who saw what at what time.

If one or multiple state machines somehow collude and generate a secret parallel fork of the event chain, then those two parallel event-chains can be packaged together as a proof of misbehavior and forwarded to the network. This assumes that at some point in the future at least one state machine, be that a participant of the original event-chain or a third party down the line sees two conflicting states, such proof of misbehavior will hit the network eventually.

Publish-Subscribe

ULedger is based on a P2P pub-sub network. Individual State Machines subscribe via secured UDP channels (in our reference implementation the RAET protocol) to one another. No relaying of information happens, only direct connections are used. Future implementations might use a CJDNS-style routing on top of RAET. Ordering of events is subjective to each state machine. State itself contains the signed order and timestamp of the publishing state machines.

Two types of state are being published:

1. Block Headers (arbitrarily at every second generated or when new internal events are required to be provable in the future) are broadcasted to all subscribing nodes.
2. Event-chain related state with individual nodes whose identities are listed as one of the input states.

ii. Asymmetric Key Network Time Protocol

Asymmetric Key Network Time Protocols make use of Asymmetric Key Cryptography for verifying the content and source of data such as a timestamp. By hashing and chaining replies from individual nodes it becomes possible to compare two or more transactions/events from two or more different nodes to roughly compare clocks roughly. The protocol is less concerned about real time, and instead can functions based on being roughly synchronized with peers.

Asymmetric Key Network Time Protocols are useful in scenarios where no single third party server can be trusted with time keeping and/or correct time synchronization, and time does not need to be absolutely precise but can have a margin of error in the second range. Furthermore, they are useful in scenarios where telling the time is in and of itself of high value to such an extent that one wants the telling of the time to be independently verifiable to third parties. This entails mostly low-trust situations as well as extremely high value and time/order sensitive data exchange.

FIG. 2 below is a simplified graphic explaining the already existing reference implementation of Google's Roughtime [8], which is an Asymmetric Key Network Time Protocol.

In this example, a client (e.g. a web browser or mobile application) wants to have a second cryptographically verifiable timestamp to either verify roughly current internal system time, or timestamp a high value data package to make it verifiable to third parties. The high value data package is the Payload. Nonces are random bit numbers. A timestamp is a standard UTC UNIX Timestamp. A Signature is an Asymmetric Key based signature of the above listed data fields.

