



US 20250260577A1

(19) **United States**(12) **Patent Application Publication**
Greco et al.(10) **Pub. No.: US 2025/0260577 A1**(43) **Pub. Date: Aug. 14, 2025**(54) **PROTOCOLS FOR DECENTRALIZED NETWORKS**(71) Applicant: **Protocol Labs, Inc.**, Wilmington, DE (US)(72) Inventors: **Nicola Greco**, Cambridge, MA (US);
Juan Batiz-Benet, Walnut, CA (US)(21) Appl. No.: **19/169,976**(22) Filed: **Apr. 3, 2025****Related U.S. Application Data**

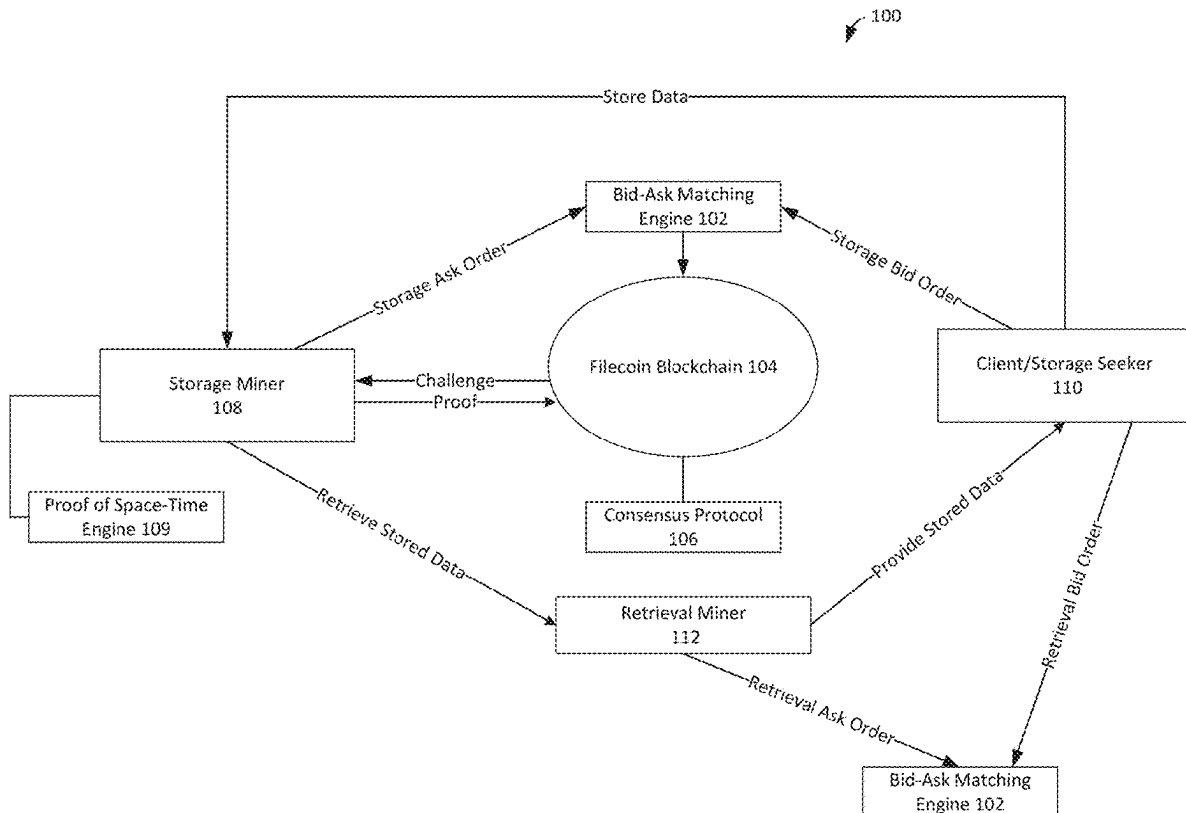
- (63) Continuation of application No. 18/075,794, filed on Dec. 6, 2022, now Pat. No. 12,294,654, which is a continuation of application No. 16/866,370, filed on May 4, 2020, now Pat. No. 11,570,001, which is a continuation-in-part of application No. 16/195,536, filed on Nov. 19, 2018, now Pat. No. 10,708,071, said application No. 18/075,794 is a continuation-in-part of application No. 16/726,520, filed on Dec. 24, 2019, now Pat. No. 11,245,528, which is a continuation-in-part of application No. 16/195,621, filed on Nov. 19, 2018, now Pat. No. 10,554,407.
- (60) Provisional application No. 62/697,123, filed on Jul. 12, 2018, provisional application No. 62/697,091, filed on Jul. 12, 2018, provisional application No. 62/697,097, filed on Jul. 12, 2018, provisional application No. 62/901,207, filed on Sep. 16, 2019.

Publication Classification(51) **Int. Cl.****H04L 9/32** (2006.01)**H04L 9/00** (2022.01)**H04L 9/06** (2006.01)(52) **U.S. Cl.**CPC **H04L 9/3218** (2013.01); **H04L 9/0637** (2013.01); **H04L 9/3213** (2013.01); **H04L 9/3271** (2013.01); **H04L 9/50** (2022.05)

(57)

ABSTRACT

Methods, systems, and apparatus, including computer programs encoded on computer storage media, for establishing a proof of storage over a specified period of time. One of the methods includes: (i) receiving, at a prover, an initial input challenge; (ii) producing, at the prover, an output proof proving that the prover has access to specified data for a specified time, wherein producing the output proof includes: (a) producing, at the prover, an initial proof responsive to the initial input challenge, the initial proof proving that the prover is storing specified data; (b) generating, at the prover, a new input challenge based at least in part on the initial proof; (c) producing, at the prover, a new proof responsive to the new input challenge, the new proof proving that the prover is storing the specified data; and (d) repeating, at the prover, the generating step and the producing a new proof responsive to the new input challenge step a number of times, the repeating step generating sequential proofs of storage to determine time of storage, wherein each generating step is based at least in part on a most recent new proof; and (iii) forwarding the output proof, e.g., to a blockchain.



