US 20230179435A1

(19) **United States**
(12) **Patent Application Publication**
Heinecke et al.

(10) **Pub. No.: US 2023/0179435 A1**
(43) **Pub. Date: Jun. 8, 2023**

(54) **SYSTEM AND METHOD FOR CREATING AND MAINTAINING IMMUTABILITY, AGREEMENT AND AVAILABILITY OF DATA**

(71) Applicant: **Blocky, Inc.**, Bozeman (MT)

(72) Inventors: **Taylor Heinecke**, Bozeman, MT (US); **David L. Millman**, Bozeman, MT (US); **Mike P. Wittie**, Bozeman, MT (US)

(21) Appl. No.: **18/072,574**

(22) Filed: **Nov. 30, 2022**

Related U.S. Application Data

(60) Provisional application No. 63/286,382, filed on Dec. 6, 2021.

Publication Classification

(51) **Int. Cl.**

H04L 9/00	(2006.01)
H04L 9/06	(2006.01)
H04L 9/32	(2006.01)

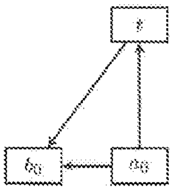
(52) **U.S. Cl.**
CPC ***H04L 9/50*** (2022.05); ***H04L 9/0618*** (2013.01); ***H04L 9/3297*** (2013.01)

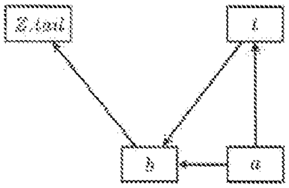
(57) **ABSTRACT**

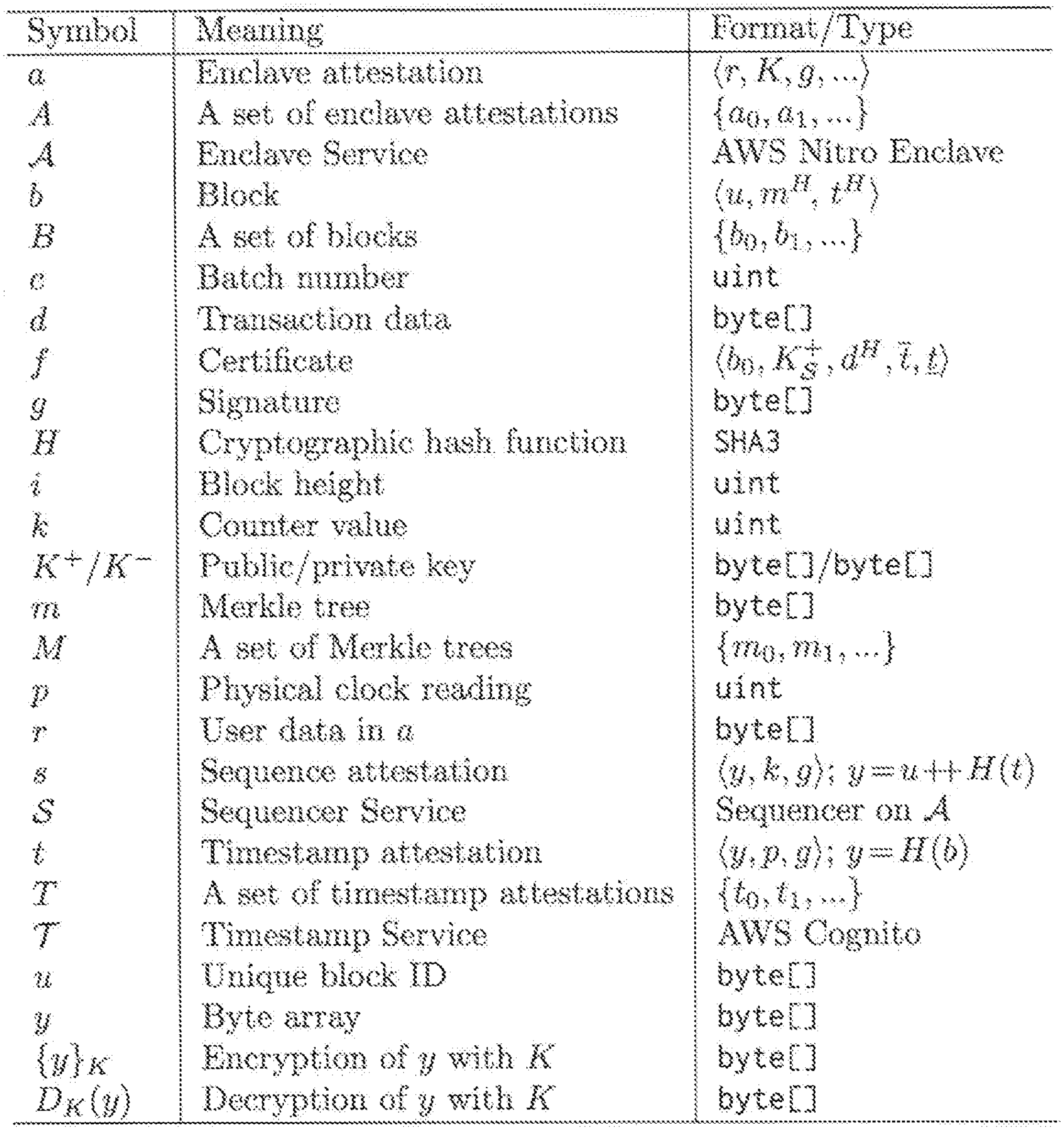
A method for creating and maintaining immutability, agreement and availability of data including the steps of building an alternating structure of blocks and timestamp attestations; determining an order of blocks; determining which of the blocks are on a main chain; and using a trusted replication service to replicate all of the blocks, timestamp attestations, sequence attestations, and leaves of the Merkle trees. A computer program product comprising a storage device storing instructions in a non-transitory manner, which instructions cause the computing device to create and maintain immutability, agreement and availability of data according to the foregoing method. A computing device comprising a processing unit, memory or other storage device coupled to the processing unit, the memory or other storage device storage instructions, which cause the computing device to create and maintain immutability, agreement and availability of data according to the foregoing method.

```
Algorithm  $omc(b_0, K_S^+, K_T^+, B, T, A)$
$Z \leftarrow []$  ▷ Main chain blocks and timestamp attestations
$k \leftarrow -1$  ▷ Sequence number of the last block in $Z$
▷ Check block $b_0$ ◁
if $\exists b \in B, t \in T, a \in A \mid H(b) = H(b_0) \wedge$
   $t.y = H(b) \wedge validate(K_T^+, t) \wedge a.r.k = 0 \wedge$
   $a.r.y = b.u \mathbin{++} H(t) \wedge check(K_S^+, a)$ then
    $Z \leftarrow Z \mathbin{++} b \mathbin{++} t$  ▷ Add $b_0, t_0$ to the main chain
    $k \leftarrow 0$  ▷ Record seq, num. of $b_0$
else
    return $\varnothing$  ▷ Block $b_0$ was invalid
▷ Add main chain blocks and attestations to $Z$ ◁
loop
    ▷ Find all immediate successors to last block in $Z$ ◁
    $C \leftarrow \{(b,t,a) \mid b \in B \wedge t \in T \wedge a \in A \wedge$
                 $b.t^H = H(Z.tail) \wedge$
                 $t.y = H(b) \wedge validate(K_T^+, t) \wedge$
                 $a.r.y = b.u \mathbin{++} H(t) \wedge check(K_S^+, a)\}$
    if $|C| = 0$ then
        return $Z$  ▷ No more successors
    ▷ Find successor with minimal sequence attestation ◁
    $z \leftarrow \mathrm{argmin}_{c \in C}\, c.a.r.k$
    ▷ Fill in the sequence gap ◁
    $G \leftarrow \{(b,t,a) \mid b \in B \wedge t \in T \wedge a \in A \wedge$
                 $t.y = H(b) \wedge validate(K_T^+, t) \wedge$
                 $a.r.y = b.u \mathbin{++} H(t) \wedge check(K_S^+, a) \wedge$
                 $a.r.k \in [k, z.a.r.k]\}$

    if $|G| \neq z.a.r.k - k - 1$ then
        ▷ Missing a sequence attestation ◁
        return $Z$
    ▷ Extend the main chain ◁
    $Z \leftarrow Z \mathbin{++} z.b \mathbin{++} z.t$
    $k \leftarrow z.r.k$
```