[8] https://roughtime.googlesource.com/roughtime

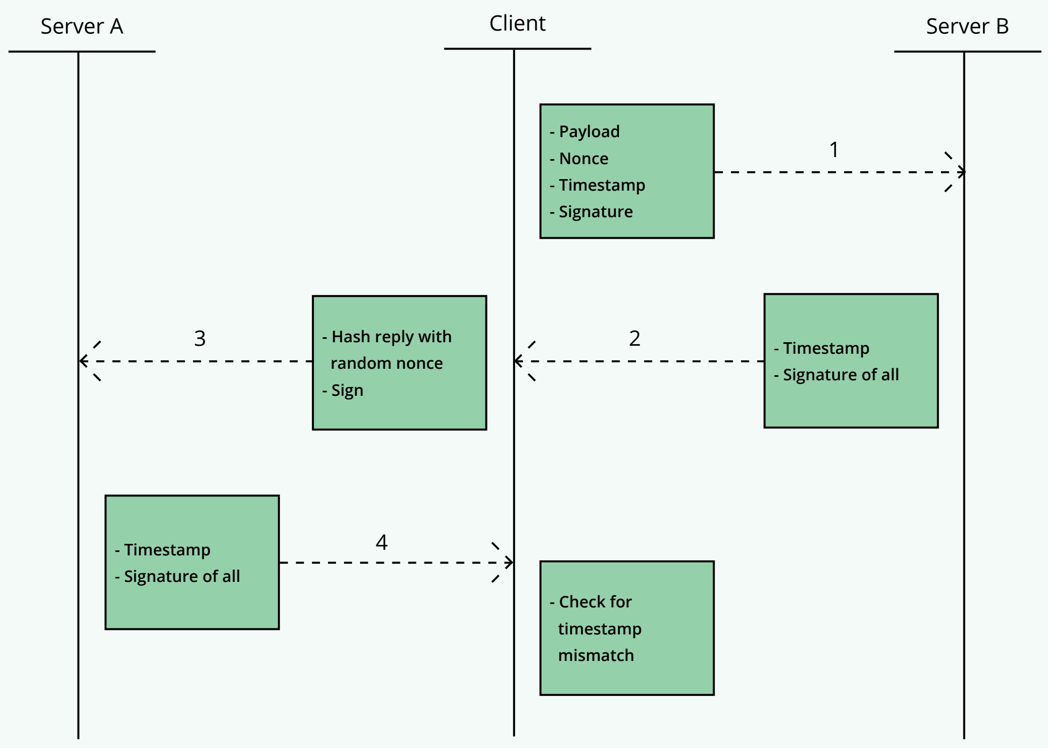

FIGURE 2

The Client sends an optional payload with a freshly generated random nonce and optional timestamp as well as optional signature to Roughtime server B for initial timestamping. Server B then adds to the response the entire request and adds a timestamp and signs the entire response with its Private Key. The signed response is send back to the client, where the client opts to cross-verify this "vouched" time with Server A by first hashing the entire Server B response with a new random nonce to prove freshness, and optionally signs the second request for Server A. Server A does not know the content nor who Timestamped and signed the message before due to the additional hashing performed by the Client. Server A performs the same step as Server B before, by Timestamping and signing the entire request and responding back to the Client. The client then compares the two (or if send to more servers) or more responses. If the Timestamps are within a pre-programmed acceptable range the client accepts them. If the timestamps are off, then the Client can Prove to any third party that the timestamps were off.

FIG. 3 below shows Modified Asymmetric Key Network Time Protocol.

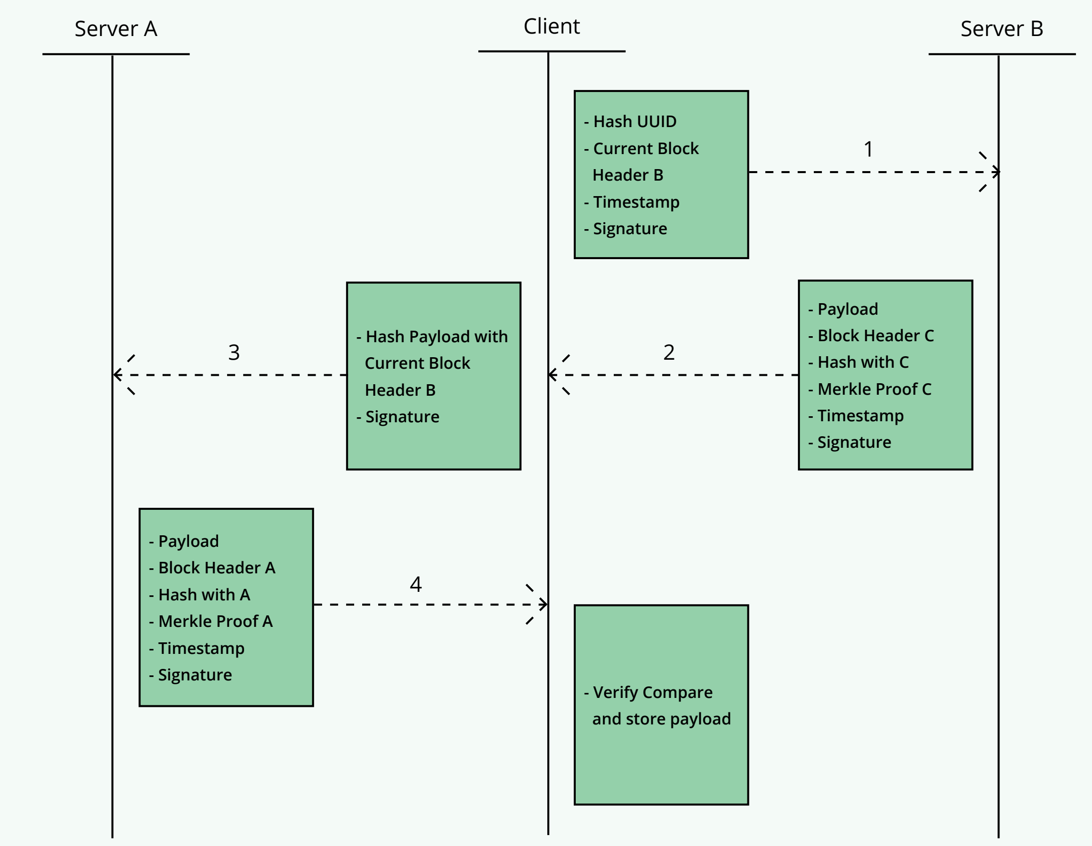

FIGURE 3

The main modifications to the Roughtime protocol are:

- The client is assumed to be a server as well, but can also be a Web/Mobile client, in which case the default timestamp interval difference has to be longer.
- All requests and responses have to be signed.
- The initial request's payload is always a hash of the message/content to be verified and timestamped.
- All Servers add nonces, whether for requests or responses.
- All randomly generated nonces are replaced by the nonces used for PoW at the current block interval of each respective Server's blockchain.
- Requests are included in the MDAG of each Server for later verification.
- Each server includes an optional Merkle proof of MDAG inclusion in its response.