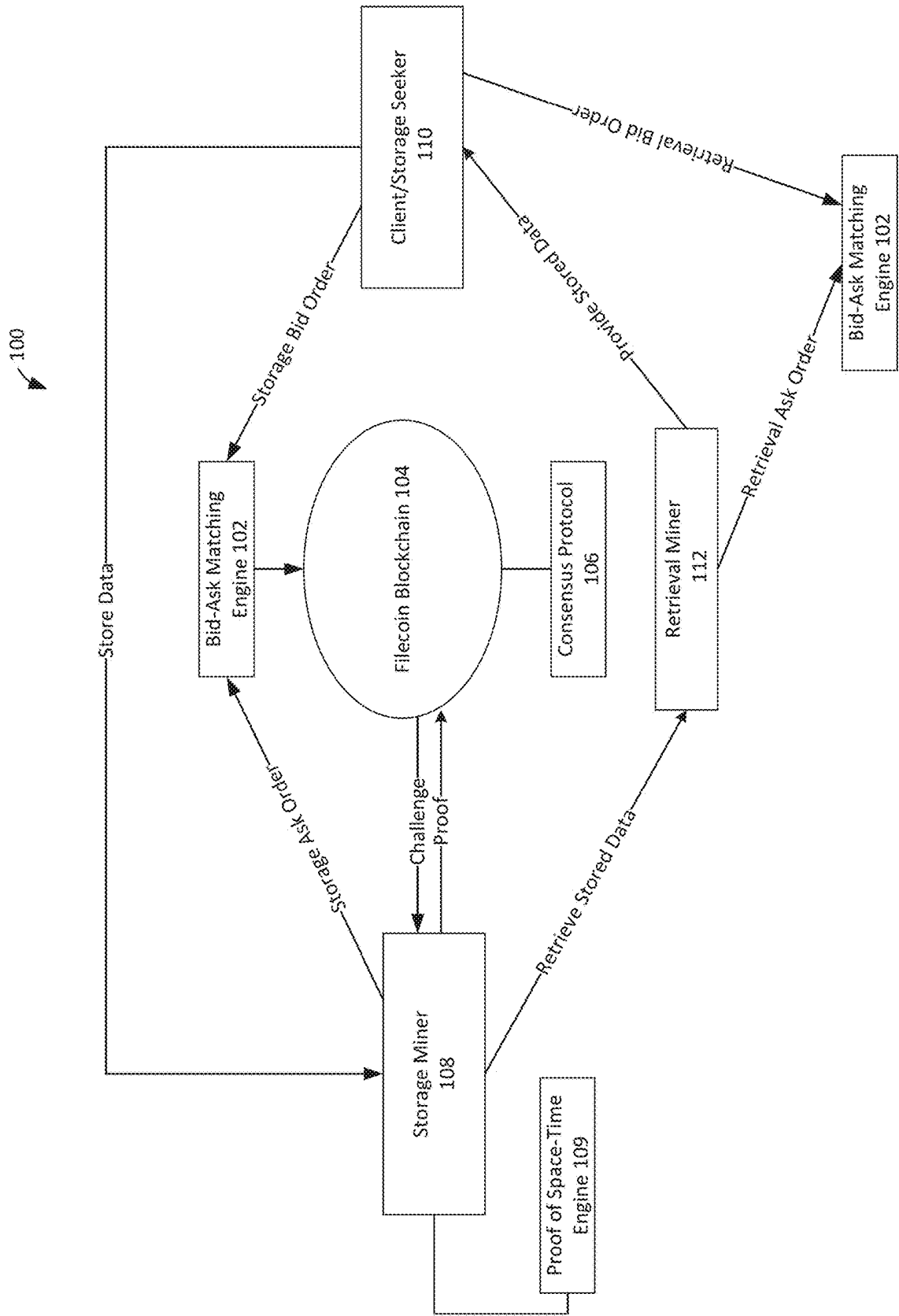


FIG. 1A

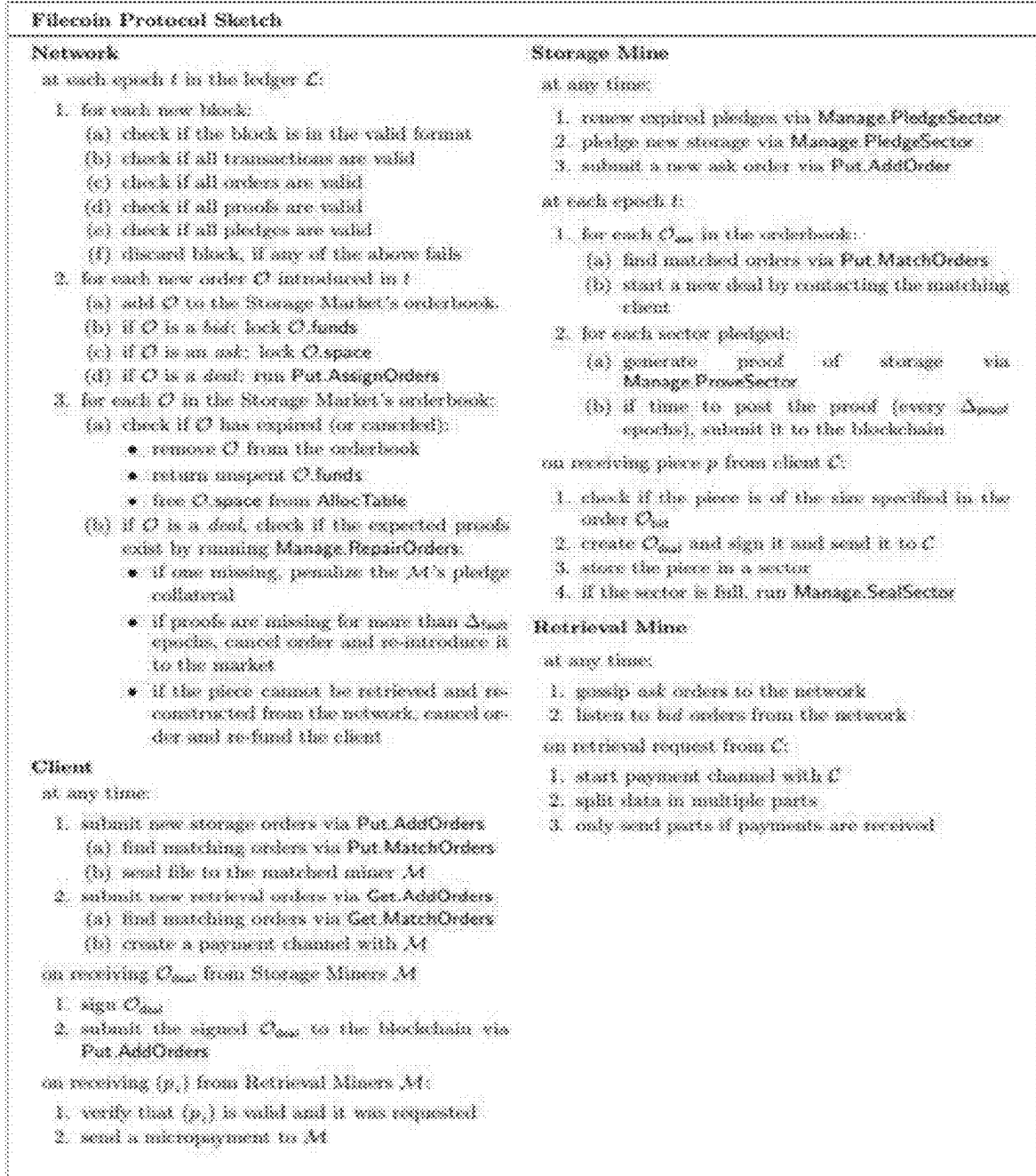


FIG. 1B

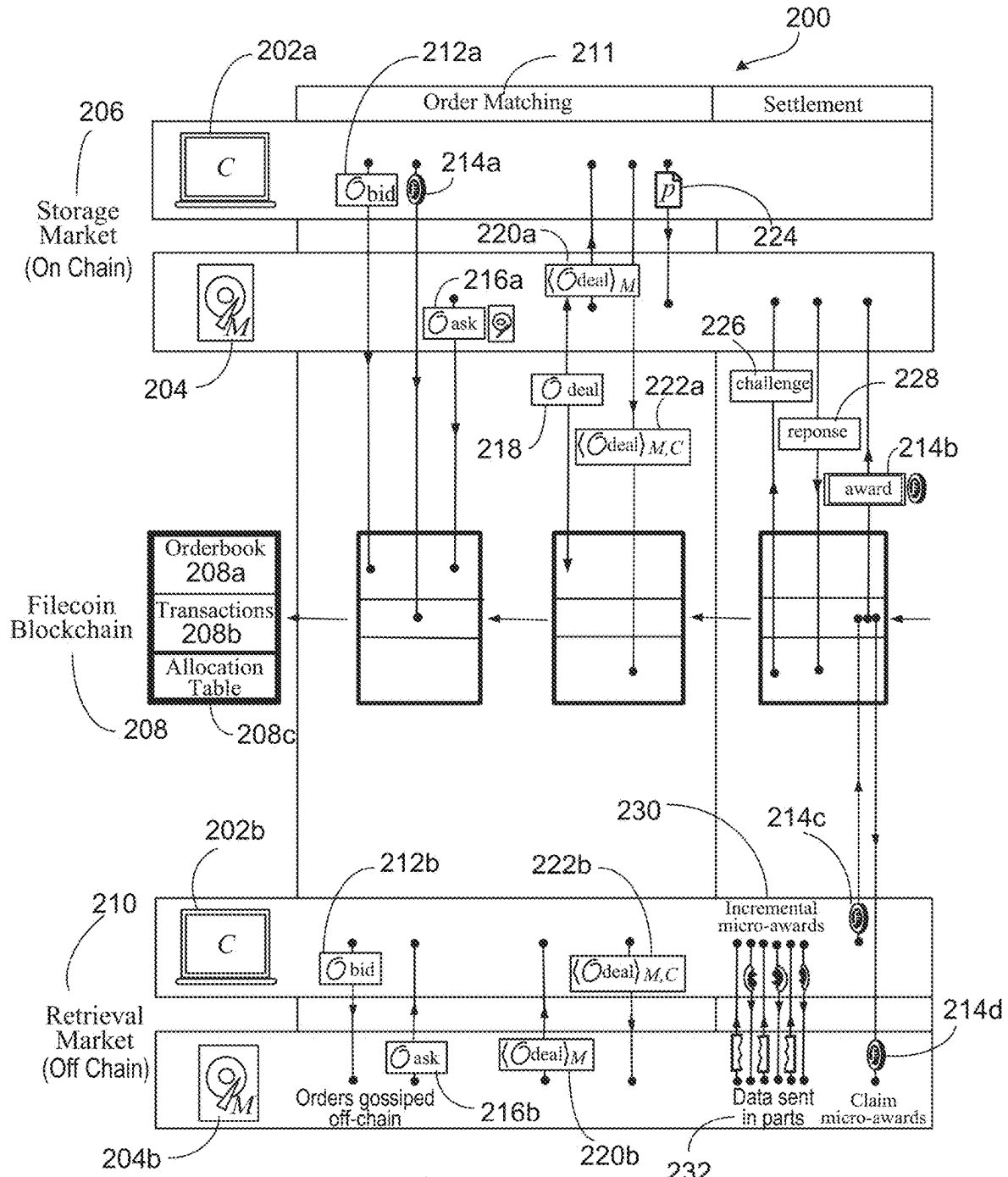


FIG. 2

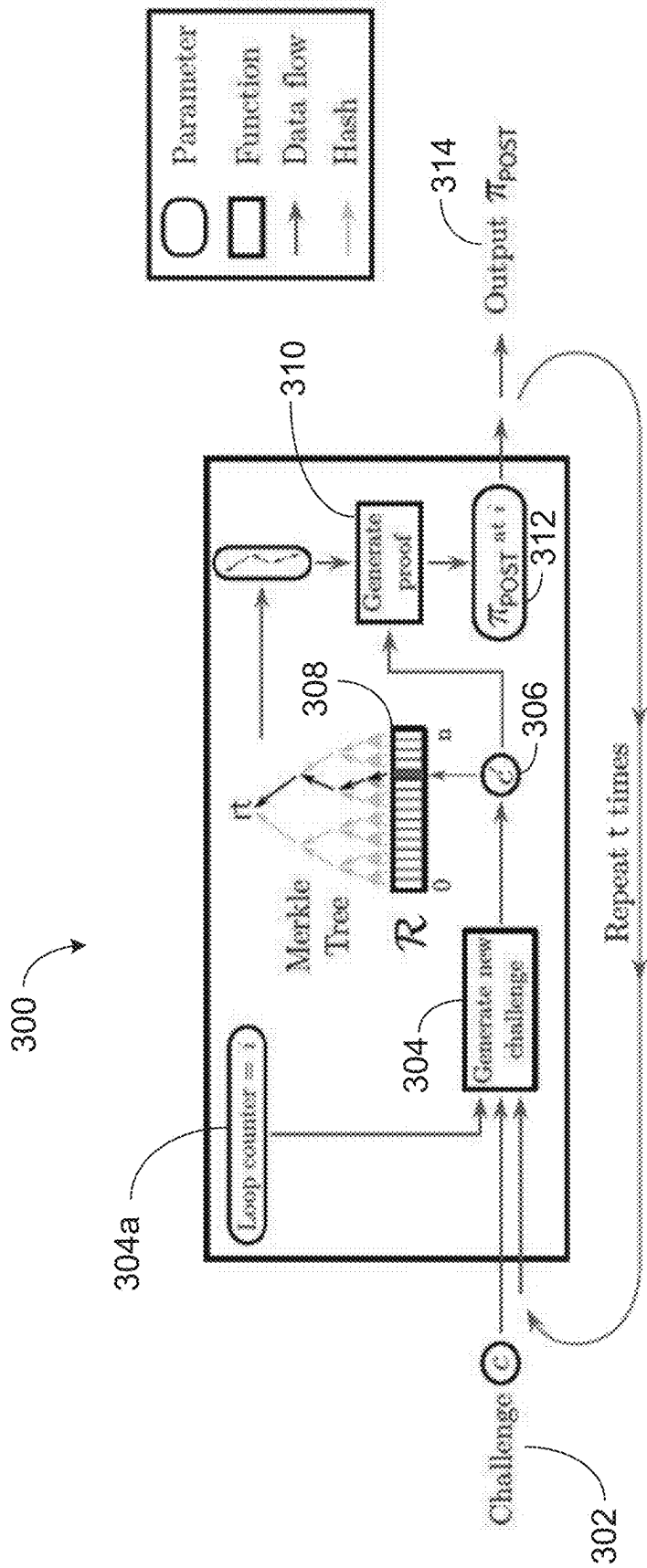


FIG. 3

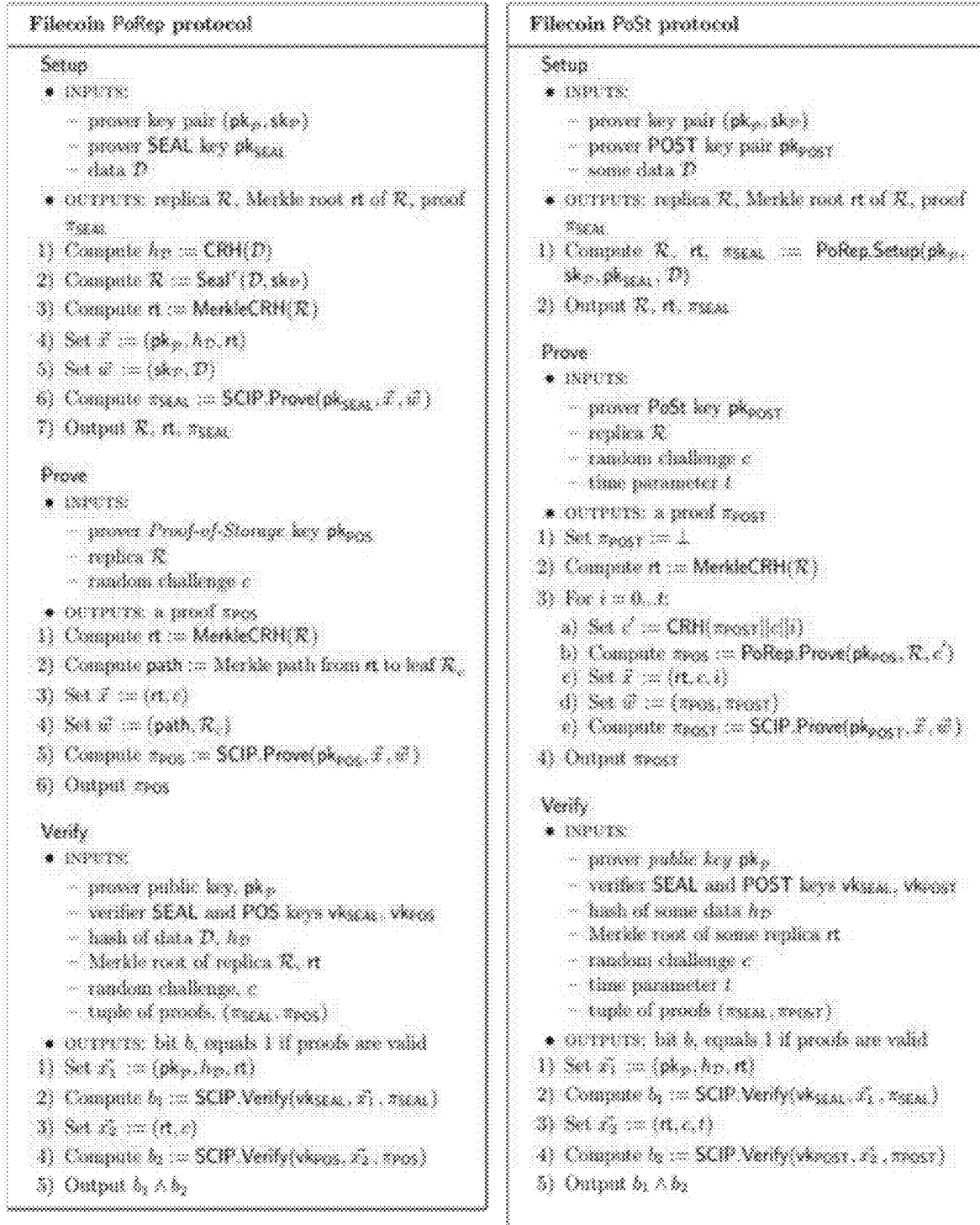


FIG. 4

Data Structures	
Pledge pledge := (size, coll) _{M_i} <ul style="list-style-type: none">• size, the size of the sector being pledged.• coll, the collateral specific to this pledge that M_i deposits.	Allocation allocTable: {M ₁ → (allocEntry..allocEntry), M ₂ ..} allocEntry: (sid, orders, last, missing) <ul style="list-style-type: none">• sid, sector id• Oⁱ, currently valid <i>deal</i>, <i>ask</i>, <i>bid</i> orders.• orders, set of orders {O_{deal}..O_{deal}}• last, last proof of storage in the ledger L• missing, counter for missing proofs
Orderbook OrderBook: (O ¹ ..O ⁿ) <ul style="list-style-type: none">• Oⁱ, currently valid <i>deal</i>, <i>ask</i>, <i>bid</i> orders.	

FIG. 5

Client	Network	Miner
AddOrders(...,O _{bid})	MatchOrders(..)	AddOrders(...,O _{ask})
SendPiece(...,O _{bid} , p)		ReceivePiece(...,O _{ask})
AddOrders(O _{deal})		AddOrders(...,O _{deal})
AddOrder(...,O _{bid})		AddOrder(...,O _{ask})
ReceivePiece(...,O _{bid})	MatchOrders(..)	SendPiece(...,O _{ask} , p)
AddOrders(...,O _{deal})		AddOrders(...,O _{deal})
Manage		
	AssignOrders(...,O _{deal})	PledgeSector()
		SealSector(..)
	RepairOrders(..)	ProveSector(..)