Symbol	Meaning	Format/Type
a	Enclave attestation	$\langle r, K, g, ... \rangle$
A	A set of enclave attestations	$\{a_0, a_1, ...\}$
$\mathcal{A}$	Enclave Service	AWS Nitro Enclave
b	Block	$\langle u, m^H, t^H \rangle$
B	A set of blocks	$\{b_0, b_1, ...\}$
c	Batch number	`uint`
d	Transaction data	`byte[]`
f	Certificate	$\langle b_0, K_{\mathcal{S}}^{+}, d^H, \bar{t}, \underline{t} \rangle$
g	Signature	`byte[]`
H	Cryptographic hash function	`SHA3`
i	Block height	`uint`
k	Counter value	`uint`
K^+/K^-	Public/private key	`byte[]/byte[]`
m	Merkle tree	`byte[]`
M	A set of Merkle trees	$\{m_0, m_1, ...\}$
p	Physical clock reading	`uint`
r	User data in a	`byte[]`
s	Sequence attestation	$\langle y, k, g \rangle$; $y = u \mathbin{++} H(t)$
S	Sequencer Service	Sequencer on $\mathcal{A}$
t	Timestamp attestation	$\langle y, p, g \rangle$; $y = H(b)$
T	A set of timestamp attestations	$\{t_0, t_1, ...\}$
$\mathcal{T}$	Timestamp Service	AWS Cognito
u	Unique block ID	`byte[]`
y	Byte array	`byte[]`
$\{y\}_K$	Encryption of y with K	`byte[]`
$D_K(y)$	Decryption of y with K	`byte[]`

FIG. 1

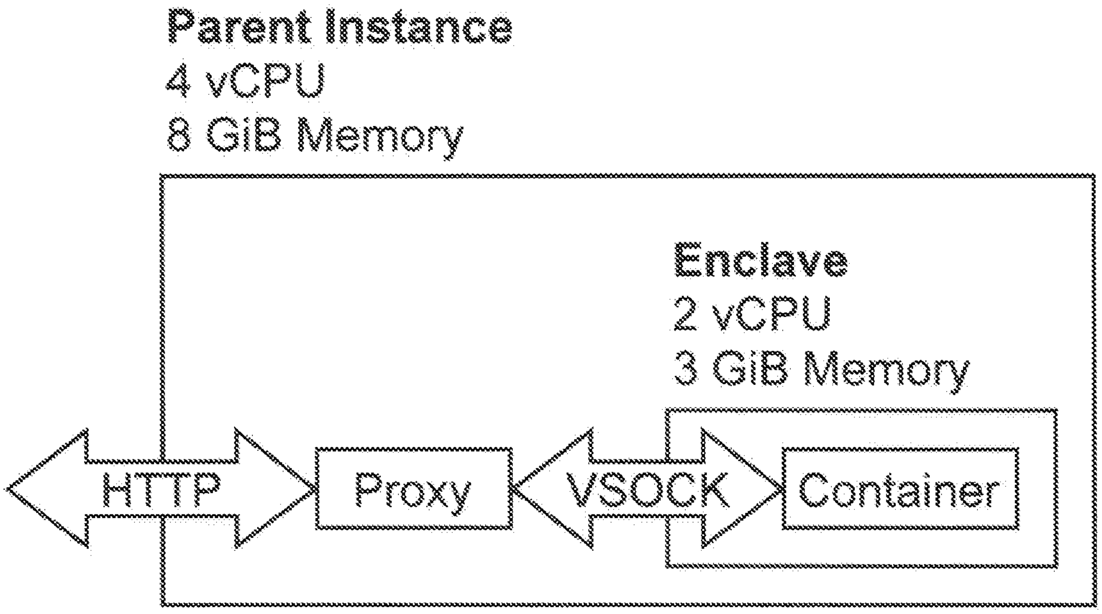

Parent Instance
4 vCPU
8 GiB Memory
Enclave
2 vCPU
3 GiB Memory
HTTP
Proxy
VSOCK
Container