With the above modifications, the freshness of each request and response (or events or transactions) can be pinpointed to a specific time at each chain. With the inclusion of each request and response in each Server's respective MDAG an internal unchangeable timeline of events is preserved for later cross-referencing.

iii. Cross-Merkelization

Cross-Merkelization refers to a proprietary ULedger method by which every State Machine keeps track of its own internal events, interactions with other state machines via the respective Event-Chains, as well as the passage of time via the inclusion of GHOST AuxPoW.

Each State machine generates a centralized Merkle DAG and orders and links events by itself. The order of perceived events is subjective, meaning every state machine sees a different order of events depending on when the state reaches it. This is important as different state machines will have different times at which they receive certain state due to network access and limitations of the speed of light. However, events which are in an event-chain can still be proven because they are individually chained. By including the latest hash of an entire event chain, the particular node essentially preserves the observation that at that point in time that particular chain of events reached him.

Whenever a new block header is generated for the Merkle DAG by the state machine, that block is broadcasted to as many other state machines as possible. Those State machines in turn include these received blocks in their Merkle DAG, ordering them according to when they received them. Upon including any number of foreign blocks in their Merkle DAG and generating their own new block, they in turn broadcast their block to everybody. Individually each state machine also sends Merkle Audit Proofs of the foreign block header's inclusion in their broadcasted block header, when they broadcast their block.

The original broadcaster of the block then includes and merkelizes both the latest block of the replying broadcaster as well as the Merkle Audit Proof of their original block being included in the neighbor's block.

Comparison between Bitcoin Block header and a ULedger Block Header:

Bitcoin Block header content:

Version	Version of the Bitcoin client software used
hashPrevBlock	Hash of the previous block header
hashMerkleRoot	Cumulative hash of all transactions in the block
Time	UTC Timestamp
Bits	PoW difficulty target in compact format
Nonce	32-bit number

ULedger Block header content:

Version	Hash of reference implementation
HashFunction	Hash function used (upgradable)
hashPrevBlock1	Highest PoW of previous block
HashPrevBlockN	Ordered list of PoW previous blocks
hashMerkleRoot	Cumulative hash of all transactions in the block
Time	UTC Timestamp
Bits	Minimum PoW difficulty target in compact form
Nonce	32-bit number

Block Headers

Block Headers are state broadcasted within the ULedger network by all nodes to as many nodes as possible. All other exchanged state is selective. Block headers include in the following hierarchy and order:

Proof of Work Generally

Proof of Work (PoW) in the context of blockchain tech generally is used to choose in an independently verifiable manner the next block creator/leader of the distributed state machine. Bitcoin was the original implementation of this design approach. PoW replaces therefore the governance layer of the distributed state machine with an economic incentive layer. This approach also makes it possible for mutually unknown or even hostile nodes to work together in the distributed state machine. Another beneficial side effect is that the so generated series of blocks become exponentially harder to “re-do” after the fact, as an attacker would need to generate a longer chain than the rest of the network combined.

The absolute minimum for a PoW blockchain is a header design with the following data fields:
1. Merkle Root (hash) of the payload
2. Nonce
3. Block header hash
4. Previous Block header
5. Difficulty target

Where the Merkle root hash is the “fingerprint” of the underlying data to be tracked, the nonce is a random bit input, and the block header hash is a hash generated from both of them combined with starting with a certain number of 0 in front of it. An example would be “0000000000000000001c07c33d665310159dfc960b4aa46ff88035709fc1c0ca”, where the number of 0s is the difficulty target.

Because the there is no way to know what the result of the hash will be before generating it, one needs to brute force a number of nonces until a block header is found by chance which matches the difficulty target. Therefore, the difficulty target is statistically representative of the number of computations that were performed in order to find that block header. This way it becomes possible to measure how much work was put into any block, but also how much cumulative work was expended to generate the whole chain.

Once the block header was found, the minder broadcasts it as widely as possible to ensure that his block is used by other nodes to build on top of.