FIG. 6

Put Protocol	Get Protocol
<p>Market</p> <p>AddOrders</p> <ul style="list-style-type: none"> • INPUTS: list of orders O^1, \dots, O^n • OUTPUTS: bit b, equals 1 if successful <ol style="list-style-type: none"> 1) Set $\mathcal{O}_{order} := (O^1, \dots, O^n)$ 2) Submit \mathcal{O}_{order} to \mathcal{L} 3) Wait for \mathcal{O}_{order} to be included in \mathcal{L} 4) Output 1 on success, 0 otherwise <p>MatchOrders</p> <ul style="list-style-type: none"> • INPUTS: <ul style="list-style-type: none"> – the current Storage Market OrderBook – query order to match O^q • OUTPUTS: matching orders O^1, \dots, O^n <ol style="list-style-type: none"> 1) Match each O^q in OrderBook such that: <ol style="list-style-type: none"> a) If O^q is an ask order: <ol style="list-style-type: none"> i) Check if O^q is bid order ii) Check O^q price $\geq O^q$ price iii) Check O^q size $\leq O^q$ space b) If O^q is a bid order: <ol style="list-style-type: none"> i) Check if O^q is ask order ii) Check O^q price $\leq O^q$ price iii) Check O^q space $\geq O^q$ size 2) Output matched orders O^1, \dots, O^n <p>Exchange</p> <p>SendPiece</p> <ul style="list-style-type: none"> • INPUTS: <ul style="list-style-type: none"> – an ask order O_{ask} – a bid order O_{bid} – a piece p • OUTPUTS: a deal order O_{deal} signed by M_i <ol style="list-style-type: none"> 1) Get identity of M_i from O_{ask} signature 2) Send (O_{ask}, O_{bid}, p) to M_i 3) Receive O_{deal} signed by M_i 4) Check if O_{deal} is valid according to Definition 5) Output O_{deal} <p>ReceivePiece</p> <ul style="list-style-type: none"> • INPUTS: <ul style="list-style-type: none"> – signing key for M_i – current orderbook OrderBook – ask order O_{ask} – bid order O_{bid} – piece p • OUTPUTS: deal order O_{deal} signed by \mathcal{L} and M_j <ol style="list-style-type: none"> 1) Check if O_{bid} is valid: <ol style="list-style-type: none"> a) Check if O_{bid} is in OrderBook b) Check if O_{bid} is not referenced by other active O_{deal} c) Check if O_{bid} size is equal to p d) Check if O is signed by M_i 2) Store p locally 3) Set $O_{deal} := (O_{ask}, O_{bid}, H(p))M_i$ 4) Get identity of \mathcal{L} from O_{bid} 5) Send O_{deal} to \mathcal{L} 6) Output O_{deal} 	<p>Market</p> <p>AddOrders</p> <ul style="list-style-type: none"> • INPUTS: list of orders O^1, \dots, O^n • OUTPUTS: none <ol style="list-style-type: none"> 1) Gossip O^1, \dots, O^n to the network <p>MatchOrders</p> <ul style="list-style-type: none"> • INPUTS: <ul style="list-style-type: none"> – the current Borrowed Market OrderBook – query order to match O^q • OUTPUTS: matching orders O^1, \dots, O^n <ol style="list-style-type: none"> 1) Match each O^q in OrderBook such that: <ol style="list-style-type: none"> a) Check O^q piece is equal to O^q piece b) If O^q is an ask order: <ol style="list-style-type: none"> i) Check if O^q is bid order ii) Check O^q price $\geq O^q$ price c) If O^q is a bid order: <ol style="list-style-type: none"> i) Check if O^q is ask order ii) Check O^q price $\leq O^q$ price 2) Output matched orders O^1, \dots, O^n <p>Exchange</p> <p>SendPiece</p> <ul style="list-style-type: none"> • INPUTS: <ul style="list-style-type: none"> – an ask order O_{ask} – a bid order O_{bid} – a piece p • OUTPUTS: a deal order O_{deal} signed by \mathcal{L} <ol style="list-style-type: none"> 1) Create O_{deal}: <ol style="list-style-type: none"> a) Set $O_{deal}.ask := O_{ask}$ b) Set $O_{deal}.bid := O_{bid}$ 2) Get identity of \mathcal{L} from O_{bid} signature 3) Setup a micropayment channel with \mathcal{L} 4) For each block of data p_j of p: <ol style="list-style-type: none"> a) Set π_j to be a merkle path from $H(p)$ to p_j b) Send (O_{deal}, p_j, π_j) to \mathcal{L} c) Receive $(O_{deal}, j)C_i$ 5) Output O_{deal} <p>ReceivePiece</p> <ul style="list-style-type: none"> • INPUTS: <ul style="list-style-type: none"> – a client's key C_i – an ask order O_{ask} – a bid order O_{bid} – merkle tree hash of p in the orders h_p • OUTPUTS: a piece p <ol style="list-style-type: none"> 1) Create O_{deal}: <ol style="list-style-type: none"> a) Set $O_{deal}.ask := O_{ask}$ b) Set $O_{deal}.bid := O_{bid}$ 2) Get identity of M_i from O_{ask} signature 3) Set up a micropayment channel with M_i (or re-using an existing one) 4) When receiving (O_{deal}, p_j, π_j) from M_i: <ol style="list-style-type: none"> a) Check if O_{deal} is valid and matches O_{ask} and O_{bid} b) Check if π_j is a valid merkle-path with root hash h_p c) Send $(O_{deal}, j)C_i$ 5) Output p

FIG. 7

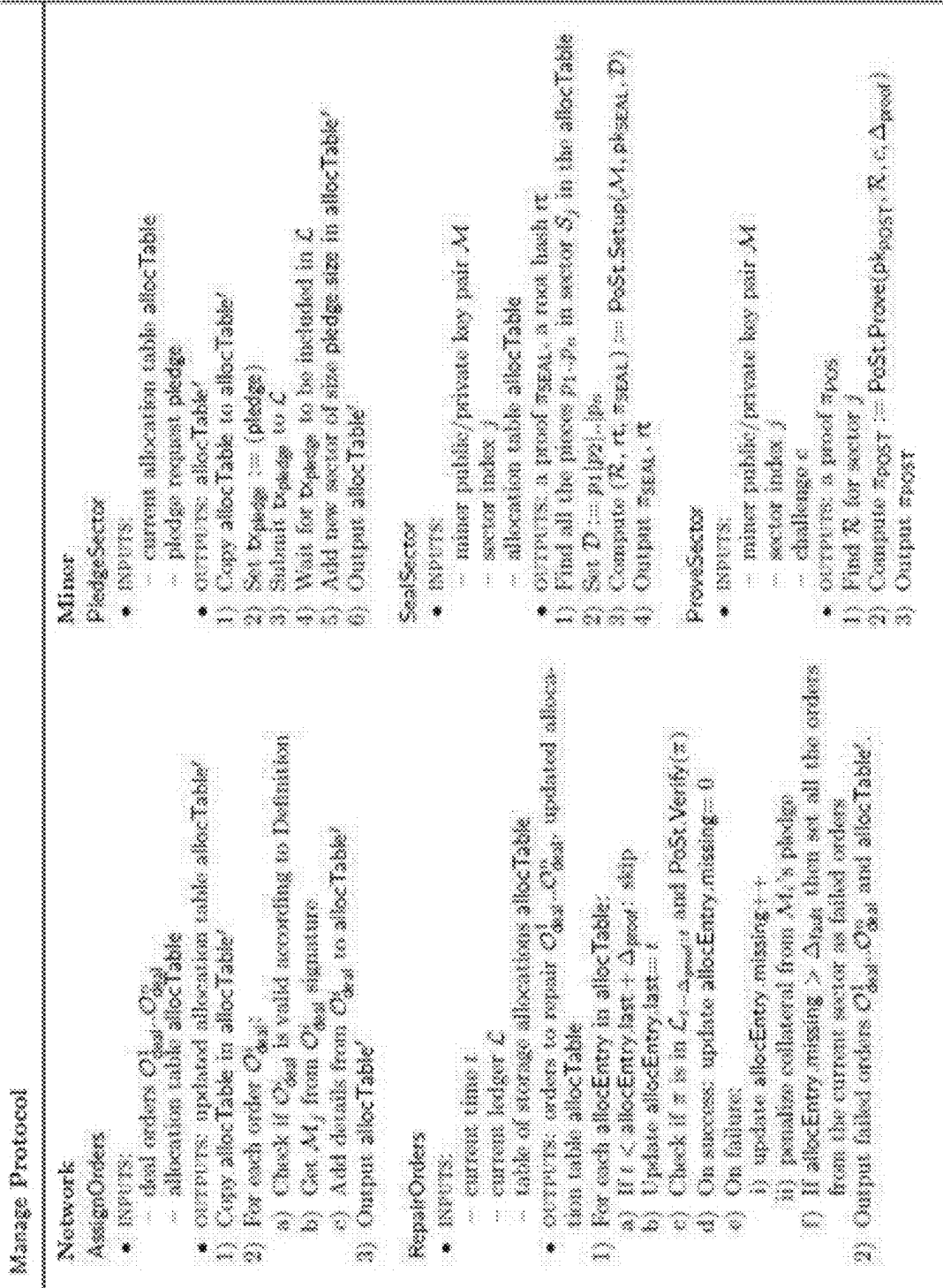


FIG. 8

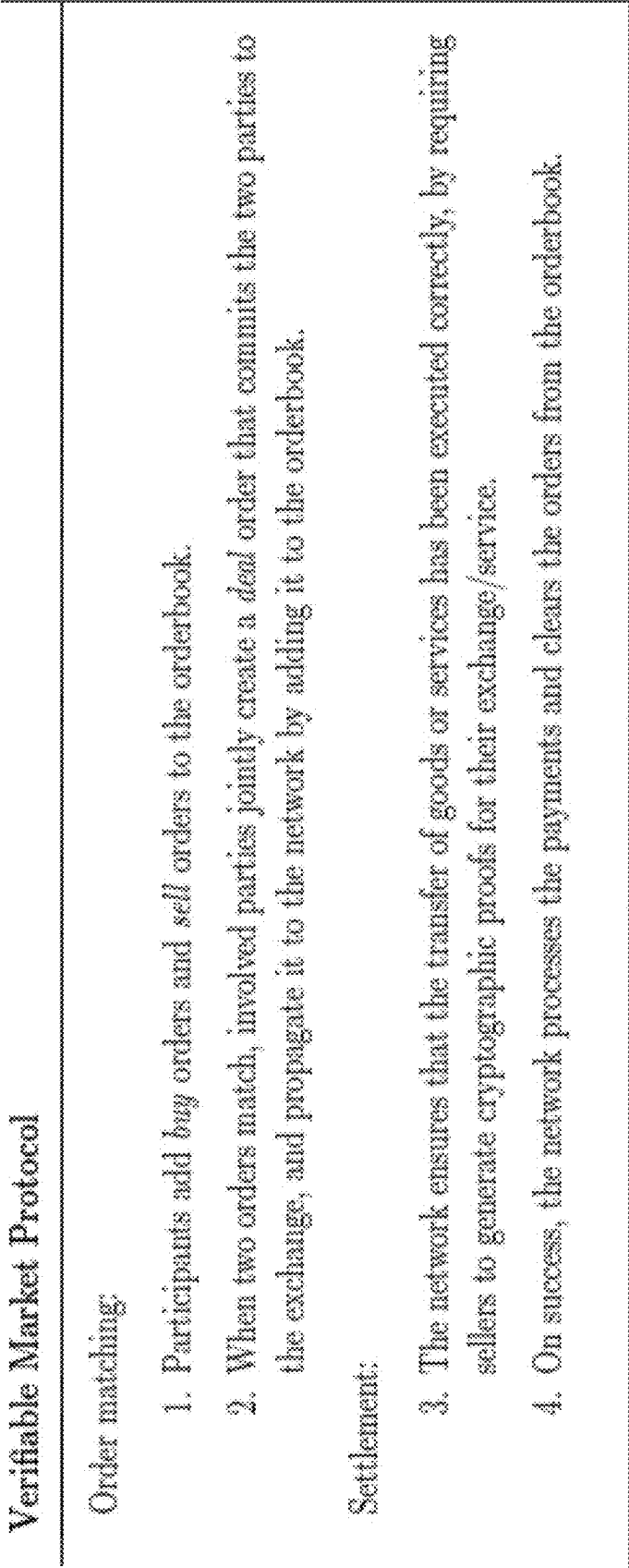


FIG. 9

Storage Market Orders	Retrieval Market Orders
<p><i>bid order</i></p> <p>$O_{bid} := (size, funds, price, time, coll, coding)_{C_i}$</p> <ul style="list-style-type: none"> • <i>size</i>, the size of the piece to be stored • <i>funds</i>, the total amount that client C_i is depositing • <i>time</i>, the maximum epoch time for which the file should be stored • <i>price</i>, the spacetime price in Filecoin • <i>coll</i>, the collateral specific to this piece that the miner is required to deposit • <i>coding</i>, the erasure coding scheme for this piece <p><i>ask order</i></p> <p>$O_{ask} := (space, price)_{M_i}$</p> <ul style="list-style-type: none"> • <i>space</i>, amount of space Storage Miner M_i is providing in the order • <i>price</i>, the spacetime price in Filecoin <p><i>deal order</i></p> <p>$O_{deal} := (ask, bid, ts, hash)_{C_i, M_j}$</p> <ul style="list-style-type: none"> • <i>ask</i>, a cryptographic reference to O_{ask} from C_i • <i>order</i>, a cryptographic reference to O_{bid} from M_i • <i>ts</i>, timestamp epoch in which the order has been signed by M_i • <i>hash</i> cryptographic hash of the piece that M_j will store <p><small>*If not specified, the piece will be stored until expiration of funds.</small></p> <p><small>*If not specified, when a Storage Miner is faulty, the network can re-introduce the order at the current best price.</small></p>	<p><i>bid order</i></p> <p>$O_{bid} := (piece, price)_{C_i}$</p> <ul style="list-style-type: none"> • <i>piece</i>, the index of the piece requested • <i>price</i>, the price at which C_i is paying for one retrieval <p><i>ask order</i></p> <p>$O_{ask} := (piece, price)_{M_i}$</p> <ul style="list-style-type: none"> • <i>piece</i>, the index of the piece requested • <i>price</i>, the price at which M_j is serving the piece for <p><i>deal order</i></p> <p>$O_{deal} := (ask, order)_{C_i, M_j}$</p> <ul style="list-style-type: none"> • <i>ask</i>, a cryptographic reference to O_{ask} from C_i • <i>order</i>, a cryptographic reference to O_{ask} from C_i <p><small>*Only pieces stored in Filecoin can be requested</small></p>

FIG. 10

Storage Market Protocol

Order Matching

1. Storage Miner M_i and Client C_j add orders to the OrderBook:
 - (a) M_i creates $O_{ask}^1, O_{ask}^2, \dots$ and C_j creates $O_{bid}^1, O_{bid}^2, \dots$
 - (b) Orders are submitted to the blockchain via $Put.addOrders(O^1, O^2, \dots)$
 - (c) On success, the orders are added to the OrderBook, the funds from C_j are deposited and the space from M_i is reserved.
2. When orders match, involved parties jointly create O_{deal} and add it to the OrderBook:
 - (a) M_i and C_j independently query the OrderBook via $Put.matchOrders(O)$.
 - (b) If M_i and C_j have matching orders :
 - C_j sends the piece p to M_i via $Put.SendPiece(O_{bid}, O_{ask}, p)$
 - M_i receives the piece p from C_j via $Put.ReceivePiece(O_{bid}, O_{ask}, p)$.
 - M_i signs O_{deal} and sends it to C_j
 - (c) C_j signs O_{deal} and adds it to the OrderBook via $Put.addOrders(O_{deal})$

Settlement

3. The Network checks if the Storage Miners are correctly storing the pieces:
 - (a) When a Storage Miner fills a sector, they seal it (they create a unique replica) via $Manage.SealSector$ and submit the proof τ_{SEAL} and π to the blockchain.
 - (b) Storage Miners generate new proofs at every epoch and add them to the Filecoin blockchain every Δ_{proof} epochs via $Manage.ProveSectors$.
 - (c) The Network runs $Manage.RepairOrders$ at every epoch. If proofs are missing or invalid, the network tries to repair in the following ways:
 - if any proofs are missing or invalid, it penalizes the Storage Miners by taking part of their collateral,
 - if a large amount of proofs are missing or invalid for more than Δ_{fault} epochs, it considers the Storage Miner *faulty*, settles the order as failed and reintroduces a new order for the same piece into the the market,
 - if every Storage Miner storing this piece is faulty, then the piece is lost and the client gets refunded.
4. When the time of the order is expired or funds run out, if the service was correctly provided, the Network processes the payments, and removes the orders.