FIG. 2

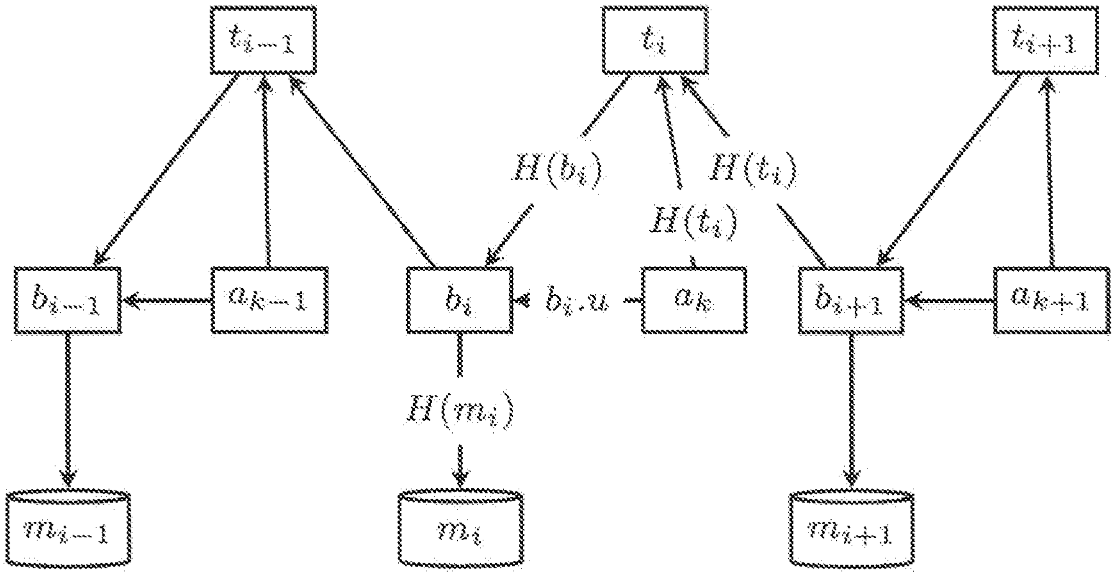


FIG. 3

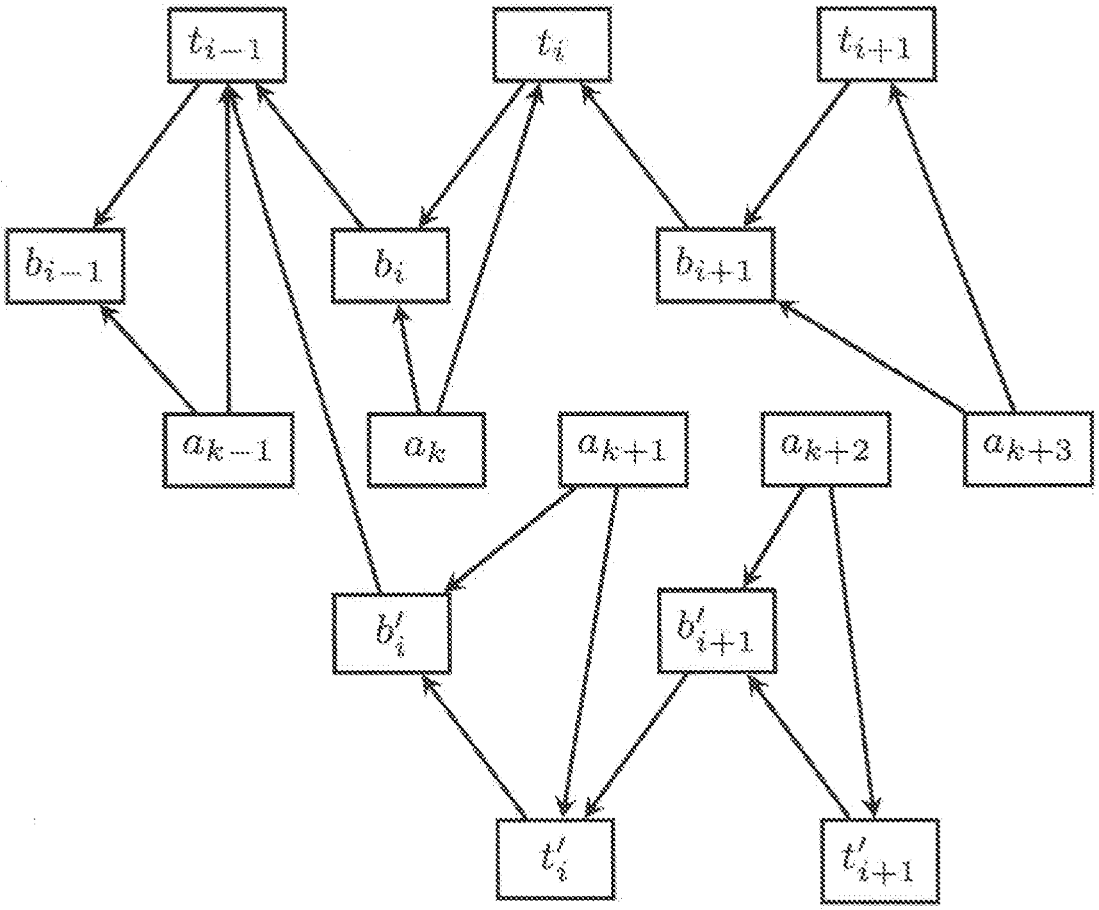


FIG. 4

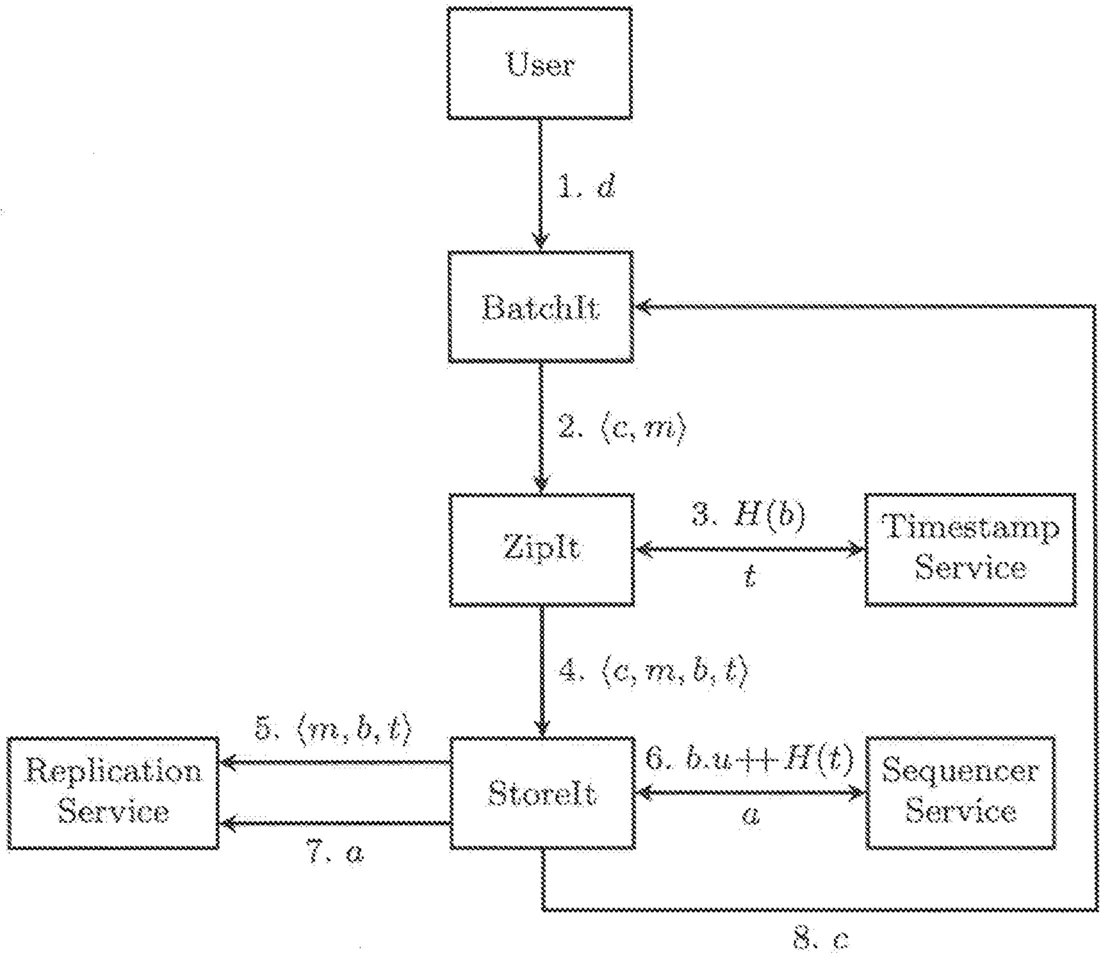
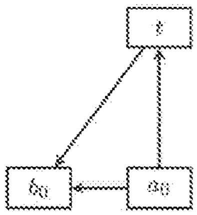

User
1. d
BatchIt
2. $\langle c, m \rangle$
ZipIt
3. $H(b)$
t
Timestamp Service
4. $\langle c, m, b, t \rangle$
5. $\langle m, b, t \rangle$
Replication Service
StoreIt
6. $b.u{+}{+}H(t)$
a
Sequencer Service
7. a
8. c

FIG. 5

```
Algorithm  $omc(b_0, K^{\ddagger}_{\mathcal{S}}, K^{\ddagger}_{\mathcal{T}}, B, T, A)$
$Z \leftarrow []$    ▷ Main chain blocks and timestamp attestations
$k \leftarrow -1$    ▷ Sequence number of the last block in $Z$
▷ Check block $b_0$    ◁
if $\exists b \in B, t \in T, a \in A \mid H(b) = H(b_0) \wedge$
   $t.y = H(b) \wedge validate(K^{\ddagger}_{\mathcal{T}}, t) \wedge a.r.k = 0 \wedge$
   $a.r.y = b.u \mathbin{++} H(t) \wedge check(K^{\ddagger}_{\mathcal{S}}, a)$ then
    $Z \leftarrow Z \mathbin{++} b \mathbin{++} t$    ▷ Add $b_0, t_0$ to the main chain
    $k \leftarrow 0$    ▷ Record seq. num. of $b_0$
else
    return $\varnothing$    ▷ Block $b_0$ was invalid
▷ Add main chain blocks and attestations to $Z$    ◁
loop
    ▷ Find all immediate successors to last block in $Z$    ◁
    $C \leftarrow \{(b, t, a) \mid b \in B \wedge t \in T \wedge a \in A \wedge$
                $b.t^{H} = H(Z.tail) \wedge$
                $t.y = H(b) \wedge validate(K^{\ddagger}_{\mathcal{T}}, t) \wedge$
                $a.r.y = b.u \mathbin{++} H(t) \wedge check(K^{\ddagger}_{\mathcal{S}}, a)\}$
    if $|C| = 0$ then
        return Z    ▷ No more successors
    ▷ Find successor with minimal sequence attestation    ◁
    $z \leftarrow \mathrm{argmin}_{c \in C}\, c.a.r.k$
    ▷ Fill in the sequence gap    ◁
    $G \leftarrow \{(b, t, a) \mid b \in B \wedge t \in T \wedge a \in A \wedge$
                $t.y = H(b) \wedge validate(K^{\ddagger}_{\mathcal{T}}, t) \wedge$
                $a.r.y = b.u \mathbin{++} H(t) \wedge check(K^{\ddagger}_{\mathcal{S}}, a) \wedge$
                $a.r.k \in [k, z.a.r.k]\}$

    if $|G| \neq z.a.r.k - k - 1$ then
        ▷ Missing a sequence attestation    ◁
        return Z
    ▷ Extend the main chain    ◁
    $Z \leftarrow Z \mathbin{++} z.b \mathbin{++} z.t$
    $k \leftarrow z.r.k$
```



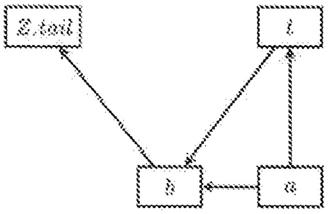
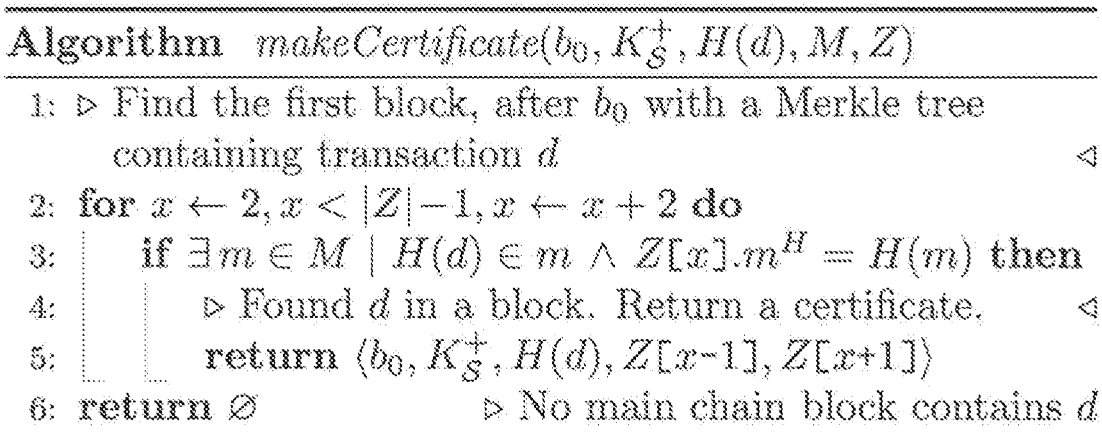