The majority of PoW blockchain technologies today use a single-threaded approach to chaining blocks, meaning they reference in their block headers only a single previous block header. This is to incentivize nodes to always mine only on one block, ensuring that all nodes agree on a single state of the blockchain. The side effect however is that the speed at which blocks can be found, and therefore the speed at which transactions can be processed by the distributed state machine is limited by the aggregate network connectivity of most of the nodes. If network connectivity is bad, or if PoW difficulty is somehow set too low, then block headers don’t have enough time to propagate through the whole network and multiple valid block headers are found in parallel, effectively splitting the network state. Such splits reduce the total security of the network by reducing the PoW backing each individual chain.

Auxiliary PoW or Merged Mining

Traditional Auxiliary PoW (AuxPoW) uses the same concept as PoW with the added rule that the miners doing the PoW to secure chain B also include the block header hash in the transaction of a stronger mother chain A, and that chain B nodes will observe the chain A transactions for whether any valid block header is included in the chain A blocks. If so it checks which PoW is higher, that of chain A or B, an if chain A is higher it accepts that block as valid for chain B as well.

FIG. 4 below illustrates a traditional AuxPOW.

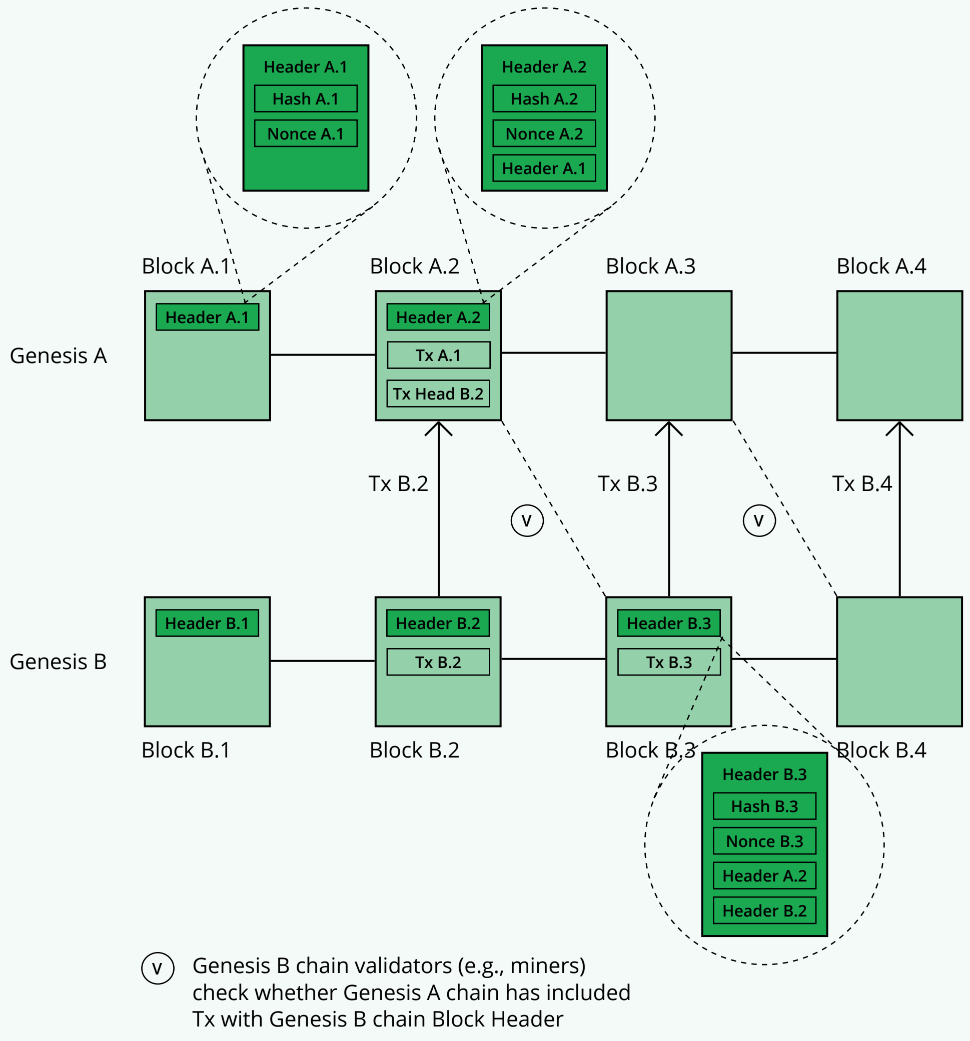

FIGURE 4

Mutual AuxPoW

ULedger uses a modified version of AuxPoW by making the block headers and PoW of both chain A and chain B, as well as any chain N mutually includable and verifiable. This means that the block headers with the individual PoW are included in any number of chains and that chain A can observe all those chains for the highest PoW. If any of those chains has a higher PoW, and has chain A's block header included as a transaction in its latest block, chain A references that external chain block as valid and as the highest PoW. This can be performed with multiple chains which use the same PoW hashing algorithm. It assumes that chains anchor and further secure their chains this way in a many-to-many relationship.