FIG. 11

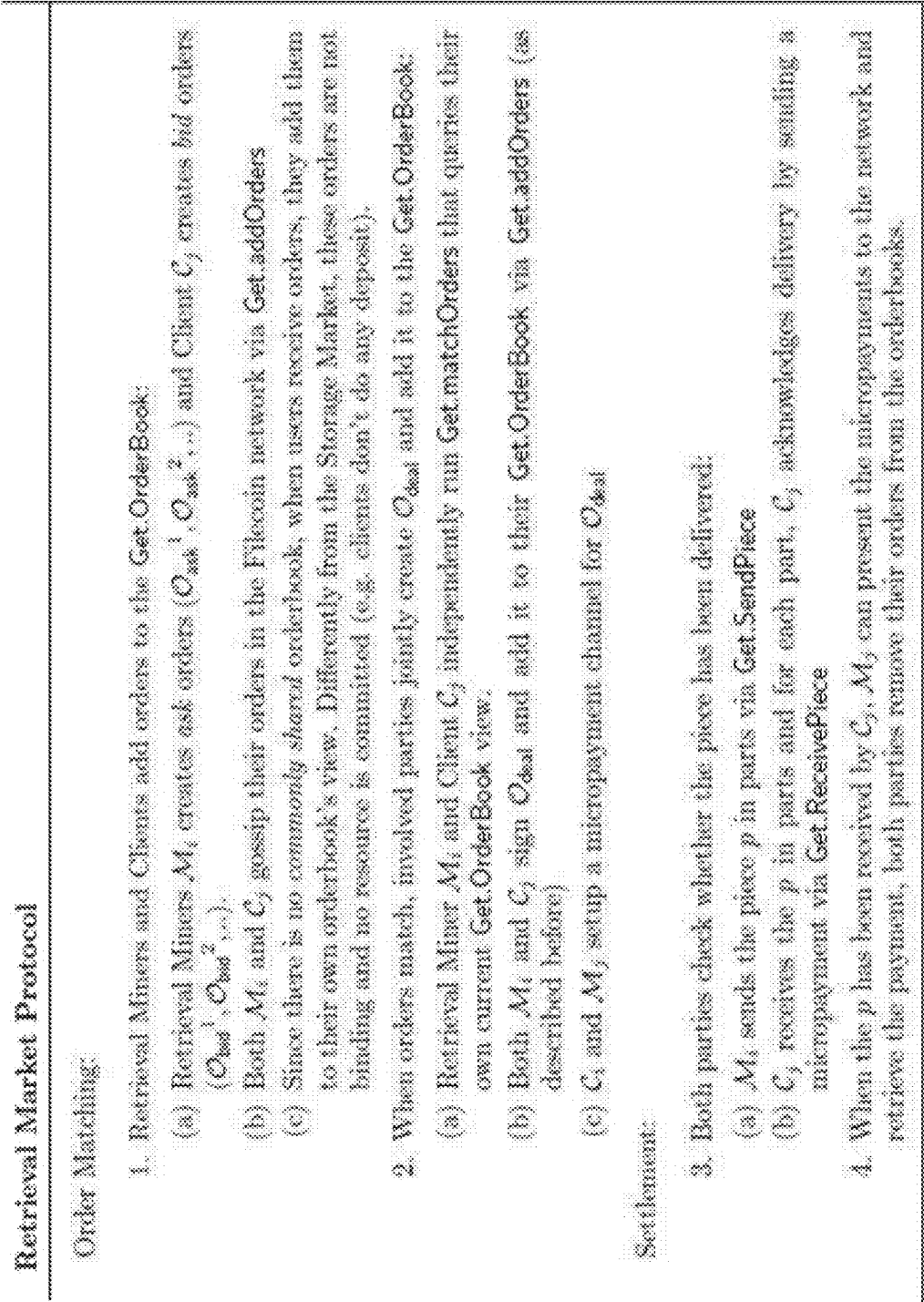


FIG. 12

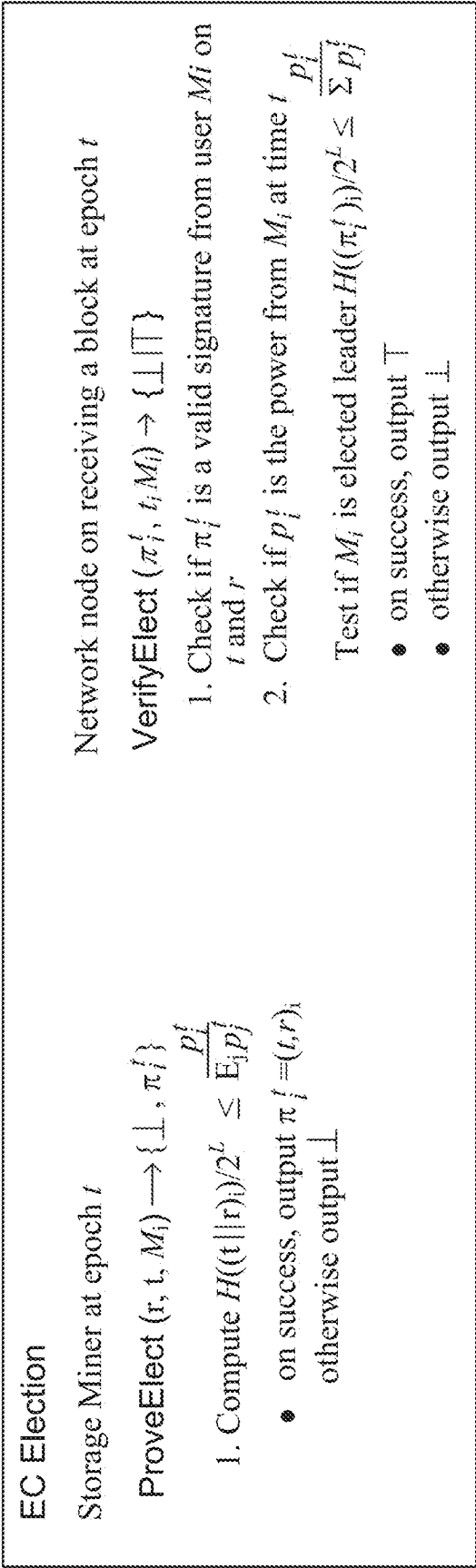


FIG. 13

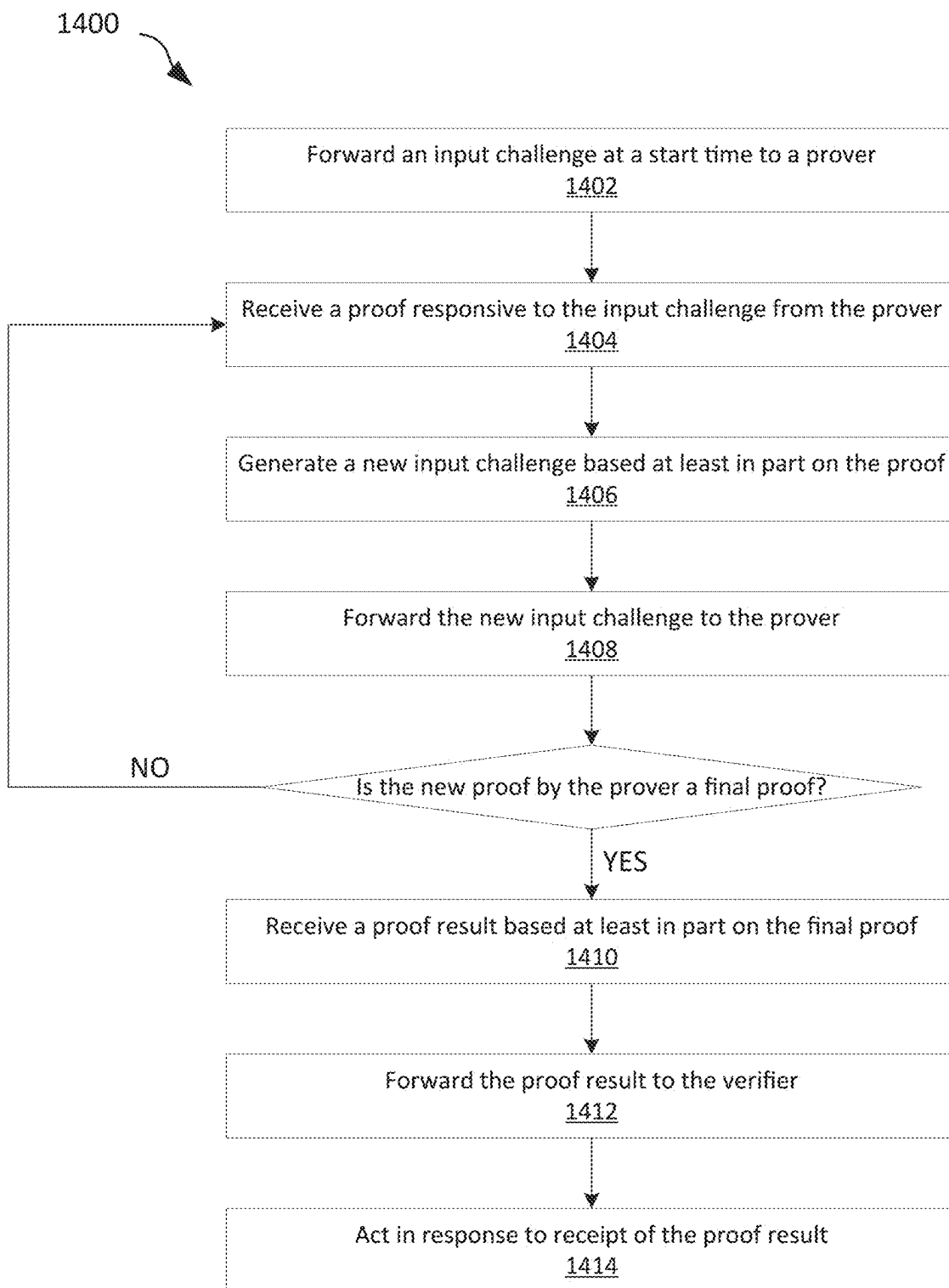


FIG. 14

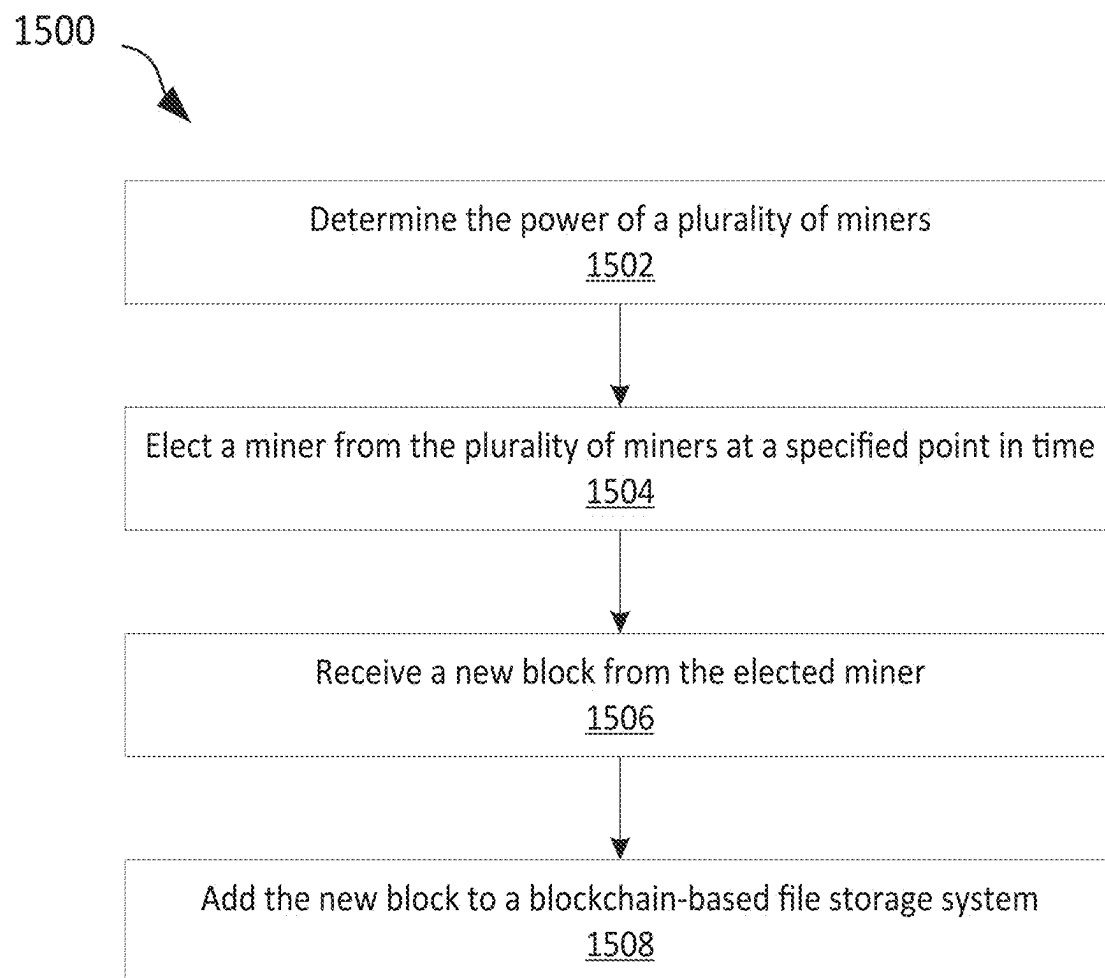


FIG. 15

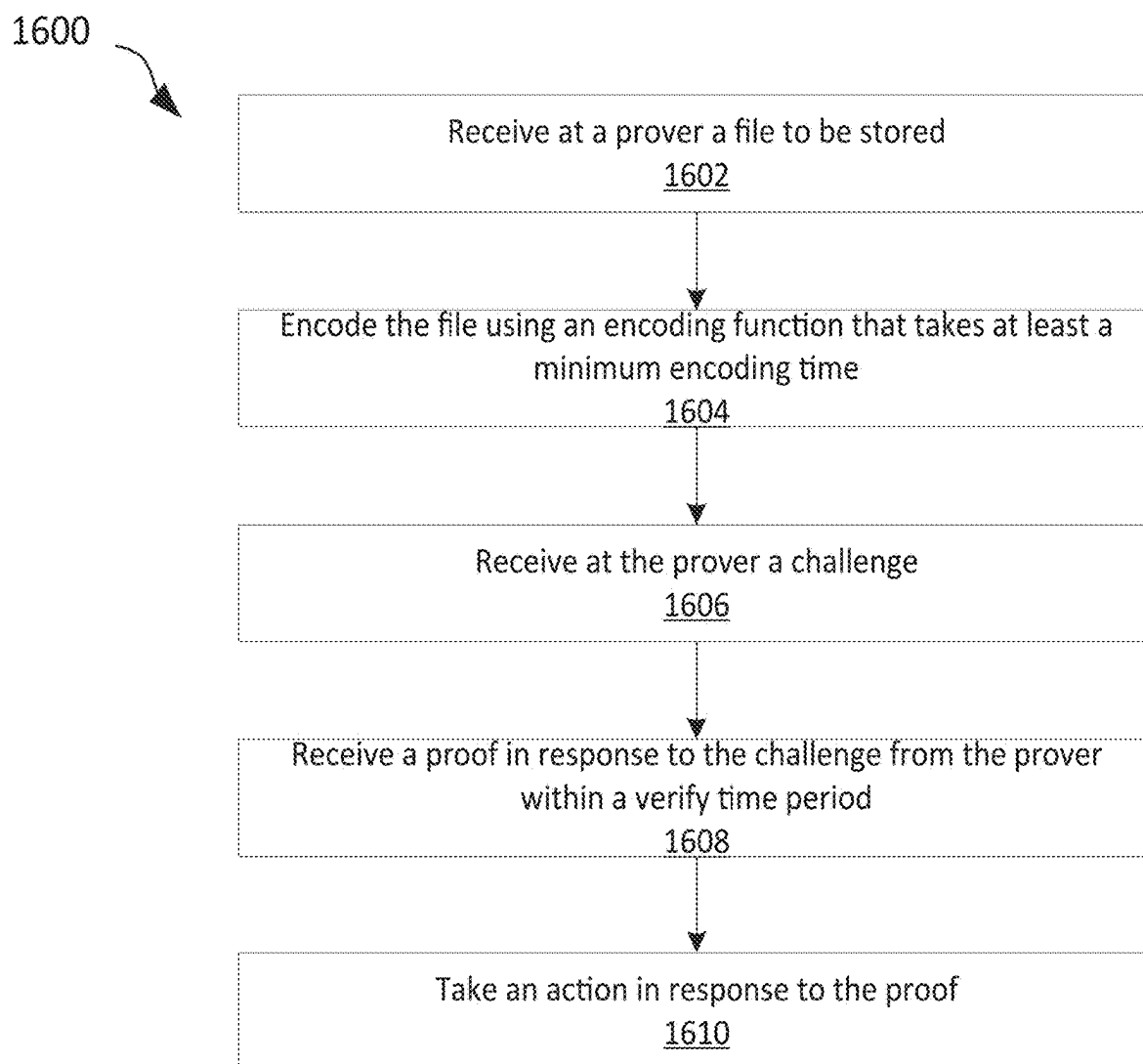


FIG. 16

PROTOCOLS FOR DECENTRALIZED NETWORKS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of, and claims priority to U.S. patent application Ser. No. 18/075,794, filed on Dec. 6, 2022, which is a continuation of, and claims priority to U.S. patent application Ser. No. 16/866,370, for Protocols for Decentralized Networks, to Nicola Greco and Juan Batiz-Benet, filed on May 4, 2020, which is a continuation-in-part of, and claims priority to U.S. patent application Ser. No. 16/195,536, filed on Nov. 19, 2018, which claims the benefit of the filing date of: U.S. Patent Application No. 62/697,123, filed on Jul. 12, 2018; U.S. Patent Application No. 62/697,091, filed on Jul. 12, 2018; and U.S. Patent Application No. 62/697,097, filed on Jul. 12, 2018, all of which are incorporated by reference herein in their entirety. As noted above, this application is a continuation of, and claims priority to U.S. patent application Ser. No. 18/075,794, filed on Dec. 6, 2022, which is also a continuation-in-part of, and claims priority to, U.S. patent application Ser. No. 16/726,520, for Protocols for Decentralized Networks, to Nicola Greco and Juan Batiz-Benet, filed on Dec. 24, 2019, which is a continuation-in-part application of, and claims priority to, U.S. patent application Ser. No. 16/195,621 for Protocols for Decentralized Networks to Nicola Greco and Juan Batiz-Benet, filed on Nov. 19, 2018 which claims the benefit of the filing date of U.S. Patent Application No. 62/901,207, for Protocols for Decentralized Networks, which was filed on Sep. 16, 2019, all of which are incorporated here by reference in their entirety. For the avoidance of doubt, the disclosure of all of the foregoing applications are incorporated here by reference in their entirety.

[0002] The present application is related to U.S. Utility application Ser. No. 16/195,597 for “Replication-Related Protocols for Decentralized Networks” to Nicola Greco, Juan Batiz-Benet and David Allen Dalrymple, filed on Nov. 19, 2018, and which is incorporated herein by reference in its entirety

BACKGROUND

Technical Field

[0003] This specification relates to distributed computing systems.

Background

[0004] A distributed computing system can serve as a decentralized data storage system with data replicated and synced across multiple computing nodes. A blockchain is a continuously growing list of data records linked and secured using cryptographic technology. A blockchain can be managed by a distributed computing system, i.e., a blockchain network, with each computing node in the distributed computing system adhering to a consensus protocol for inter-node communication and new block validation.

SUMMARY

[0005] This specification describes technologies for proving in a distributed computing system that one computing node has stored specified data for a specified time period.

These technologies generally involve a task-performing computing node generating a proof and the distributed computing system verifying the proof and awarding the task-performing computing node. As one example, Filecoin is a protocol token whose blockchain runs on a proof, called Proof-of-Spacetime, where blocks are created by miners that are storing data. Filecoin protocol provides a data storage and retrieval service via a network of independent storage providers that does not rely on a single coordinator, where: (1) clients pay to store and retrieve data, (2) storage miners earn tokens by providing storage and (3) retrieval miners earn tokens by serving data.

[0006] In general, one innovative aspect of the subject matter described in this specification can be embodied in methods that include the actions of: (i) receiving, at a prover (e.g., a storage provider), an initial input challenge; (ii) producing, at the prover, an output proof, wherein producing the output proof includes: (a) producing, at the prover, an initial proof responsive to the initial input challenge, the initial proof proving that the prover is storing specified data; (b) generating, at the prover, a new input challenge based at least in part on the initial proof; (c) producing, at the prover, a new proof responsive to the new input challenge, the new proof proving that the prover is storing the specified data; and (d) repeating, at the prover, the generating step and the producing a new proof responsive to the new input challenge step a number of times, the repeating step creating sequential proofs of storage proving that the prover has access to specified data for at least a specified period of time, wherein each generating step is based at least in part on a most recent new proof; (iii) forwarding the output proof (e.g. to a blockchain); and (iv) receiving, at a prover, a response, e.g., tokens, at least in part in response to forwarding the output proof.

[0007] These methods can use a variety of proofs of storage. These methods can also be used with a variety of proofs of retrievability. In proving that the prover has access to specified data for at least a specified period of time, the specified period of time can be any length; a verifier can make the period longer if the verifier wants to reduce communication with the prover. The prover can generate a proof of spacetime and then submit on chain (to the blockchain) the proof along with an identifier for the prover's identity and for the data the prover is claiming to store (e.g., a cryptographic commitment to the data the prover is claiming to store).