FIG. 6



```
**Algorithm** $makeCertificate(b_0, K_S^+, H(d), M, Z)$
▷ Find the first block, after $b_0$ with a Merkle tree containing transaction $d$ ◁
for $x \leftarrow 2, x < |Z|-1, x \leftarrow x+2$ do
    if $\exists m \in M \mid H(d) \in m \wedge Z[x].m^H = H(m)$ then
        ▷ Found $d$ in a block. Return a certificate. ◁
        return $\langle b_0, K_S^+, H(d), Z[x-1], Z[x+1] \rangle$
return $\varnothing$                ▷ No main chain block contains $d$
```

FIG. 7

SYSTEM AND METHOD FOR CREATING AND MAINTAINING IMMUTABILITY, AGREEMENT AND AVAILABILITY OF DATA

CROSS-REFERENCE TO RELATED APPLICATION

[0001] Pursuant to 35 U.S.C. § 119(e), this application claims prior back to U.S. Patent Application No. 63/286,382 filed on Dec. 6, 2021.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] This invention was made with government support under Contract No. 2052375 to Blocky, Inc., awarded by the National Science Foundation. The government has certain rights in the invention.

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0003] The present invention relates generally to the field of distributed ledger technologies, or blockchains, and, more particularly, to a system and method for creating and maintaining immutability, agreement and availability of data.

2. Description of the Related Art

[0004] Distributed ledger technologies (DLTs), or blockchains, make possible a growing number of decentralized alternatives to traditionally centralized financial, governmental, and legal services. Proof of work blockchains made the development of decentralized services only marginally practical due to high delay, low throughput, and high and unpredictable cost of recording and executing transactions. Newer blockchain proposals address these limitations with novel consensus mechanisms, faster block distribution techniques, and less speculative transaction pricing [1]-[3]. Fundamentally, however, the performance of blockchains is limited by their reliance on distributed consensus [4], seemingly central to distributed ledger correctness guarantees.

[0005] Blockchains are a mechanism that guarantees immutability, agreement, and availability of data. Blockchains provide immutability by cryptographically linking blocks in a way that makes their retroactive modification without renewed distributed agreement near-impossible. Agreement comes from a blockchain's distributed consensus protocol, which ensures that the creation of new blocks follows preset rules. Finally, availability comes from the replication of blockchain state among distributed nodes that prevents its deletion and, in the case of public blockchains, provides censorship resistance.

[0006] In other words, blockchains achieve their correctness guarantees by relying on distributed algorithms that operate among a number of nodes separated by a network. When distributed algorithms need to maintain consistent state, but the network is slow, the result is momentary interruptions of system availability [5], perceived by users as degraded throughput. Though blockchain mechanisms vary, the tradeoff between system consistency and availability results in network performance placing a limit on blockchain throughput.

[0007] The present invention proposes to improve performance by revisiting the underlying trust relationships between a blockchain and its users. Blockchains are considered trustless peer-to-peer systems because to use them, users do not need to trust each other; other widely used systems are based on strong, but limited, trust relationships. For example, users generally trust certificate authorities' assertions of public keys. Similarly, authentication services based on OAuth 2.0 [6] are generally trusted to issue correct authentication tokens. Such trust relationships emerge from a clear self-interest in the correctness of the service by its provider.

[0008] The present invention demonstrates that limited trust in third-party services can give rise to a novel system to create immutability, agreement, and availability through a mechanism that transfers users' trust in these third-party services into trust of blockchain correctness guarantees. Specifically, the present invention incorporates: (a) the design and implementation of trusted timestamp, sequencer, and replication services; and (b) the design and implementation of a blockchain construction method that transfers the trust in such services into trust in blockchain correctness guarantees.

BRIEF SUMMARY OF THE INVENTION

[0009] The present invention is a method for creating and maintaining immutability, agreement and availability of data comprising the steps of: building an alternating structure of blocks and timestamp attestations by: constructing a first Merkle tree having leaves of transaction data; creating a first block that contains a root of the first Merkle tree; using a trusted timestamp service to create a timestamp attestation over the first block; constructing a second Merkle tree having leaves of transaction data; creating a second block that contains a root of the second Merkle tree; linking the second block to the timestamp attestation of the first block; using a trusted timestamp service to create a timestamp attestation over the second block; and repeating the foregoing steps over a series of blocks and timestamp attestations to create the alternating structure in which each block is linked to the timestamp attestation of an immediately preceding block; determining an order of blocks by: using a trusted sequencer service to assign a sequence attestation over each block and its timestamp attestation, wherein each sequence attestation has a unique number; and wherein each block has a height, creating a total order of the blocks based on the height of each block and the sequence attestation assigned to each block; determining which of the blocks are on a main chain by: checking validity of the timestamp attestation over the first block; checking validity of the sequence attestation over the first block and of the timestamp attestation over the first block; adding the first block and the timestamp attestation over the first block to the main chain; wherein the main chain has a last block, wherein the last block has a sequence attestation, extending the main chain from the last block by: finding all successor blocks of the last block, wherein each successor block has a timestamp attestation and a sequence attestation; identifying a successor block with a sequence attestation that is lower than the sequence attestations of all other successor blocks; checking validity of the timestamp attestation and validity of the sequence attestation of the successor block identified in step (c)(iv)(2); and if all blocks with sequence attestations between the sequence attestation of the last block on the main chain and the sequence attestation of the successor block with the lowest sequence attestation can be found, adding to the main chain the successor block with the lowest sequence attestation and the timestamp attestation over the successor block with the lowest sequence attestation; and repeating the foregoing steps over a set of blocks, timestamp attestations, and sequence attestations; and using a trusted replication service to replicate all of the blocks, all of the timestamp attestations, all of the sequence attestations, and all of the leaves of the Merkle trees.

[0010] The present invention is also a computer program product comprising a storage device storing instructions in a non-transitory manner, which instructions, when executed by a processing unit of a computing device, cause the computing device to: create and maintain immutability, agreement and availability of data using the method described above. In addition, the present invention is a computing device comprising a processing unit, memory or other storage device coupled to the processing unit, the memory or other storage device storage instructions, which, when executed by the processing unit, cause the computing device to: create and maintain immutability, agreement and availability of data using the method described above.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] FIG. **1** is a table of definitions of notations used herein.

[0012] FIG. **2** is a diagram of a parent instance of AWS Nitro Enclaves.

[0013] FIG. **3** is a diagram of a blockchain formed by the present invention.

[0014] FIG. **4** is a diagram of a fork in a blockchain formed by the present invention.

[0015] FIG. **5** is a flow diagram of the write function of the present invention.

[0016] FIG. **6** is a flow diagram of the omc function of the present invention.

[0017] FIG. **7** is a flow diagram of the makeCertificate function of the present invention.

DETAILED DESCRIPTION OF INVENTION

A. Trusted Services

[0018] The present invention makes a departure from distributed blockchain implementations to provide its guarantees of immutability, agreement, and availability based on trusted services. This section describes the abstractions and implementations of trusted timestamp, trusted sequencer, and trusted replication services within the context of the present invention.

[0019] 1. Trusted Timestamp

[0020] The function of the trusted Timestamp Service is to provide accurate and trustworthy physical clock timestamps. A Timestamp Service $\mathcal{T}$ maintains a public/private key pair $K_{\mathcal{T}}^{+}/K_{\mathcal{T}}^{-}$ and an accurate physical clock. A Timestamp Service provides the following abstract interface:

t:=timestamp(y)

true/false:=validate(K,t)

[0021] The timestamp function takes bytes y as input, reads the physical clock value p, and uses these to create a timestamp attestation t. The attestation is a tuple t=⟨ y,p,g⟩ , with signature g={H(y,p)}$_{K_{\mathcal{T}}^{-}}$, where H a cryptographic hash function. The validate function takes a key K and a timestamp attestation t as input to determine that y and p are correctly signed, or that H(t,y,t,p)=D$_K$(t,g), where D is a decryption function. Note that the "," operator is used for member selection. When users trust a Timestamp Service and validate($K_{\mathcal{T}}^{+}$,t) returns true, they can trust that the Timestamp Service $\mathcal{T}$ has witnessed bytes t.y at time t.p. Note that while the timestamp function must execute on the trusted Timestamp Service, the validate function may be executed by a user as long as $K_{\mathcal{T}}^{+}$ is well-known.