FIG. 5 below provides a representative illustration of Mutual AuxPoW.

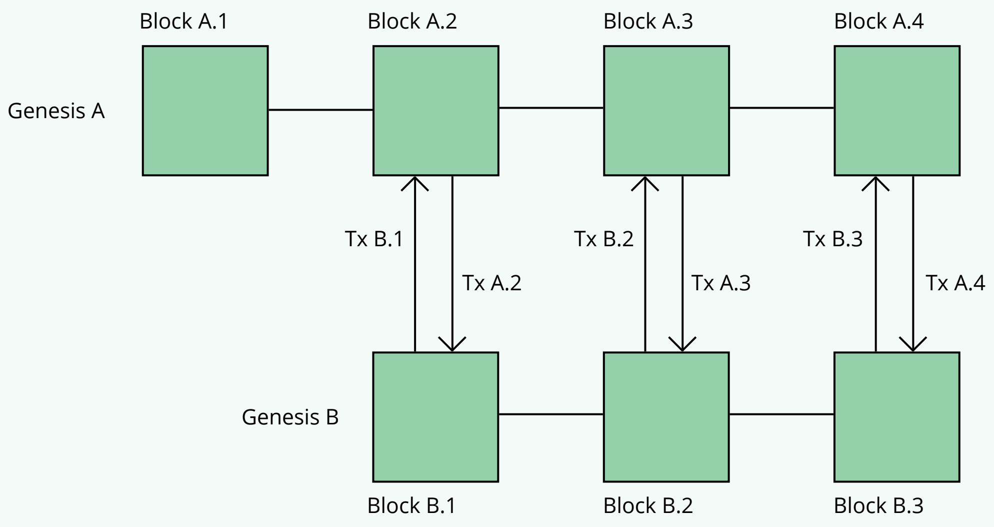

FIGURE 5

GHOST AuxPoW

State Machines are incentivized to include other blocks in their Merkle DAG because they want to have as high a GHOST AuxPoW as possible to be faster and more immutable.

In most circumstances, it should be cheaper for state machines to collect others Block Headers with their GHOST AuxPoW and build their own PoW on top of that than to generate brute force a stronger PoW of their own. This incentivizes high network connectivity and cross-merkelization, hence faster immutability across the network. It also incentivizes state machines to broadcast their block headers as quickly and as broadly as possible to be included in others' blocks which still are not too far along the cumulative PoW timeline.

This means, if a state machine is removed far from the main network cluster, for its Merkle DAG to be just as quickly immutable across the network, it needs to perform disproportionately more PoW than the rest of the network. In the ULedger network connectivity and PoW are mostly sliding scales.

Under the condition of network closeness, the relatively little PoW that needs to be done by each state machine also furthers the overall irreversibility of everybody's Merkle DAG as well as acts as DDOS protection. The blocks of state machines which are far removed from the main cluster take disproportionately more time to become immutable, unless the state machines perform disproportionate amounts of PoW.

FIG. 6 below provides a representative illustration of the GHOST AuxPoW.