[0008] Once the response (e.g. an output proof) is added to the allocation table of the blockchain, an award can be provided to the prover/storage miner. The decentralized network can determine and process the award in a variety of ways. In Filecoin: a miner can create a “deal” with a client, where the client promises that a specified amount of money will unlock at every proof they provide. This basically represents the reward for storing users data and is agreed upon by the client and the miner. This can be implemented via a smart contract or through payment channels (using payment channels can be easier logistically). In the case of the contract, the contract can state the following: if a correct proof is seen in the range of blocks X, X+allowed_delay, then the miner is given the money locked in the contract. In the case of the payment channel: the client gives to the miner a signed statement which says that if the miner has a proof

at the right time on chain between time X, X+allowed time, then the miner can unlock some money locked by the client in the payment channel.

[0009] What is discussed immediately above are storage rewards. The method can also provide block rewards. The network can have a power table and when a miner submits a proof of replication, new power is added to the power table. In one implementation, every specified period of time, a miner must submit a proof of space time to the blockchain to maintain the power with which the prover started. If the prover stops doing so, the prover loses power. At every block, a miner is sampled from the power table. The sampled miner wins the block reward. This reward is implemented via a consensus rule.

[0010] Other embodiments of this aspect include corresponding computer systems, apparatus, and computer programs recorded on one or more computer storage devices, each configured to perform the actions of the methods. For a system of one or more computers to be configured to perform particular operations or actions means that the system has installed on it software, firmware, hardware, or a combination of them that in operation cause the system to perform the operations or actions. For one or more computer programs to be configured to perform particular operations or actions means that the one or more programs include instructions that, when executed by data processing apparatus, cause the apparatus to perform the operations or actions.

[0011] The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination. In particular, one embodiment includes all the following features in combination. The output proof can be labeled with a loop counter index. The method can further include checking, at a verifier, if the prover is storing the specified data for at least a minimum period of time. Receiving a response to adding the output proof can include receiving at a prover a stake in management of a file storage system and/or receiving specified benefits. The output proof can be a sequence of consecutive proofs, which can be referred to as a proof chain. An input challenge has a timestamp from the chain. The output proof is posted on chain and hence time stamped by the chain, again. The type and frequency of input challenges can be specified during a setup process by the prover and the verifier.

[0012] The setup process for Filecoin takes place in a context where a prover is a miner and the verifier is the blockchain. There is a client that indicates that it needs to see one proof every specified number of blocks in order to provide rewards. Then the blockchain verifies that such proofs are being received as required. In some implementations, e.g., in filecoin, the client does not pay a verifier, although there could be systems where this is the case. Rather in Filecoin, the client pays a primary miner and the primary miner pays any other miners that participate in the proof.

[0013] In general, another innovative aspect of the subject matter described in this specification can be embodied in methods that include the actions of: (i) receiving data at a prover; performing a seal operation on the data; (ii) receiving, at the prover, an initial input challenge; (iii) producing an output proof, wherein producing the output proof comprises: (a) producing an initial proof responsive to the initial input challenge, the initial proof proving that the prover is

storing specified data; (b) generating a new input challenge based at least in part on the initial proof; (c) producing a new proof responsive to the new input challenge, the new proof proving that the prover is storing the specified data; and (d) repeating the generating step and the producing a new proof responsive to the new input challenge step a specified number of times resulting in an output proof, wherein the generating a new input challenge is based at least in part on a most recent new proof, wherein performing the seal operation is several orders of magnitude slower than producing an output proof; (iv) forwarding the output proof to a blockchain; and (v) receiving a response, e.g., tokens at least in part based on forwarding the output proof to the blockchain. A seal is the slow sequential encoding of the proof of replication. The proof of the proof of replication proves that the seal hash been done correctly.