[0022] The present invention implements the Timestamp Service based on the user authentication service Amazon Cognito™ (auth service). The auth service accepts user credentials (username and password) and when valid, produces a JSON Web Token (JWT). The JWT contains the username, timestamp, and other fields, signed by the auth service.

[0023] To produce a JWT such that it is a timestamp attestation for bytes y, the present invention performs two steps. First, it creates, on the auth service, a user with:

username=$y@bky\cdot sh$

and a random password. Second, it uses the username and password to authenticate with the auth service to produce a JWT. As the JWT contains; y in the username; the auth service's timestamp; and both pieces of data are signed by the auth service, the resulting JWT is sufficient for a timestamp attestation; therefore, a user that trusts Amazon Cognito can trust that bytes y were seen by Amazon Web Services™ (AWS) at a specific time.

[0024] 2. Trusted Sequencer

[0025] The function of the trusted Sequencer Service is to provide consecutive numbers to distinct events, A Sequencer Service S maintains a public/private key pair K_S^{+}/K_S^{-} and a counter. A Sequencer Service provides the following abstract interface:

s:=sequence(y)

true/false:=check(K,s)

[0026] The sequence function takes as input bytes y representing a unique event ID, increments the counter by one, records the counter value as k, and produces a sequence attestation s=⟨ y,k,g⟩ , where g={H(y,k)}$_{g_S^{-}}$. Note that some symbols, such as k, are reused when their meaning can be differentiated by member selection. For example, the physical timestamp t.k can be differentiated from the sequence number s.k because the k's in question are members of different types. The check function takes a key K and a sequence attestation s as input to determine that y and k are correctly signed, or that H(s,y,s,k)=D$_K$(s.g). When users trust a Sequencer Service and check(K_S^{+},g) returns true, they can trust that a unique event represented by s, y was witnessed by the Sequencer Service S as the s, k^{th} event. Similarly to timestamp, the check function may be executed by a user as long as K_S^{+} is well-known.

[0027] The Sequencer Service is based on the security and correctness guarantees provided by the AWS Nitro Enclaves™ trusted execution environment (TEE). A WS Nitro Enclaves creates an isolated execution environment inside an Amazon EC2™ (Elastic Cloud Compute) instance based on the same Nitro Hypervisor technology that provides isolation between Amazon EC2 instances themselves. Inside an Amazon EC2 parent enclave, as shown in FIG. **2**, an enclave runs a container on its own kernel, memory, and virtual CPU (vCPU) resources sequestered from the parent instance. An enclave has no persistent storage, interactive access, or external networking. The only means for the parent instance to interact with an enclave is through a VSOCK socket. The parent instance, however, may proxy external requests for example by running a Hypertext Transfer Protocol(HTTP) server. Finally, applications running inside the enclave may request attestations from the Nitro Secure Module (NSM). An attestation includes information about the enclave environment recorded as hashes of the continuous measurements of the parent instance ID, the enclave image file (container), and the application requesting the attestation. Optionally, the attestation may also include the public key of the enclave and up to 1024 B of user data [7]. NSM packages the attestation as a Concise Binary Object Representation (CBOR)-encoded, CBOR Object Signing and Encryption (COSE)-signed object by the AWS Nitro Enclaves Attestation key pair $K_{\mathcal{A}}^{+}/K_{\mathcal{A}}^{-}$, where $K_{\mathcal{A}}^{+}$ is in a well-known root certificate.

[0028] The implementation of the trusted Sequencer Service on an AWS Nitro Enclaves is as follows. Since AWS Nitro Enclaves produce their own attestations, the Sequencer Service interface is modified to:

$$a{:=}\text{sequence}(y)$$

$$\text{true/false}{:=}\text{check}(K,a)$$

in which an enclave attestation a includes a sequence attestation s and the signed hash of the image file running on the enclave. To provide a trusted Sequencer Service, the parent instance runs a gateway that proxies calls to the sequence function running on enclave $\mathcal{S}$. A sequence request includes bytes y representing a unique event ID and serves as an idempotency key inside $\mathcal{S}$.

[0029] Upon receiving a sequence request the enclave increments an in-memory counter and produces a sequence attestation s=⟨ y,k,g⟩ as defined above. It is important to note that repeated requests to sequence the same y will not increment the counter and produce a new sequence attestation; instead, the Sequencer Service will serve a cached ⟨ y,k,g⟩ . The enclave then requests an enclave attestation a from the NSM, where public key a.K=$K_{\mathcal{S}}^{+}$ and user data a.r=s. Finally, the enclave returns a to the parent instance, which forwards it to the client.

[0030] Upon receiving the enclave attestation a, a client can verify it by calling the check function locally. The check function first verifies that the signature of a against Amazon's certificate of $K_{\mathcal{A}}^{+}$. Next, check verifies the sequence attestation against the enclave's public key by testing that H(a,r,y,a,r,k)=$D_{a.K}$(a,r,g). If both verifications pass check($K_{\mathcal{S}}^{+}$,a) returns true, at which point the client can trust that event a.r.y was assigned the sequence number a.r.k by a Sequencer Service $\mathcal{S}$.

[0031] It is important to note that in the Sequencer Service implementation the enclave generates the key pair $K_{\mathcal{S}}^{+}$ / $K_{\mathcal{S}}^{-}$ on startup, which is distinct across all enclave instantiations. Consequently, every sequence attestation produced by an enclave is unique since the enclave uses a distinct $K_{\mathcal{S}}^{-}$ to sign an incremented k. As a result, it is not possible, even if a Sequencer Service is restarted, to produce two sequence attestations with the same k for different y signed by $K_{\mathcal{S}}^{-}$.

[0032] The last issue is that of user trust. A user may trust that the AWS Nitro Enclaves system works correctly, but the Sequencer Service is based on a specific implementation. The implementation's code and build tools are publicly available and so a user may perform an audit of the code, build an image, and take the hash of the image H(I). Recall that the enclave attestation contains a signed hash of the image running on the enclave H(I'); therefore, when a user trusts an implementation of the Sequencer Service, they can verify that a sequence attestation was produced from the trusted implementation by checking H(I)=H(I'). In other words, the hash of the image that they built matches the hash of the image in the enclave attestation.

[0033] 3. Trusted Replication

[0034] The function of the trusted Replication Service is to provide stable storage to data objects. Since storage nodes have non-zero mean time between failures (MTBF), storage stability is probabilistic and comes from replication of data objects among nodes with mostly independent failures. A Replication Service provides the following abstract interface:

$$\text{replicate}(y)$$

$$y{:=}\text{fetch}(H(y))$$

The replicate function takes as input object bytes y and replicates them across storage nodes under H(y) as the retrieval key. The fetch function takes the hash of the object bytes H(y) as input and returns the object bytes y from one or more replicas.

[0035] Fundamental models of distributed systems commonly assume that nodes have access to stable storage to imply that protocol data survives node failures. To argue the correctness of the present invention, the notion of stability needs to be strengthened and made more specific. In this context, a stable storage service is defined as providing durability, immutability, and verifiability. Durability means that a stored object will remain eventually accessible. Immutability means that a stored object will not change in storage. Verifiability means that a third-party may verify that a storage service provides durability and immutability.

[0036] The Replication Service is implemented based on the correctness guarantees of Write Once, Read Many (WORM) cloud storage systems. WORM systems enable their clients to protect stored data against inadvertent modification, or deletion, to meet regulatory requirements on data retention[8]. Specifically, the Amazon S3™ (Simple Storage Service). Microsoft Azure Blob Storage™, and Google Cloud Storage™ guarantee object immutability through legal/compliance holds on data objects that prevent anyone, including the bucket owner, from deleting or modifying objects [9]-[11]. The durability of storage systems are often measured as follows. For k>1 and time duration d, k-nines per d means that the vendor promises to not loose $(100-10^{(1-k)})$ percent of a user's data over duration d. For example, Amazon, Microsoft and Google provide systems with 11 nines per year of durability by replicating data across availability zones within a region [7], [12], [13]. Finally, cloud storage providers allow, with some custom configuration, to make bucket settings publicly readable, which allows anyone to verify that object holds are enabled. Alternatively, an AWS Nitro Enclaves with read-only credentials may inspect bucket settings and emit publicly verifiable attestations over bucket.