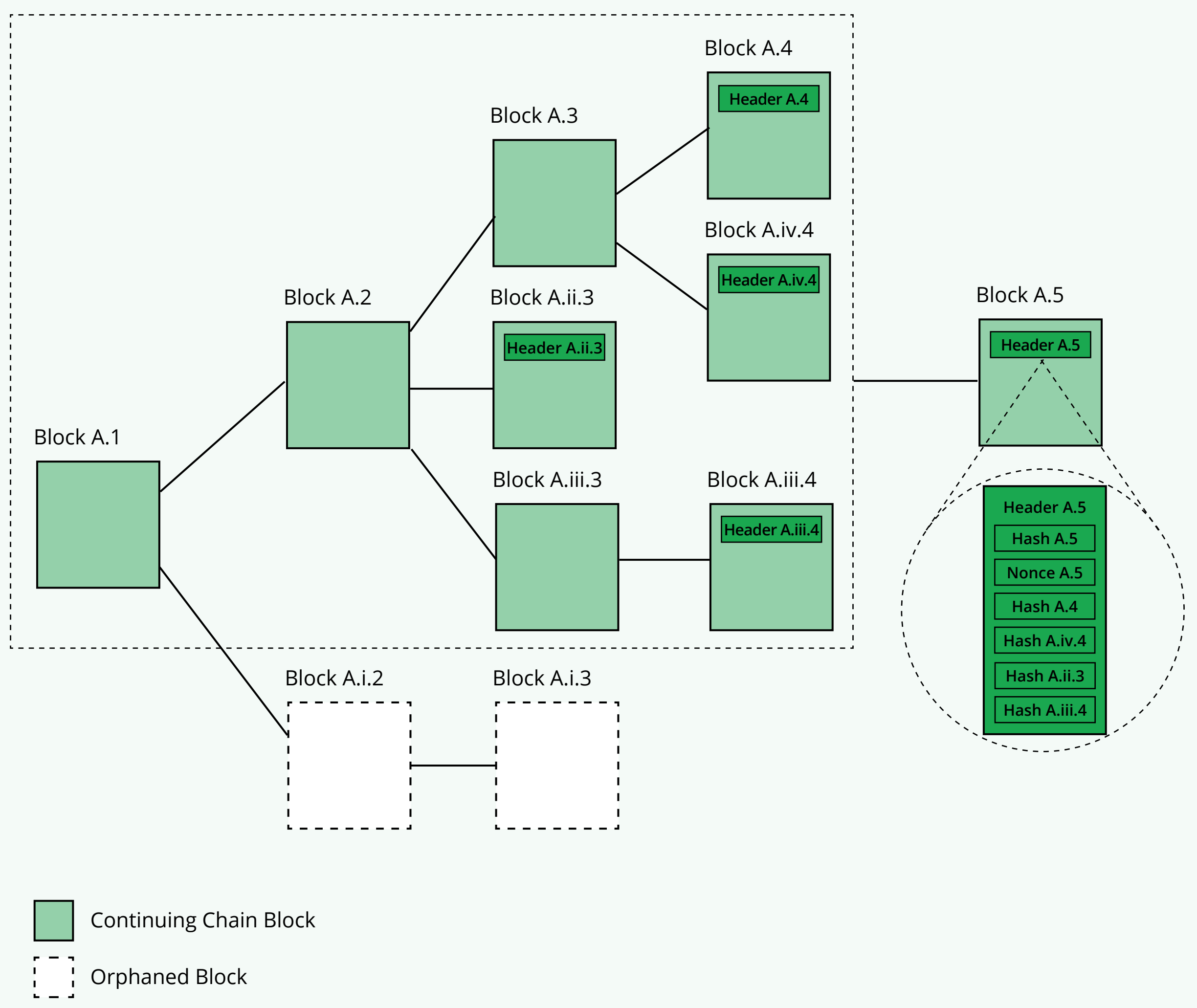

FIGURE 6

GHOST Protocol and AuxPoW Synthesis

Cumulative Difficulty Adjustment

ULedger does not include a global network-wide single-PoW difficulty adjustment. PoW difficulty adjustments happen on a per single Merkle DAG level. The adjustment is for a minimum cumulative PoW. Each state machine tries to include a certain amount of cumulative PoW in its own chain, before generating their own PoW on top of that for broadcasting.

Each state machine readjusts its cumulative difficulty target at every block the state machine generates. This means that state machines generate their own new block when they either a) have collected a small number of other blocks with high PoW each, or b) include a large number of blocks with small PoWs each.

Difficulty Wave Dissemination Pattern

Because of how the network is structured and how GHOST AuxPoW is generated/combined and disseminated, if a disproportionately high single PoW hits the network, that PoW will dominate all Merkle DAGs into which that PoW gets included. Furthermore, that PoW will readjust the difficulty of all those chains. This difficulty change hits the network like a wave. At any given time multiple such waves can hit multiple parts of the network.

ULedger makes use of the same concept of PoW but instead of referencing a single block header ULedger references multiple previous block headers all at once. The block headers being included are:

1. The single highest single PoW header received in that interval or included in chains being observed
2. The highest cumulative PoW block headers (considering total history of the chain(s))
3. Block headers of chains with which internal transactions have interacted with in this interval.