[0014] Receiving an input challenge can include receiving an input challenge at a start time, and the input challenge can have a time stamp indicating the start time. The output proof can also have a time stamp indicating an approximation for the time that the blockchain receives the output proof. A third-party service can provide the time-stamp for the start time and for the approximate time that the blockchain receives the output proof. The time stamp for the input challenge can be based on the time the third party service receives the input challenge and the time stamp for approximate time the blockchain receives the output proof can be based on the time the third party service receives the proof result. The seal operation can be selected to last at least a minimum seal time and the method can further include determining that a time period between the start time time-stamp and the output proof time-stamp is less than the minimum seal time. By requiring that the prover provide an output proof in less time than the minimum seal time, a proof of spacetime protocol prevents certain types of malicious behavior such as performing a seal operation only upon receipt of the challenge.

[0015] The subject matter described in this specification can be implemented in particular embodiments so as to realize one or more of the following advantages. A distributed computing network described in this specification can reliably and quickly verify that a task-performing node has stored specified data for a specified time period and participants can verify such storage in the distributed computing network (without requiring the intervention of a third-party verifier that is not a participant in the distributed computing network and that does not use an implementation of the verification protocols described in this specification). Such a so-called proof-of-spacetime facilitates the operation of a decentralized storage network which can utilize vast otherwise unused resources, e.g., storage resources. Proof-of-spacetime also facilitates useful work proofs in a blockchain context increasing the usefulness of blockchain technology. Proof-of-spacetime can be used to facilitate establishing consensus in a blockchain context. Proof of spacetime in the cloud setting can provide storage guarantees to clients. Proof of spacetime can provide a proof of work (beyond blockchain) as an anti-spam technique (e.g. in order to send an email, an entity needs to prove it had some specified data for some time, e.g., for a specified period of time). Proof-of-spacetime and other methods described in this specification are useful to reduce the effectiveness of attacks that malicious miners might try to exploit, e.g., Sybil, outsourcing, and generation attacks (described further below).

[0016] The details of one or more embodiments of the subject matter of this specification are set forth in the accompanying drawings and the description below. Other features, aspects, and advantages of the subject matter will become apparent from the description, the drawings, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] FIG. 1A is a schematic depiction of an example distributed computing system.

[0018] FIG. 1B is an example of a decentralized storage network protocol.

[0019] FIG. 2 is a schematic depiction of an example workflow for fulfilling a storage request in the distributed computing system.

[0020] FIG. 3 is a schematic depiction of an example workflow for generating a space-time proof, e.g., at a prover.

[0021] FIG. 4 illustrates examples of proof-of-replication and proof-of-spacetime protocols.

[0022] FIG. 5 illustrates an example of data structures in a decentralized storage network (DSN) scheme.

[0023] FIG. 6 illustrates an example execution of a Filecoin DSN, grouped by party and sorted chronologically by row.

[0024] FIG. 7 illustrates examples of the put and get protocols in a Filecoin DSN.

[0025] FIG. 8 illustrates example of the manage protocol in a Filecoin DSN.

[0026] FIG. 9 illustrates an example generic protocol for a verifiable market/exchange.

[0027] FIG. 10 illustrates an example of order data structures for retrieval and storage markets/exchanges.

[0028] FIG. 11 illustrates an example of a detailed storage market/exchange protocol.

[0029] FIG. 12 illustrates an example of a detailed retrieval market/exchange protocol.

[0030] FIG. 13 illustrates an example of a leader election process via an expected consensus protocol.

[0031] FIG. 14 is a flowchart of an example process for a space-time proof.

[0032] FIG. 15 is a flowchart of an example process for determining a winning prover from a plurality of candidate provers.

[0033] FIG. 16 is a flowchart of an example process for proof of replication.

[0034] Like reference numbers and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0035] This specification relates to technologies for proving in a distributed computing system that one computing node has stored specified data for a specified time period. These technologies further relate to a task-performing computing node generating a proof of performance of the task, e.g., a proof of storage, and the distributed computing system verifying the proof and awarding the task-performing computing node. As one example, Filecoin protocol is a protocol utilizing a blockchain that runs on a proof, called Proof-of-Spacetime, where blocks are created by miners that are storing data. The Filecoin protocol provides a data storage and retrieval service via a network of independent storage providers that does not rely on a single coordinator, where: (1) clients reward storage providers/storage miners

and storage retrievers/retrieval miners for storing and retrieving data, (2) storage miners earn tokens by providing storage and (3) retrieval miners earn tokens by retrieving data. Storage miners can also earn rewards (e.g., tokens) for participating in a consensus process, e.g., in a consensus process based on storage provided by miners.

[0036] The Filecoin protocol builds upon four components (this specification contemplates other protocols as well as variations on these components):

[0037] 1. A Decentralized Storage Network (DSN) is a network of independent storage providers that offer storage and retrieval services. The Filecoin protocol is an incentivized, auditable and verifiable DSN construction. A decentralized network reduces or eliminates the networks reliance on a single coordinator (e.g., a single computing node) for at least a primary function of the network.

[0038] 2. Proofs-of-Storage: The Filecoin protocol utilizes two Proofs-of-Storage: (1) Proof-of-Replication allows storage providers to prove that data has been replicated to its own uniquely dedicated physical storage. Enforcing unique physical copies enables a verifier to check that a prover is not deduplicating multiple copies of the data into the same storage space; (2) Proof-of-Spacetime allows storage providers to prove they have stored specified data throughout a specified amount of time.

[0039] 3. Verifiable Exchanges: The Filecoin protocol models storage requests and retrieval requests as orders in two decentralized verifiable exchanges operated by the Filecoin network. Verifiable exchanges ensure that rewards are distributed when a service has been correctly provided. The Storage Exchange and the Retrieval Exchange allow miners and clients to submit storage and retrieval orders, respectively.

[0040] 4. Useful Proof-of-Work: This specification describes how to construct a useful Proof-of-Work based on Proof-of-Spacetime that can be used in consensus protocols. Miners do not need to spend wasteful computation to mine blocks, but instead store data in the network to do so.

[0041] The Filecoin protocol is a DSN construction built on a blockchain and with a native token, i.e., the Filecoin token. Clients use tokens for storing and retrieving data. Miners earn tokens by storing and serving data. The Filecoin DSN handles storage and retrieval requests respectively via two verifiable exchanges: the Storage Exchange and the Retrieval Exchange. Clients and miners set the values for the services requested and offered and submit their orders to the exchanges.

[0042] The Filecoin network operates the exchanges which employ Proof-of-Spacetime and Proof-of-Replication to guarantee that storage miners have correctly stored the data they committed to store. Finally, miners can participate in the creation of new blocks for the underlining blockchain. In one embodiment, the probability of a candidate miner being a miner that can create a new block is proportional to the candidate miner's assigned storage. Assigned storage is indicated at least in part by the candidate miner's proof of work, e.g., the candidate miner's proof of the amount of its storage currently in use in the network. Stated differently, the probability of a storage miner being selected (by a selection protocol) to create a new block in the blockchain is proportional to the amount of storage currently being provided by the storage miner via the DSN. In other implementations, the probability of a storage miner being selected can take into account the storage miner's storage over a period of time.

[0043] FIG. 1A is a schematic depiction of one example of a Filecoin DSN 100. The illustrated Filecoin DSN 100 includes a bid-ask matching engine 102, a Filecoin blockchain 104, a consensus protocol 106 for maintaining the Filecoin blockchain/DSN 104, a plurality of computing nodes serving as a storage seeker/client 110, a storage provider/storage miner 108 connected/including a proof of Space-Time Engine 109, and a storage retriever/retrieval miner 112. Each of the plurality of computing nodes in the Filecoin DSN 100 can take on one or more roles as the client 110, the storage miner 108, and/or the retrieval miner 112.

[0044] At step A, the client/storage seeker 110 can submit a storage bid order (e.g., specifying terms that the client is willing to provide in order for a storage miner to store specified data for a specified time) to a bid-ask matching engine 102. At step B, the storage miner 108 can submit a storage ask order (e.g., indicating the amount of storage in can provide for at least one specified time period for a specified reward) to the bid-ask matching engine 102. Steps A and B do not have to occur in order. At step C, the bid-ask matching engine 102 determines if there is a matching client bid/storage miner ask. If there is a match, the bid-ask matching engine can create a deal and (as described further below) the storage miner who submitted the ask and the client who submitted the bid can sign the deal. When the deal is signed by both the relevant storage miner and the relevant client, the signed deal can be recorded on the blockchain. At step D, the agreed upon data is transmitted (e.g., from the client 110 to the storage miner 108) for storage. Alternatively, the bid-ask matching engine, upon determining that there is a matching client bid and storage miner ask, can instruct the relevant client to send the relevant storage miner client data. The storage miner can then check if the received data is of the right size as specified in the storage bid order. If the received data is of the right size, the storage miner can sign the order (which can then be referred to as a deal order), transmit the deal order to the blockchain, and store the data.

[0045] At step E, a verifier on the DSN 100 (shown in FIG. 1 as residing in element 104) can issue a challenge to the storage miner 108. In response to the challenge, at step F, the storage miner 108 can provide a proof back to the verifier on the DSN 100. In an alternative embodiment, the storage miner does not require a challenge to provide the proof.

[0046] Similarly to the storage scenario described above, at step G, a client 110 can submit a retrieval bid order to the bid-ask matching engine 102 and a retrieval miner 112 can submit a retrieval ask order to bid-ask matching engine 102. If there is a match, at step I the retrieval miner 112 can retrieve the stored data from the storage miner 108 and, at step J, the retrieval miner 112 can provide the stored data to the client 110.

[0047] In one implementation, each of the plurality of computing nodes, e.g., client 110, the storage miner 108, and/or the retrieval miner 112, is connected to every other computing node in the Filecoin DSN 100. For example, the client 110, the storage miner 108, and/or the retrieval miner 112 can be connected using wired or wireless connections such as Ethernet, WLAN, Bluetooth, and WiFi.

[0048] The client 110, the storage miner 108, and the retrieval miner 112 each includes one or more processors programmed to execute a set of commands according to a Filecoin protocol. The Filecoin protocol specifies actions each computing node is programmed to take such as sub-

mitting a storage request, submitting a storage offer, submitting a retrieval request, and submitting a retrieval offer.

[0049] The storage miner 108 includes one or more memory devices that can store, replicate, and return data per requests from the client 110 or the retrieval miner 112. The memory devices can be divided into sectors and each sector can be sealed when storing a piece of data. This specification may refer to data as data, piece of data or piece interchangeably. Sealing is described below.

[0050] The Filecoin DSN 100 includes a Filecoin blockchain 104. The Filecoin blockchain 104 can be an immutable, append-only linked list of blocks. Each block can record the distribution of Filecoin tokens, the proposed and completed transactions, currently open storage/retrieval orders, and proofs of completed orders submitted at a specific time. The Filecoin blockchain 104 is used to implement a ledger-based reward system, with the storage provider 108 rewarded for successfully storing data from the client 110. Each computing node in the Filecoin DSN can keep a local copy of the Filecoin blockchain. In an alternative embodiment, each miner or full node keeps a local copy of the Filecoin blockchain.

[0051] In one embodiment, when a change is made to the Filecoin blockchain, e.g., a new completed transaction, the change is broadcast to the entire Filecoin DSN and each local copy of the Filecoin blockchain is updated to reflect the change. An alternative embodiment can propagate changes only to a subset of nodes, or post a proof of correct changes.

[0052] FIG. 1B is an example of a decentralized storage network protocol, e.g., the Filecoin protocol. The Filecoin protocol includes a set of instructions that cause the Filecoin DSN 100, the client(s) 110, the storage miner(s) 108, and the retrieval miner(s) 112 to perform operations facilitating the storage/retrieval of data and the settlement of transactions in Filecoin tokens.

Network

[0053] The Filecoin DSN 100 comprises a plurality of computing nodes. In one embodiment, for each new block added to the Filecoin blockchain 104, the Filecoin DSN 100 is configured to check:

- [0054] (a) if the block is in the valid format
- [0055] (b) if all transactions contained in the block are valid
- [0056] (c) if all orders contained in the block are valid
- [0057] (d) if all proofs contained in the block are valid
- [0058] (e) if all pledges contained in the block are valid

[0059] In one implementation, if any of the above conditions are not met, the block can be disregarded and not appended to the Filecoin blockchain 104.

[0060] In one embodiment, for each new order introduced to the Filecoin DSN 100, the Filecoin DSN 100 is configured to:

- [0061] (a) add the order to the Filecoin blockchain's orderbook
- [0062] (b) if the order is a storage bid order, lock the associated Filecoin tokens from the client 110
- [0063] (c) if the order is a storage ask order, lock the associated memory space from the storage miner
- [0064] (d) if the order is a deal order (e.g., a order for storage that has been agreed to by both a client and a storage miner), assign the data from the client 110 to the storage miner 108.

[0065] In one embodiment, for each order that has already been recorded in the Filecoin blockchain **104**, the Filecoin DSN is configured to:

[0066] (a) check if the order has expired or been canceled. For example, each order can have a time associated with the order. If the specified time runs out, the Filecoin DSN **100** can remove the order from the Filecoin blockchain **104**, return unspent Filecoin tokens to the client, and free the space for the storage miner.

[0067] (b) If the order is a deal order, check if the expected proofs have been generated by the storage miner. If any of the proofs are missing, the Filecoin DSN penalizes the storage miner by taking part of its pledged collateral (a storage miner can be required to commit collateral to the DSN as part of submitting a storage ask order to the DSN). If a large amount of proofs are missing, the Filecoin DSN can cancel the order and reintroduce the order to the market. In another example, if the data cannot be retrieved from the storage miner, the Filecoin DSN cancels the order and refunds the client.