[0037] Although the 11-nines per year durability guarantee is the industry standard, it is not sufficient by itself for storage of the present invention's objects. To achieve sufficient durability, the present invention uses the following storage scheme. First, the system uses random linear network coding [14] to encode each object into 6 shares such that any 3 shards can decode the object. Second, the shards are partitioned into six buckets (two buckets per provider) such that all buckets are in distinct regions. Partitioning allows for failures to be considered independent. Following the Backblaze™ method for computing durability [15], the storage scheme achieves 14-nines of durability over 100 years [16].

[0038] 4. Reliability

[0039] In addition to being trusted, the design of the present invention also requires the Timestamp Service, Sequencer Service, and Replication Service to be reliable in that they can recover from crash failures. While reliability at the cost of temporary unavailability may be assumed for Amazon Cognito and cloud storage services, the same cannot be done for AWS Nitro Enclaves. An enclave may crash, but because it relies on an internally generated key pair and in-memory state to provide unique sequence attestations, the enclave may not be restarted. For the remainder of this document, it is assumed that the Sequencer Service is reliable, at the cost of no new blocks being possible in the case of a Sequencer Service failure.

B. Detailed Description of the Invention

[0040] The present invention generates a new blockchain that provides immutability, agreement, and availability based on strong, but limited, user trust in the Timestamp, Sequence, and Replication services. The key innovation of the present invention is the structure and construction process of its chains that transfer user trust in these services into trust in the present invention's correctness guarantees.

[0041] FIG. **3** depicts the present invention's data representation. Let m be a Merkle tree. Note that H(m) is overloaded to mean the root of the tree (not the hash of the entire tree). A block includes the root of a Merkle tree created from a set of leaves containing transaction data. For a block b, the height of b is the number of blocks on the path from b to b_0. For convenience, a block b at height i can be referred to as b_i and its height denoted as b.i. The block b_i includes the root of the Merkle tree m_i as $b_i.m^H=H(m_i)$. The blocks also include a Universally Unique IDentifier (UUID) [17] field u assigned during block creation.

[0042] The physical timestamp for each block comes from a timestamp attestation created by the trusted Timestamp Service. For example, the timestamp attestation t_i includes the hash of the block b_i as $t_i.y=H(b_i)$. To link each timestamp attestation to the next block, the next block b_{i+1} includes the hash of t_i as $b_{i+1}.t^H=H(t_i)$.

[0043] The sequence number for each block comes from its sequence attestation created by the trusted Sequencer Service. For example, the k^{th} sequence attestation s_k (inside of a_k) includes the unique id of the block b_i. To make dealing with forks easier during the verification process, s_k attests both the block and its timestamp attestations as $s_k.y=b_i.u{+}{+}H(t_i)$. Note the "++" operator is used for concatenation.

[0044] It is possible for multiple chains to coexist. The present invention's chains can be differentiated by their block zero b_0 and b'_0 (made unique by their UUIDs) associated with a specific Sequencer Service instances identified by their unique public keys K_S^+ and $K_{S'}^+$ generated during the startup of S and S', respectively. Just as users can trust the correctness guarantees of one chain, so can multiple chains trust each other's correctness guarantees; however, the total order of transactions is maintained within a chain, but not across chains.

[0045] The structure of the present invention guarantees immutability, agreement, and availability. Each of these guarantees is described more fully below, following which is a discussion of an implementation of the present invention.

[0046] 1. Immutability

[0047] The present invention guarantees immutability by the alternating structure of blocks and timestamp attestations that come together like the teeth of zipper. A timestamp attestation provides a third-party trusted signature over the hash of the block, which includes the Merkle root constructed from a set of transaction data, also referred to as transactions. Any change to these transactions would be detectable as a mismatch between the block hash and the bytes signed by the timestamp attestation. Thus, as long as a timestamp attestation remains a part of a chain, its block remains immutable.

[0048] A timestamp attestation remains in the chain because the following block includes its hash. Thus, an attestation at the at end of a chain signs its block and, transitively, the previous attestation and, indirectly, its block, and so on.

[0049] A user that considers a block as belonging to an instance of the described blockchain can be sure of the block's integrity by verifying its timestamp attestation. The user can also check the integrity of the chain by verifying the preceding blocks and attestations all the way to a well-known block zero for a particular chain instance.

[0050] 2. Agreement

[0051] The present invention guarantees agreement by detecting and eliminating chain forks so that all users see the same totally ordered set of blocks and, by extension, transactions. A fork is defined as the existence of two blocks b_i and b'_i with the same block height i>0 and the same previous block b_{i-1}. FIG. **4** illustrates a fork in a chain produced by the present invention, where blocks b_i and b'_i point to the same previous timestamp attestation t_{i-1} and transitively block b_{i-1}.

[0052] Forks are problematic in blockchains because they create the possibility of an inconsistency of application state represented on the blockchain. For example, assume two users submit transaction data d and d' that are incompatible with each other. If there is no fork in the chain, any client reading the blockchain sees the same consistent history in which, say, d precedes d' in the same block, or across different blocks. Then, according to the rules of an application, the transaction d may be applied to the state and d' may be ignored. If, on the other hand, d and d' are in the forked blocks b_i and b'_i it is not clear how to order d and d' to determine which transaction to apply and which to ignore.

[0053] Forks also create another problem unique to the present invention. A user verifying the integrity of an instance of the described blockchain by walking through it backwards from a given block, may not know whether the encountered blocks are on the main chain, or on a fork. This uncertainty is problematic because transactions in blocks on a fork will not be considered valid by the users following the main chain. Additionally, the forked blocks may be maliciously deleted without users on the main chain detecting that deletion.

[0054] The present invention detects forks by using block heights and sequence attestations to create a total order of all bloc (those on the main chain and on all forks). Given two tuples $\langle b,t,a \rangle$ and $\langle b',t',a' \rangle$ of corresponding blocks, timestamp, and enclave attestations (containing sequence attestations) such that a.r.y=bμ++H(t) and a'.r.y=b'.u++H(t'), an order relation (→) is defined as follows:

$$(b,a) \rightarrow (b',a') \Leftrightarrow b.i < b'.i \vee (b.i = b'.i \wedge a.r.k < a'.r.k).$$

[0055] For example, the total order of the pairs of blocks and enclave attestations in FIG. **4** is $(b_{i-1},a_{k-1}) \rightarrow (b_i,a_k) \rightarrow (b'_i,a_{k+1}) \rightarrow (b'_{i+1},a_{k+2}) \rightarrow (b_{i+1},a_{k+3})$. When there is a fork, the user determines which block is on the main chain using two rules. First, a block may be on the main chain only if its parent is on the main chain. Second, if multiple blocks have a parent on the main chain, the lowest order block is on the main chain. For example, following FIG. **4**, assume that b_{i-1} is on the main chain and the fork starts at block height i. Given, (b_i,a_k) and (b'_i,a_{k+1}) at the start of the fork, the user determines which block is on the main chain by observing that $(b_i,a_k) \rightarrow (b'_i,a_{k+1})$, which implies (deterministically) that b_i is on the main chain, while b'_i is not. Similarly, given that b_i is on the main chain, users can deterministically decide that b_{i+1} is on the main chain even though $(b'_{i+1},a_{k+2}) \rightarrow (b_{i+1},a_{k+3})$ because b_{i+1}'s predecessor is on the main chain, while the predecessor of b'_{i+1} is not.

[0056] 3. Availability

[0057] The present invention provides strong, probabilistic guarantees on the availability of data by using a trusted Replication Service to increase the distribution of blocks, timestamp attestations, sequencer attestations, and the leaves of the Merkle trees. Data replication with a trusted Replication Service creates redundant shards of data distributed among independently failing replicas. As a consequence, the encoded data remains available to users even if some of the replicas become unavailable. Even in the case where a sufficient number of replicas is not momentarily available, users remain confident that the stored data remains intact, because of the high durability guarantees of a trusted Replication Service, and will become available again. It is important to note that the present invention records transaction data in a manner that allows users to verify immutability and agreement over blockchain transactions as long as the guarantees of a trusted Replication Service remain in place, even if the other trusted services, or the blockchain creation mechanism, become unavailable.