Including all three types of PoW interpretation has the following benefits and effects:

- Blocks with single-block extremely high PoW will propagate further through the network compared to low-PoW blocks. This will ensure that those blocks and hence blockchain history is included in a larger number of blockchains. We expect mining farms similar to Bitcoin to offer such services for propagating extremely high value transactions as fast as possible.
- Observing cumulative PoW block headers requires receiving blocks in the publish-subscribe network from the broadcasting nodes. Nodes are incentivized to do that for blockchains with which they regularly interact. Our assumption is that most interactions are repeat interactions between already known nodes. This ensures very fast and quickly verifiable immutability across closely knit blockchains, which will be the case within industries, supply chains, countries, etc.
- Every node will want to include the block header of the interacting blockchain which references the transactions that have been exchanged.

Proof of misbehavior

If any node receives or interacts with state which is inconsistent with some other state as signed by the same Identity (e.g. same block height, inconsistent order, inconsistent Merkle DAG, etc), then those two or more mutually exclusive states are in turn packaged together, cross-merkelized by the node that found the inconsistency, and forwarded to all neighboring nodes in full.

Nodes which receive such inconsistency packages check themselves whether the states are mutually exclusive, and if they are, forward the package to all other neighboring nodes as well. Furthermore, they check whether they have any state in their Merkle DAG which in the past has interacted with that inconsistent state. If there is an additional inconsistency they repackage that further, blacklist that state and all subsequent state that used it as input, and forward the newly discovered inconsistency as well. If subsequent state is corrupted due to misbehaving input state, the node forwards that information to past interacting identities which relied on subsequent input state as well. This way all nodes slowly create a common repository of inconsistent state, with associated misbehaving Identities. This information can be used by system administrators to restrict, block, or ban cross-merkelization and other types of interaction with these Identities.

Misbehavior also extends to submitting false PoW. By default, misbehaving Identities are fully blacklisted from the network. This blacklisting, together with the PoW required for cross-merkelization acts as a barrier to Sybil attacks on the network. It is possible to create state transition reputation systems with this data, effectively forecasting how trustworthy any future interaction with those identities will be in the future.

The approach of ULedger is "Trust but verify." By default all nodes trust one another to do the correct state transitions (unless the system administrator sets up other internal rules), but every state transition must be verifiable. Contrary to many other systems, in ULedger a resilient DHT Gossip protocol is used for propagating the packages of misbehaviors. The order of events is in this case only of secondary concern as the state itself always includes the relative order of events and timestamping. The order in which the individual nodes store and receive the misbehavior packets is also of no concern and highly subjective.

Timekeeping

Our goal is to do for Time what IPFS is doing for State. We believe humanity has the potential of becoming an interplanetary species. This step requires new technological approaches that consider relativistic space-time and the bottleneck that is the speed of light. Current time and networking systems used in our planetary network (the Internet) do not take these crucial requirements into account. That said, even before we reach for the stars, very similar requirements exist even locally on earth. Local use cases of ULedger include:

- Edge computing where devices have inconsistent network connectivity and have a strong requirement for synchronization and event ordering
- IoT, again for the same reasons as Edge Computing
- Data Integrity systems
- Internet of Agreements / Smart Contracts
- Integrated business databases and API economy
- Deferred super massive cloud computing (e.g. BOINC, Extremely Large Telescope, etc)
- Provable centralized Oracles/data feeds for other linked timestamps and Blockchains

Order of immutability

Interactions between nodes are ordered in several ways in ULedger.

The first method of absolute ordering between pre-existing mutually known nodes are Event-Chains which are a modified Vector Clock design. The so resulting Merkle Roots of the Event-Chains are regularly timestamped and verified via the modified Roughtime Protocol. This way timestamped and blockchain verified proofs of those events are provided by several parties, including parties which might not have interacted with the event-chain at all. These proof bundles can in turn be used to build new event chains and use the bundles as starting points. Any two chains starting a new Event-Chain with these bundled inputs first checks whether the bundles contain any misbehavior.

The Blockchain keeps track of the passage of time, that is the passage of events being observed and proven in this fashion. It is a record of “this is what I have been aware at this point in time”. Each node only trusts its own blockchain, because the order in which something might have been observed can and will be different under bad network connectivity or very far distances/space-time curvature.

Starting with Event-Chains being fully ordered, continuing with the roughly ordered modified Roughtime Protocol, and concluding with the relatively ordered independently verifiable observations of reality contained in each individual blockchain, we believe our system makes it possible to independently prove data and the ordering of events across vast distances, bad network connectivity, and even mutually distrusting parties.