Storage Miner

[0068] In one embodiment, at any time the storage miner **108** can execute the Filecoin protocol to perform the following example actions: (1) renew expired pledges by putting new collateral into the Filecoin blockchain, (2) pledge new storage, and (3) submit a new storage ask order to the Filecoin blockchain.

[0069] In one embodiment, for each storage ask order in the orderbook of the Filecoin blockchain **104**, the storage miner **108** is configured to:

[0070] (a) find matched storage bid orders from clients stored in the orderbook, and

[0071] (b) start a deal order by contacting the matching client **110**.

[0072] In one embodiment, for each sector of storage space pledged to the Filecoin blockchain **104**, the storage miner **108** is configured to:

[0073] (a) generate proofs of storage, and

[0074] (b) at the appropriate time, post the proofs to the Filecoin blockchain **104**.

[0075] In one embodiment, after a storage ask order is matched, the storage miner **108** receives data from the client **110**. The storage miner **108** can then be configured to, as specified by the Filecoin protocol:

[0076] (a) check if the piece of data is of the right size as specified in the storage bid order

[0077] (b) create a deal order and sign the deal order, for example, by signing the order with a public key associated with the storage miner in the Filecoin DSN

[0078] (c) store the data in a sector of its storage space, and

[0079] (d) seal a sector if the sector is full.

Client

[0080] In one embodiment, at any time the client **110** can submit a new storage bid order to the Filecoin blockchain **104**. The client **110** can further find one or more matching storage ask orders and send data to the matched storage miner **108**.

[0081] After receiving a signed deal order from a storage miner **108**, the client **110** can sign the deal order, e.g., using

its own public key in the Filecoin DSN **100**, and submit the signed deal order to the Filecoin blockchain **104**.

[0082] In one embodiment, at any time the client **110** can submit a new storage retrieval order to the Filecoin blockchain **104**. The client **110** can find one or more matching retrieval orders and create a Filecoin token payment channel with the retrieval miner **112**.

[0083] After receiving data from the retrieval miner **112**, the client **110** can verify the data and send Filecoin tokens as a reward to the retrieval miner **112**.

Retrieval Miner

[0084] In one embodiment, at any time the retrieval miner **112** can submit a retrieval ask order to the Filecoin blockchain **104** and search for a matching retrieval bid order from the Filecoin blockchain **104**.

[0085] In one embodiment, after matching a retrieval request from the client **110**, the retrieval miner **112** is configured to:

[0086] (a) initiate a payment/reward channel with the client **110**;

[0087] (b) retrieve specified data in multiple parts from the relevant storage miner, and

[0088] (c) send the data in parts to the client after receiving payment/reward for a respective part.

Storage Exchange and Storage Requests

[0089] With continued reference to FIG. 1, in one embodiment, the client **110** can store data using the Filecoin DSN **100** by rewarding a storage miner **108** in Filecoin tokens for storing specified client data. For example, the client **110** can submit a bid order according to the Filecoin protocol to a storage exchange orderbook. The storage exchange orderbook is part of the Filecoin blockchain and records the requests from all clients in the Filecoin DSN **100**. For example, the bid order can include information such as the identity of the client **110** in the Filecoin DSN **100**, the amount of data to be stored, the number of copies to be stored, the length of storage time, the maximum amount of Filecoin tokens to be awarded. The updated storage exchange orderbook will be broadcast to the Filecoin DSN **100** to be visible to participating computing nodes.

[0090] The storage miner **108** can earn Filecoin tokens specified in the bid order by fulfilling the bid order. To fulfill a bid order, the storage miner **108** first pledges to provide storage to the Filecoin DSN **100** according to the Filecoin protocol. For example, a pledge can include collateral in the form of Filecoin tokens. The collateral is deposited for the time intended to provide the service, and is returned to the storage miner **108** if the storage miner successfully fulfills the bid order, e.g., generating proofs of storage for the data it commits to store. If some proofs of storage fail, a proportional amount of the collateral will be lost. Generating proofs is explained below, e.g., with reference to FIG. 3. The pledges are recorded in the Filecoin blockchain **104** to be visible to computing nodes in the Filecoin DSN **100**.

[0091] In one embodiment, once the pledge appears in the Filecoin blockchain **104**, the storage miner **108** can submit an ask order to the DSN orderbook. In one embodiment order matching, pledges, and proofs are added on chain, meaning they are added to the blockchain. However parts of this protocol can be run off-chain using state channels and different trust assumptions (e.g. one embodiment of the

protocol can elect a set of nodes to perform the verification tasks that blockchain nodes would have to do otherwise). There are at least two things that can be done off-chain: propagation of proofs/orders (one can propagate proofs/orders with traditional gossiping) and verification of correctness (as described above)

[0092] Similar to the bid order, the ask order can include the identity of the storage miner **108**, the amount of storage the storage miner **108** offers, the length of the storage, and the asking award in Filecoin tokens. The bid-ask matching engine **102** periodically checks the current state of the ask orders from the client **110** and bid orders from the storage miner **108**, and matches ask orders to bid orders where the ask order substantially satisfies the bid order.

[0093] Once the bid-ask matching engine matches an ask order and a bid order, a piece of data is sent from the client **110** to the storage miner **108** for the storage miner to store. Upon receiving the data, the storage miner performs a sealing operation to transform the received data into a replica, e.g., a unique physical copy of the data that is associated with an identity of the storage miner.

[0094] The sealing operation is a slow, sequential operation that is several orders of magnitude slower to perform compared to the proof generation operation. As a result, the storage miner **108**, when requested to provide a proof of the storage, cannot generate a new replica on the fly since the expected time for a proof is much shorter than that the time to perform a sealing operation. For example, sealing can be realized using pre-existing techniques such as layering iterations of AES-256, Depth Robust Graphs, Slow Hash-functions (e.g. Sloth) to transform the received data into an encoded replica.

[0095] The deal order is submitted to the orderbook after the match. A network entity, e.g., the client/storage requester or the storage provider, can submit the order to the orderbook.

[0096] When the storage miner **108** is assigned the data, it can generate one or more proofs of replication (described further below) to indicate that the data is indeed being stored. The storage miner **108** posts a proof of storage on the Filecoin blockchain and other computing nodes in the Filecoin DSN **100** can verify the proofs. If a storage miner fails to generate a proof, or if a storage miner's proof is invalid, the storage miner **108** can be penalized (loss of tokens) and/or not rewarded. For example, in some embodiments, the storage miner **108** can lose all or part of its collateral.

Retrieval Exchange/Market and Retrieval Requests

[0097] In one embodiment, upon a request from a client **110**, a retrieval miner **112** retrieves specified stored data from the storage miner **108** for the client **110** and is awarded Filecoin tokens.

[0098] The client **110** can submit a bid order to the retrieval market orderbook by broadcasting a request to the Filecoin DSN **100**. The orderbook can be run off-chain by having nodes broadcasting orders.

[0099] The retrieval miner **112** submits a retrieval ask order to the same retrieval market orderbook by asking for a specified award in Filecoin tokens. When a matching bid-ask order is found, the client **110** receives the data from the retrieval miner **112**. When the data is received, the client and the retrieval miner **112** sign a deal order and submit it to the Filecoin blockchain **104** to confirm that the exchange succeeded.

[0100] The bid-ask engine again monitors the orderbook and matches asks and bids for the retrieval exchange. Micro-awards are made once the bid-ask engine makes a match and the deal is completed (as described further below with reference to FIG. 2).

Example Workflows

[0101] FIG. 2 is a schematic depiction of an example workflow for fulfilling a storage request in a distributed computing system. For convenience, the process **200** will be described as being performed by a system of one or more computers, located in one or more locations, and programmed appropriately in accordance with this specification. For example, a distributed computing system, e.g., the Filecoin DSN system **100** of FIG. 1, appropriately programmed, can perform the process **200**.

[0102] Stated differently, FIG. 2 is an illustration of one implementation of the Filecoin protocol, showing an overview of client (**202a**, **202b**)-miner (**204a**, **204b**) interactions. The storage market **206** and retrieval market **210** are shown above and below the blockchain **208**, respectively, with time advancing from the order matching phase **211** on the left to the Settlement **213** phase on the right. The blockchain includes an orderbook **208a** section, a transactions **208b** section and an allocation table **208c** section.

[0103] Turning first to order matching **211** for the storage market **206**, a client **202a** makes a storage bid order **212a** that is recorded in the orderbook **208a**. The client **202a** also records filecoin from the client pursuant to the storage bid in the transactions section **208b** of the blockchain **208**. One or more storage miners **204a** can also submit a storage ask order **216a** to the orderbook **208a** of the blockchain **208**. For example, the storage miner **204a** can first pledge its storage to the Filecoin DSN **100** (FIG. 1) by depositing collateral in Filecoin tokens according to a command specified in the Filecoin protocol. The pledge transaction is recorded on the Filecoin blockchain **208** and is visible to computing nodes in the Filecoin DSN **100** (FIG. 1). The collateral is deposited for the time intended to provide the storage service, and is provided back to the storage miner **204a** if the storage miner **204a** generates valid proofs of storage for the data it committed to store. If some proofs of storage fail, a proportional amount of the collateral is lost or given to the client **202a**.

[0104] After the storage miner **204a** completes the pledge transaction, the storage miner **204a** can offer its storage to client **202a** by submitting a storage ask order **216a** to the Filecoin blockchain **208** according to a command specified in the Filecoin protocol. The storage ask order **216a** is recorded in the orderbook **208a** of the Filecoin blockchain **208** and is visible to computing nodes in the Filecoin DSN **100** (FIG. 1). The storage ask order **216a** includes information such as the network identity of the storage miner **204a**, the amount of collateral, the storage space and length of time, and the asking price in Filecoin tokens.

[0105] With reference to FIG. 1, a bid-ask matching engine **102** can determine if there is an ask order from a storage miner that matches the client bid order, e.g., the engine **102** can determine if there is a storage miner that has submitted a qualifying ask order indicating the storage miner can store the data specified by the client under mutually agreeable terms. The matching can be performed by com-

paring one or more properties of the storage bid order **212a** to one or more corresponding properties of one or more storage ask orders **216a**.

[0106] With reference again to FIG. 2, the matched bid-ask deal **218** is logged in the orderbook **208a** of the blockchain **208** and forwarded to miner **204a** for signing. The storage miner signed deal **220a** is then forwarded to the client **202a** for signing. The storage miner and client signed deal **220a** is then logged in the allocation table **208c** of the blockchain **208** and a piece of data in question is transferred from the client **202a** to the storage miner **206**.

[0107] Turning to settlement **213** for the storage phase **206**, for the storage miner **204** to receive an award **214b** in Filecoin tokens for storing the data **224**, the storage miner **204** can prove to the Filecoin DSN that it is correctly storing the data **224** through the agreed upon time period. The storage miner **204** seals a sector of its storage device, thus creating a unique replica of the data **224**. A challenge **226** may issue to the storage miner **204** regarding a piece of data. In response to the challenge **226**, the storage miner **204** generates a proof **228** and submits the proof to the Filecoin blockchain. The proof is recorded in an allocation table **208c**. For example, the storage miner **204** can generate new proofs at every epoch and add them to the Filecoin blockchain using a command in the Filecoin protocol. As a result (e.g., if the response is verified), the award in filecoin **214b** can be transferred to the storage miner **204a**.

[0108] In one embodiment, the proofs are visible to the Filecoin DSN and can be checked by other computing nodes. If proofs are missing or invalid, the Filecoin DSN penalizes the storage miner **204** by taking part of the collateral. If a specified amount of proofs are missing or invalid for more than a specified period of time, the Filecoin DSN considers the storage miner **204** to be faulty, settles the order as failed, and reintroduces a new order for the same data into the Filecoin Blockchain **208**.

[0109] In one embodiment, when the time of the order expires or when the amount of time paid for by the client **202a** runs out, and the proof **228** is verified by the Filecoin DSN, the Filecoin DSN transfers the agreed amount of Filecoin tokens **214b** to the storage miner **204** and removes the order from the Filecoin blockchain **208**.

[0110] With regard to order matching **211** for retrieval **210**, the client **202b** can post a retrieval bid order, e.g., the client's retrieval bid order can be gossiped to a retrieval miner **204b** off chain. Similarly, a retrieval miner **204b** can post an ask order to a client **202b**, e.g., a retrieval miner's ask order can be gossiped to a client **202b** off chain. If there is a match between the client's retrieval bid order and the retrieval miner's ask order, the retrieval miner **204b** can sign the deal **220b** and the client **202b** can sign the deal **222b**.

[0111] Turning to the settlement **213** for the retrieval phase **210**, the retrieval miner **204b** can send the retrieved data **232** to the client **202b** in parts and after each part is sent the client **202b** can commit to micro-awards **230** for the retrieval miner **204b**. Note that in one embodiment before micro-awards can be made for retrieval, the client locks the tokens for the microtransaction, e.g., the client indicates that the tokens are locked for use as a micro-award for a specified deal. Once the client **202b** has received the specified data, the client **202b** can log a filecoin transfer **214c** with the transactions section **208b** of the blockchain **208**. Finally, the retrieval miner **204b** can claim its filecoin award **214d**.

Definition of a Decentralized Storage Network

[0112] This specification introduces the notion of a Decentralized Storage Network (DSN) scheme. DSNs aggregate storage offered by multiple independent storage providers and self-coordinate to provide data storage and data retrieval to clients. Coordination is decentralized and does not require trusted parties: the secure operation of these systems is achieved through protocols that coordinate and verify operations carried out by individual parties. DSNs can employ different strategies for coordination, including Byzantine Agreement, gossip protocols, or Conflict-free Replicated Data Types (CRDTs), depending on the requirements of the system. Later, this specification provides a construction for a Filecoin DSN.