[0058] 4. Implementation

[0059] The present invention provides the following abstract interface:

$$\text{write}(d)$$

$$f := \text{verify}(b_0, K_S^+, K_T^+, H(d))$$

The write function takes the transaction data d as input and starts the process of recording d on an instance of the described blockchain invention. The verify function takes the chain with genesis block b_0, the public key of the Sequence Service K_S^+, the public key of the Timestamp Service K_T^+, and the hash of a transaction H(d) as input to return a certificate f, which confirms that the transaction exists in a finalized block on a chain with genesis block b_0. The certificate contains the transaction hash d_H, two timestamps $\bar{t}$ and $\underline{t}$ representing the upper and lower time bounds tor transaction acceptance, and the chain fields b_0. When users trust the verify function they know that the transaction d was written onto a chain between $\bar{t}$ and $\underline{t}$ and will remain unchanged on that chain.

[0060] a. Recording a Transaction

[0061] FIG. **5** shows the process implementing the write function in the present invention. The numbering of the process description below corresponds to the arrow numbering in the figure.

[0062] 1. To sign a transaction d, a user calls write(d). The BatchIt service receives d and enqueues the write request internally.

[0063] 2. Periodically, the ZipIt service asks BatchIt for a batch of write requests. BatchIt dequeues a set of requests and creates a batch identified by a batch ID c. BatchIt then creates a Merkle tree m, where each leaf is transaction data d from a write request in the batch. BatchIt then replies to ZipIt with the tuple $\langle c,m \rangle$.

[0064] 3. Upon receiving a batch from BatchIt, the ZipIt service creates a block $b = \langle u, m^H, t^H \rangle$, where u is a freshly generated UUID, $m_H = H(m)$ is the root of the Merkle tree, and $t^H = H(t_{i-1})$ is the hash of the preceding timestamp attestation. The starting point of the blockchain is a well-known block zero b_0 and its corresponding timestamp attestation t_0, and so ZipIt always has a t_{i-1} to include in a block. Next. ZipIt invokes the timestamp function of a trusted Timestamp Service by passing it the hash of the block H(b). The attestation service returns a timestamp attestation $t_i = \langle y,p,g \rangle$, where y=H(b) and $g = \{H(y,p)\}_{K_T^-}$. Finally, ZipIt saves t internally as t_{i-1} to include its hash in the next block.

[0065] 4. ZipIt sends the tuple $\langle c,m,b,t \rangle$ to the StoreIt service.

[0066] 5. StoreIt sends the tuple $\langle m,b,t \rangle$ for replication to the Replication Service. StoreIt does not move onto the next step until the Replication Service confirms that the tuple has been successfully replicated.

[0067] 6. Once a block has been replicated it is safe to assign it a sequence number. StoreIt send the unique ID of the block b.u concatenated with the hash of the timestamp t attestation H(t) to the Sequencer Service, which replies with an enclave attestation a containing the sequence attestation $s = \langle b.u{+}{+}H(t),k,g \rangle$, where $g = \{H(b.u{+}{+}H(t),k)\}_{K_S^-}$.

[0068] 7. The StoreIt service passes the enclave attestation a to the Replication Service for replication. Upon completing the replication of the enclave attestation, all transactions in the block are finalized.

[0069] 8. Finally. StoreIt sends the batch id c to BatchIt to notify it that a block corresponding to a batch of requests has been replicated, which allows BatchIt to delete the batch. It is important to note that the write function does not guarantee that a transaction has been recorded on the blockchain. The write function returns after step 1 notifying the user that the transaction has been accepted for processing. The user knows that a transaction has been recorded on the blockchain only after the verify function returns a certificate. The user can expect that verity will produce the correct result on a transaction only after the transaction is finalized.

[0070] b. Verifying a Transaction

[0071] After calling the write function on a transaction, the user needs to verify that the transaction was written onto the blockchain. i.e., that the transaction exists in a finalized block on the main chain. The verification process proceeds as follows:

[0072] 1. To verify a transaction, a user calls verify(b_0, $\mathcal{K}_S^+$,$\mathcal{K}_T^+$,H(d)) by passing in transaction data d and the information identifying the blockchain, namely, a block zero b_0, along with the and the public key of a Sequencer Service K_S^+ and the public key $\mathcal{K}_T^+$ of the Timestamp Service. As mentioned earlier, the verify function can execute on the user's machine; for this reason, the user does not need to trust the organization running the present invention to execute the function correctly.

[0073] 2. The verify function downloads the current state of the blockchain from the Replication Service. Specifically, this state comprises the set of blocks (B), Merkle trees (M), timestamp attestations (T), and enclave attestations (A). In practice, these sets and their verification as described below may be cached (bootstrapped) and extended as the chain grows.

[0074] 3. Next, the verify function determines which blocks and timestamp attestations are on the main chain. To do so, verify calls the function omc, shown in FIG. **6**, which produces a set of alternating main chain blocks and timestamp attestations Z=omc(b_0,$\mathcal{K}_S^+$,$\mathcal{K}_T^+$,B,T,A).

[0075] 4. Finally, the verify function determines whether any main chain block contains the transaction data d. It is assumed that transactions are idempotent and their hashes unique; therefore, the certificate of a transaction always pertains to its first instance. To create a certificate, verify calls the makeCertificate function, shown in FIG. **7**, which produces a certificate f=makeCertificate(b_0,$\mathcal{K}_S^+$,H (d),M,Z). If f≠ø, verify returns the certificate to the user.

[0076] When the verify function returns a certificate f, the function asserts that the transaction data d existed before the time $f.\bar{t}$ and after time $f.\underline{t}$ on the chain with genesis block b_0. When verify returns a certificate it also indicates that the transaction is final and will not change on the blockchain.

[0077] Let f and f' be certificates over transaction data d and d', respectively, such that b is the block with hash $f.\underline{t}.b^H$ and b' is the block with hash $f'.\underline{t}.b^H$. Data d proceeds d' if and only if b.i<b'.i, or b.i=b'.i and d is to the left of d' in the leaves of the Merkle tree m, such that $H(m)=b.m^H$.

[0078] 5. Advantages Over Prior Art

[0079] Early blockchain designs, such as Bitcoin [18], were made possible by a Proof-of-Work (PoW) and the Nakamoto probabilistic consensus. A miner creates a new block by solving a cryptographic puzzle and guesses a nonce, the hash of which, together with other parts of the blockchain, produces a hash that is sufficiently small when considered as a binary number. While this mechanism has proven resilient to coordinated attacks, it is costly in terms of electricity used by mining hardware. To address the cost of mining of new blocks. Peercoin™ [19] was the first to adopt Proof-of-Stake (PoS), where the opportunity to create the next block is decided by a lottery weighted by the number of coins staked by a verifier node rather than the node's hash power. Often, correctness is enforced by a Byzantine Fault Tolerant (BFT) consensus mechanism. Both PoW and PoS designs have led to a number of well-established public blockchains [18], [20].

[0080] The limiting factor to the performance of these blockchains is the network performance between its miner/verifier nodes [20]. It simply takes some time to disseminate a new block, so that the verifiers can create its successor, rather than a fork. The block interval then is governed by the size of the block, block interval, and network performance. While one might naturally worry about block processing time, for example, in face of complex smart contracts, verifier processing speed has not been the limiting factor to blockchain performance as of yet [2], [4].

[0081] To gain higher performance DLT designs follow two directions. The first direction, primarily, reduces transaction recording delay though decreasing the block interval by using smaller blocks that take less time to disseminate. The second direction, primarily, increases transaction throughput by relying on a constrained number of well-connected verifier nodes. Some blockchains combine the two approaches [21], [22].

[0082] In the first direction, when decreasing block interval, blocks can become so small as to contain only a few transactions. But a block can reference multiple (typically two) previous blocks, which forms a directed acyclic graph (DAG)[1], [21]-[23]. In a DAG, blocks at the same height in the DLT do not necessarily create a fork since bifurcations can be merged in subsequent blocks. Transaction finality still depends on agreement among some quorum of nodes, but nodes reach agreement independently on each transaction. Independent agreement tends to speed up transaction delay, but not necessarily throughput.

[0083] In the second direction, when constraining the number of verifiers, speed of block dissemination improves though high capacity, direct connections between verifiers. A DLT may identify these verifiers through delegation [21] or through a technique called proof-of-authority (PoA) [24], [25]. In PoA, verifiers stake their reputation to follow the protocol. Not following the protocol forfeits reputation and with it, the right to make future blocks. Unfortunately, in most systems, it is not clear how to quantify reputation.