Appendix

Key Terms

- **State/Event** - For the purposes of this paper we are referring to State and Event interchangeably.
- **State Machines** - State Machines are an abstract model of computation describing the processes and steps involved in transitioning state to another state. This transition can be triggered by external inputs or internally by keeping track of time as an internal trigger. State Machines are always input-output based. One can think of them as black boxes performing some processes on inputs coming in on the one side, and outputs coming out the other side. In this paper, the concept of state machines is applied more broadly to enclosed systems performing state transitions such as computers, software programs, distributed computational networks, human institutions such as companies and governments performing certain processes, as well as people.
- **Identity** - For the purposes of this paper we define Identities as verifiable Public Keys in an asymmetric cryptographic system which can be, but must not necessarily be, within a Public Key Infrastructure (PKI).
- **Independent Proof** - For the purposes of this paper we define an independent proof as any cryptographically based proof of 1) the consistency of any given state, 2) the correct ordering and timestamping of that state in relation to other state, 3) the correct and irrefutable attribution of that state to an identity. Furthermore, for something to be an Independent proof the method by which the proof happens must be non-interactive, meaning, the content of the received state must be enough to perform the proof, without the need for additional state to be exchanged with any other state machine.
- **Immutability** - Is any state which cannot be altered after it was created. Any transitions of the state is append-only. The appended state in turn is immutable and only represents the difference to the original state (delta).
- **System Time** - The perceived passage of time by a single state machine.
- **Transactions** - For the purposes of this paper, transactions are independently verifiable state being exchanged with other state machines.
- **Trusted Third Parties (TTP)** - TTPs are 3rd parties which stand between two or more interacting parties and provide trust in some fashion between the parties which don't trust one another. TTPs generally have the ability to selectively censor interactions between parties.

- **Proof of Work (PoW)** - From the Bitcoin Wiki: "A proof of work is a piece of data which is difficult (costly, time-consuming) to produce but easy for others to verify and which satisfies certain requirements. Producing a proof of work can be a random process with low probability so that a lot of trial and error is required on average before a valid proof of work is generated."
- **Merkle Tree** - Also called a Hash Tree, is a data structure where hashes of input state (called leaves) are further hashed together until all leaves are represented in a single hash, also called the Merkle root. This data structure looks like a reverse tree. Merkle Trees are an efficient way of verifying contents of a lot of state. Merkle Trees are mostly used for trust-less verification of state being transmitted between state machines, such as in BitTorrent, Git, Bitcoin, IPFS, etc.
- **Merkle Audit Proof** - A Merkle Audit Proof lets you independently verify that a single leave of a Merkle Tree is consistent as well as represented in the correct order within the Merkle Tree without having to expose the entire state or Merkle Tree to the proof recipient.
- **Direct Acyclic Graph (DAG)** - A DAG is a finite directed graph of nodes with no directed cycles, meaning there is no way for lower nodes to connect up with higher nodes again.
- **Merkle DAG** - A Merkle DAG is a more general data structure of a Merkle Tree. It is an arbitrarily large DAG where each node is a leave in a Merkle Tree and the direction of the DAG is determined by the hashing mechanism itself.
- **Linked Timestamps** - Is a type of timestamp which can be trusted due to mostly cryptographic approaches to linking the timestamps. Linked timestamps cannot be forged as they would break subsequent links in their chain. Linked Timestamps typically are constructed as Merkle DAGs.
- **Hash Calendars** - A Hash Calendar is a Linked Timestamp in the form of a Merkle DAG where each leaf of the Merkle DAG represents a new second or other clock interval. Events that happened within that second are typically Gossip Protocols, a communication protocol which does not assume reliable communication channels nor network structure consistency.
- **Event-chains** - Event-chains are input-output based chains of events. Each event is an output generated by another state machine. Event-chains always start with an arbitrary input state created by one or more state machines. Event-chains only deal with one specific "thread" across multiple state machines. The confluence between all these threads across multiple state machines is done through the cross-merkelization process.

Contact

ULedger, Inc.

910 Main St. Boise, ID 83702

info@uledger.co

208-954-6874

Meet our Team

Josh McIver
Co-Founder & CEO

Dave Otander
Vice President & Compliance

Taulant Ramabaja
Co-Founder & CTO

Jolene Anderson
Enterprise Sales

Pete Anewalt
COO & Development

William Mougayar
Token Event Advisor

Advisory Board

Stephen Orenberg
Former President of Kaspersky Labs

Jerry Dunn
President A10 Capital

Eric Quigley
CEO Mercury Global

Mac Chavaria
Managing Partner Basin Pacific Insurance

Gary McIver
Retired Partner Deloitte

Jerry Henley
President Rubicon Capital

Lorien Newman
ICO Advisor

Sal Papa
CCO & General Counsel
BAM Advisor Network