[0084] The present invention may be characterized as a "delegated" PoA, insofar as it transfers the reputation of dependent services, such as Amazon Cognito, Amazon S3, and AWS Nitro Enclaves into a blockchain mechanism. With delegation, there are multiple metrics to quantify reputation. Examples include users, revenue, or number of projects using the service. With delegated PoA, the key observation is that, independent of the present invention, dependent services are already staking their reputation. That is, if a service such as Amazon Cognito deviates from its protocol, its reputation will sink resulting in loss of users, revenue, etc.

[0085] The key feature of a delegated PoA blockchain resulting from the present invention is that its throughput depends on the performance of the network among its component services, as shown in FIG. **5**. Since the component services may be all placed in the cloud, their connections can approach line speeds, which is orders of magnitude faster than the interconnects of more distributed blockchain architectures. The result is orders of magnitude higher throughput for the present invention, subject to a constant transaction finality.

[0086] Although the preferred embodiment of the present invention has been shown and described, it will be apparent to those skilled in the art that many changes and modifications may be made without departing from the invention in its broader aspects. The appended claims are therefore intended to cover all such changes and modifications as fall within the true spirit and scope of the invention.

We claim:

1. A method for creating and maintaining immutability, agreement and availability of data comprising the steps of:

(a) building an alternating structure of blocks and timestamp attestations by:

(i) constructing a first Merkle tree having leaves of transaction data;

(ii) creating a first block that contains a root of the first Merkle tree;

(iii) using a trusted timestamp service to create a timestamp attestation over the first block;

(iv) constructing a second Merkle tree having leaves of transaction data;

(v) creating a second block that contains a root of the second Merkle tree;

(vi) linking the second block to the timestamp attestation of the first block;

(vii) using a trusted timestamp service to create a timestamp attestation over the second block; and

(viii) repeating steps (a)(i)-(a)(vii) over a series of blocks and timestamp attestations to create the alternating structure in which each block is linked to the timestamp attestation of an immediately preceding block;

(b) determining an order of blocks by:

(i) using a trusted sequencer service to assign a sequence attestation over each block and its timestamp attestation, wherein each sequence attestation has a unique number, and

(ii) wherein each block has a height creating a total order of the blocks based on the height of each block and the sequence attestation assigned to each block;

(c) determining which of the blocks are on a main chain by:

(i) checking validity of the timestamp attestation over the first block;

(ii) checking validity of the sequence attestation over the first block and of the timestamp attestation over the first block;

(iii) adding the first block and the timestamp attestation over the first block to the main chain;

(iv) wherein the main chain has a last block, wherein the last block has a sequence attestation, extending the main chain from the last block by:

(A) finding all successor blocks of the last block, wherein each successor block has a timestamp attestation and a sequence attestation;

(B) identifying a successor block with a sequence attestation that is lower than the sequence attestations of all other successor blocks;

(C) checking validity of the timestamp attestation and validity of the sequence attestation of the successor block identified in step (c)(iv)(2); and

(D) if all blocks with sequence attestations between the sequence attestation of the last block on the main chain and the sequence attestation of the successor block with the lowest sequence attestation can be found, adding to the main chain the successor block with the lowest sequence attestation and the timestamp attestation over the successor block with the lowest sequence attestation; and

(v) repeating steps (c)(iv)(1)-(c)(iv)(4) over a set of blocks, timestamp attestations, and sequence attestations; and

(d) using a trusted replication service to replicate all of the blocks, all of the timestamp attestations, all of the sequence attestations, and all of the leaves of the Merkle trees.

2. A computer program product comprising a storage device storing instructions in a non-transitory manner, which instructions, when executed by a processing unit of a computing device, cause the computing device to:

create and maintain immutability, agreement and availability of data, wherein to create and maintain immutability, agreement and availability of data comprises:

(a) building an alternating structure of blocks and timestamp attestations by:

(i) constructing a first Merkle tree having leaves of transaction data;

(ii) creating a first block that contains a root of the first Merkle tree;

(iii) using a trusted timestamp service to create a timestamp attestation over the first block;

(iv) constructing a second Merkle tree having leaves of transaction data;

(v) creating a second block that contains a root of the second Merkle tree;

(vi) linking the second block to the timestamp attestation of the first block;

(vii) using a trusted timestamp service to create a timestamp attestation over the second block; and

(viii) repeating steps (a)(i)-(a)(vii) over a series of blocks and timestamp attestations to create the alternating structure in which each block is linked to the timestamp attestation of an immediately preceding block;

(b) determining an order of blocks by:

(i) using a trusted sequencer service to assign a sequence attestation over each block and its timestamp attestation, wherein each sequence attestation has a unique number; and

(ii) wherein each block has a height, creating a total order of the blocks based on the height of each block and the sequence attestation assigned to each block;

(c) determining which of the blocks are on a main chain by:

(i) checking validity of the timestamp attestation over the first block;

(ii) checking validity of the sequence attestation over the first block and of the timestamp attestation over the first block;

(iii) adding the first block and the timestamp attestation over the first block to the main chain;

(iv) wherein the main chain has a last block, wherein the last block has a sequence attestation, extending the main chain from the last block by:

(A) finding all successor blocks of the last block, wherein each successor block has a timestamp attestation and a sequence attestation;

(B) identifying a successor block with a sequence attestation that is lower than the sequence attestations of all other successor blocks;

(C) checking validity of the timestamp attestation and validity of the sequence attestation of the successor block identified in step (c)(iv)(2); and

(D) if all blocks with sequence attestations between the sequence attestation of the last block on the main chain and the sequence attestation of the successor block with the lowest sequence attestation can be found, adding to the main chain the successor block with the lowest sequence attestation and the timestamp attestation over the successor block with the lowest sequence attestation; and

(v) repeating steps (c)(iv)(l)-(c)(iv)(4) over a set of blocks, timestamp attestations, and sequence attestations; and

(d) using a trusted replication service to replicate all of the blocks, all of the timestamp attestations, all of the sequence attestations, and all of the leaves of the Merkle trees.

3. A computing device comprising a processing unit, memory or other storage device coupled to the processing unit, the memory or other storage device storage instructions, which, when executed by the processing unit, cause the computing device to:

create and maintain immutability, agreement and availability of data, wherein to create and maintain immutability, agreement and availability of data comprises:

(a) building an alternating structure of blocks and timestamp attestations by:

(i) constructing a first Merkle tree having leaves of transaction data;

(ii) creating a first block that contains a root of the first Merkle tree;

(iii) using a trusted timestamp service to create a timestamp attestation over the first block;

(iv) constructing a second Merkle tree having leaves of transaction data;

(v) creating a second block that contains a root of the second Merkle tree;

(vi) linking the second block to the timestamp attestation of the first block;

(vii) using a trusted timestamp service to create a timestamp attestation over the second block; and

(viii) repeating steps (a)(i)-(a)(vii) over a series of blocks and timestamp attestations to create the alternating structure in which each block is linked to the timestamp attestation of an immediately preceding block;

(b) determining an order of blocks by:

(i) using a trusted sequencer service to assign a sequence attestation over each block and its timestamp attestation, wherein each sequence attestation has a unique number; and

(ii) wherein each block has a height, creating a total order of the blocks based on the height of each block and the sequence attestation assigned to each block;

(c) determining which of the blocks are on a main chain by:

(i) checking validity of the timestamp attestation over the first block;

(ii) checking validity of the sequence attestation over the first block and of the timestamp attestation over the first block;

(iii) adding the first block and the timestamp attestation over the first block to the main chain;

(iv) wherein the main chain has a last block, wherein the last block has a sequence attestation, extending the main chain from the last block by:

(A) finding all successor blocks of the last block, wherein each successor block has a timestamp attestation and a sequence attestation;

(B) identifying a successor block with a sequence attestation that is lower than the sequence attestations of all other successor blocks;

(C) checking validity of the timestamp attestation and validity of the sequence attestation of the successor block identified in step (c)(iv)(2); and

(D) if all blocks with sequence attestations between the sequence attestation of the last block on the main chain and the sequence attestation of the successor block with the lowest sequence attestation can be found, adding to the main chain the successor block with the lowest sequence attestation and the timestamp attestation over the successor block with the lowest sequence attestation; and

(v) repeating steps (c)(iv)(1)-(c)(iv)(4) over a set of blocks, timestamp attestations, and sequence attestations; and

(d) using a trusted replication service to replicate all of the blocks, all of the timestamp attestations, all of the sequence attestations, and all of the leaves of the Merkle trees.

* * * * *