[0113] Definition A DSN scheme is a tuple of protocols run by storage providers and clients:

[0114] (Put, Get, Manage)·

[0115] Put (data)→key: Clients execute the Put protocol to store data under a unique identifier key.

[0116] Get (key)→data: Clients execute the Get protocol to retrieve data that is currently stored using a specified key.

[0117] Manage () The network of participants coordinates via the Manage protocol to: control the available storage, audit the service offered by providers and repair possible faults. The Manage protocol is run by storage providers often in conjunction with clients or a network of auditors.

In one embodiment, a DSN scheme guarantees data integrity and retrievability as well as tolerates management and storage faults defined in the following sections.

Fault Tolerance

[0118] For example, management faults can be defined to be byzantine faults caused by participants in a Manage protocol. A DSN scheme relies on the fault tolerance of its underlying Manage protocol. Violations on the fault tolerance assumptions for management faults can compromise liveness and safety of the system. For example, consider a DSN scheme, where the Manage protocol requires Byzantine Agreement (BA) to audit storage providers. In such a protocol, the network receives proofs of storage from storage providers and runs BA to agree on the validity of these proofs. If the BA tolerates up to f faults out of n total nodes, then one implementation of a DSN can tolerate $f < n/2$ faulty nodes. Violations of this assumption can compromise audits, i.e., a violation of this assumption can reduce an auditor's (or client's) confidence, e.g., in the relevant proof of storage over time.

Storage Faults

[0119] Storage faults can be defined to be byzantine faults that prevent clients from retrieving the data: i.e. Storage Miners lose their pieces (data) and/or Retrieval Miners stop serving pieces. A successful Put execution is (f, m) -tolerant if it results in its input data being stored in m independent storage providers (out of n total) and it can tolerate up to f byzantine providers. The parameters f and m depend on protocol implementation; protocol designers can fix f and m or leave the choice to the user, extending Put (data) into Put (data, f, m). A Get execution on stored data is successful if there are fewer than f faulty storage providers.

[0120] For example, consider a simple scheme, where the Put protocol is designed such that each storage provider stores all of the data. In this scheme $m=n$ and $f=m-1$. Note that $f=m-1$ is not always true. Some schemes can be designed using erasure coding, where each storage provider stores a special portion of the data, such that x out of m storage providers are required to retrieve the data; in this case $f=m-x$.

Properties

[0121] In one embodiment, a DSN scheme can have two central properties, i.e., data integrity and retrievability, in addition to other properties.

[0122] The data integrity property states that no bounded (i.e., bounded in terms of time and storage resources) adversary A can convince clients to accept altered or falsified data at the end of a Get execution.

[0123] The DSN scheme provides data integrity if, for any successful Put execution for some data d under key k , there is no computationally-bounded adversary A that can convince a client to accept d' , for d' not equal to d at the end of a Get execution for identifier k .

[0124] The retrievability property states that, given the fault-tolerance assumptions of the DSN scheme, if some data has been successfully stored in the DSN scheme and storage providers continue to follow the protocol, then clients can eventually retrieve the data.

[0125] The DSN scheme provides retrievability if for any successful Put execution for data under a key, there exists a successful Get execution for the key for which a client retrieves data.

[0126] In one embodiment, DSN schemes can provide other properties specific to their application. Three example properties required by the Filecoin DSN include: public verifiability, auditability, and incentive-compatibility.

[0127] For example, a DSN scheme is publicly verifiable if for each successful Put, the network of storage providers can generate a proof that the data is currently being stored. The Proof-of-Storage convinces any efficient verifier, where the verifier only knows a key associated with the data in question and does not have access to the data in question.

[0128] A DSN scheme can be auditable, for example, if it generates a verifiable trace of operation that can be checked in the future to confirm storage was indeed stored for the correct duration of time.

[0129] A DSN scheme can be incentive-compatible, for example, if storage/retrieval providers are rewarded for successfully offering storage and/or retrieval services, or penalized for misbehaving, such that the storage providers' dominant strategy is to store and/or retrieve data.

Proof-of-Replication and Proof-of-Spacetime

[0130] In the Filecoin protocol, storage providers/miners convince their clients that they stored the data they are being paid to store; in practice, storage providers generate Proofs-of-Storage (PoS) that the blockchain network (or the client itself) verifies.

[0131] This specification describes implementations for Proof-of-Replication (PoRep) and Proof-of-Spacetime (PoSt) schemes used in Filecoin.

Motivation

[0132] In one implementation, Proofs-of-Storage (POS) schemes allow a user, e.g., the client, who outsources data to a server, e.g., the storage miner, to repeatedly check if the server is still storing the data outsourced to the server. The user can verify the integrity of the data outsourced to a server in a more efficient way than downloading the data. For example, the server can generate a probabilistic proof of possession by sampling a random set of blocks and transmitting a small amount of data in a challenge/response protocol with the user.

[0133] Provable data possession (PDP) and proof of retrievability (POR) schemes only guarantee that a prover had possession of some data at the time of the challenge/response. In Filecoin, stronger proofs are useful to prevent three types of attacks that malicious miners might try to exploit in an effort to get rewarded for storage they are not providing: Sybil attacks, outsourcing attacks, generation attacks.

[0134] Sybil attacks: malicious miners can pretend to store (and get paid for) more copies than the ones physically stored by creating multiple sybil identities, but in fact storing the data only once.

[0135] Outsourcing attacks: malicious miners could commit to store more data than the amount they can physically store, relying on quickly fetching data from other storage providers.

[0136] Generation attacks: malicious miners could claim to be storing a large amount of data which they are instead efficiently generating on-demand using a small program. If the program is smaller than the purportedly stored data, this inflates the malicious miner's likelihood of winning a block reward in Filecoin, which (as described further below) can be proportional to the miner's storage currently in use.

Proof-of-Replication

[0137] Proof-of-Replication (PoRep) is a Proof-of-Storage which allows a server (i.e. the prover P) to convince a user (i.e. the verifier V) that some data D has been replicated to its own uniquely dedicated physical storage. The PoRep protocol is an interactive protocol, where the prover P : (a) commits to store n distinct replicas (physically independent copies) of some data D , and then (b) convinces the verifier V , that P is indeed storing each of the replicas via a challenge/response protocol. PoRep improves on PoR and PDP schemes, reducing/preventing Sybil Attacks, Outsourcing Attacks, and Generation Attacks.

[0138] Definition (Proof-of-Replication) A PoRep scheme enables an efficient prover P to convince a verifier V that P is storing a replica R , a physical independent copy of some data D , unique to P . A PoRep protocol is characterized by a tuple of polynomial-time algorithms:

(Setup, Prove, Verify)

[0139] PoRep: Setup ($1^\lambda, D$) \Rightarrow R, S_p, S_v , where S_p and S_v are scheme-specific setup variables for P and V , and λ is a security parameter. PoRep: Setup is used to generate a replica R , and give P and V the necessary information to run PoRep: Prove and PoRep: Verify. Some schemes may require the prover or interaction with a third party to compute PoRep: Setup.

[0140] PoRep: Prove (S_p, R, c) $\Rightarrow \Pi^c$, where c is a random challenge issued by a verifier V , and π^c is a proof that a prover has access to R a specific replica of D . PoRep.Prove is run by P to produce a π^c for V .

[0141] PoRep: Verify (S_v, c, π^c) $\Rightarrow \{0, 1\}$, which checks whether a proof is correct. PoRep. Verify is run by V and convinces V whether P has been storing R .

Proof-of-Spacetime

[0142] Proof-of-Storage (POS) schemes allow a user to check if a storage provider is storing the outsourced data at the time of the challenge. A user can repeatedly (e.g. every minute) send challenges to a storage provider in an attempt to determine that the storage provider is storing data at more times than at a single instance. However, the communication complexity required in such a scheme can be a bottleneck in systems such as Filecoin, where storage providers submit their proofs to the blockchain network.

[0143] To address this issue, one can use Proof-of-Spacetime, where a verifier can check if a prover is storing its outsourced data for at least a minimum period of time without having the user (e.g., a client or other verifier) being required to repeatedly send challenges to the storage provider over a period of interest. One intuition is to require the prover to (1) generate sequential Proofs-of-Storage (in one case Proof-of-Replication), as a way to determine time and (2) recursively compose the executions to generate a short proof.

[0144] Definition (Proof-of-Spacetime): A PoSt scheme enables an efficient prover P to convince a verifier V that P is storing some data D for some period of time t . A PoSt is characterized by a tuple of polynomial-time algorithms:

(Setup, Prove, Verify)

[0145] PoSt: Setup ($1^\lambda, D$) $\Rightarrow S_p, S_v$, where S_p and S_v are scheme-specific setup variables for P and V , λ is a security parameter. PoSt: Setup is used to give P and V the necessary information to run PoSt: Prove and PoSt: Verify. Some schemes may require the prover or interaction with a third party to compute PoSt: Setup.

[0146] PoSt: Prove (S_p, D, c, t) $\Rightarrow \Pi^c$, where c is a random challenge issued by a verifier V , and π^c is a proof that a prover has access to D for some period of time t . PoSt: Prove is run by P to produce a π^c for V .

[0147] PoSt: Verify (S_v, c, t, π^c) $\Rightarrow \{0, 1\}$, which checks whether a proof is correct. PoSt: Verify is run by V and convinces V whether P has been storing D for some period of time t .

Practical PoRep and PoSt

[0148] This specification describes practical PoRep and PoSt constructions that can be deployed in existing systems and do not rely on trusted parties or hardware. One construction for PoRep uses a very slow sequential computation Seal performed during Setup to generate a replica. The protocol sketches for PoRep and PoSt are presented in FIG. 4 and the underlying mechanism of the proving step in PoSt is illustrated in FIG. 3.

[0149] More specifically, FIG. 3 is a schematic depiction of an example workflow for generating a space-time proof, e.g., at the storage miner 108 in FIG. 1A (also depicted as 204a in FIG. 2). FIG. 3 illustrates the underlying mechanism of PoSt.Prove showing an iterative proof to demonstrate

storage over a period of time. Initially, an input challenge 302 is forwarded to a prover, e.g., from a verifier (FIG. 2 shows the challenge 226 coming from the allocation table of the blockchain). The proof of spacetime engine 300 at the prover receives an initial input challenge 302 at the new challenge generation engine 304. Given receipt of the initial input challenge, the new challenge generation engine 304 passes the initial input challenge on to proof generation engine 310 which generates a proof 312, e.g., an initial proof labelled to indicate that it is responsive to the initial challenge 302.

[0150] In some embodiments, the prover does not need to receive a challenge for the very first proof. In other words, some proof of replication schemes might be non-interactive, and thus do not need a challenge (for example, using fiat-shamir heuristics). The proof described can be interactive or non-interactive proofs (where the challenge for the first proof is not needed).

[0151] The initial proof is then fed back into the new challenge generator 304 to generate a new challenge 306. The process of feeding a proof back into the new challenge generator completes a loop of the proof of spacetime engine 300 and increments a loop counter 304a that also feeds into the new challenge generator 304 allowing the new challenge generator to label the challenge 306 with a loop counter index which in turn allows the proof generator 310 to label each output proof 312 with a loop counter index.

[0152] How often a verifier requests a proof can be dictated by the time that it takes to run a sealing process (e.g., to run the setup for the proof of replication). However, in a general setting (no proof of replication), the closer the requests, the higher the guarantees of files being stored through time. If one wants a file to be available every hour, then one can have a challenge per hour, if one want a file to be more regularly available, one can have a challenge every minute.

Cryptographic Building Blocks

[0153] Collision-resistant hashing. With continuing reference to FIG. 3, the proof-of-spacetime engine 300 can use a collision resistant hash function CRH: $\{0, 1\}^* \rightarrow \{0, 1\}^O(\lambda)$. The protocol can use a collision resistant hash function MerkleCRH 308, which divides a string in multiple parts, constructs a binary tree and recursively applies CRH and outputs the root 309. The root 309 can then be an input into the proof generation engine 310.

[0154] zk-SNARKs. Implementations of PoRep and PoSt can rely on zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs). Because zk-SNARKs are succinct, proofs are very short and easy to verify. More formally, let L be an NP (nondeterministic polynomial time) language and C be a decision circuit for L . A trusted party conducts a one-time setup phase that results in two public keys: a proving key pk and a verification key vk . The proving key pk enables any (untrusted) prover to generate a proof attesting that $x \in L$ for an instance x of her choice. The non-interactive proof x is both zero-knowledge and proof-of-knowledge. A proof about some computation is zero knowledge if it doesn't leak any private input or intermediate steps. A malicious verifier can learn nothing beyond the output. A proof of knowledge is a proof where a prover can convince a verifier that they know of some witness. The prover needs to demonstrate that they have this knowledge. In a standard protocol, the prover can just send the witness

to the verifier. In a zero-knowledge proof of knowledge, the prover proves that they have some knowledge without leaking what their knowledge is or any intermediate steps (in one case the knowledge is a file being stored). In certain implementations zero knowledge is not used. Anyone can use the verification key vk to verify the proof π ; in particular zk-SNARK proofs are publicly verifiable: anyone can verify x , without interacting with the prover that generated x . The proof x has constant size and can be verified within a time period that is linear in $|x|$.

[0155] A zk-SNARK for circuit satisfiability is a triple of polynomial-time algorithms

(KeyGen, Prove, Verify)

[0156] KeyGen ($1^\lambda, C$) \Rightarrow (pk, vk). On input security parameter λ and a circuit C , KeyGen probabilistically samples pk and vk . Both keys are published as public parameters and can be used to prove/verify membership in L_C .

[0157] Prove (pk, x, w) $\Rightarrow \pi$. On input pk and input x and witness for the NP-statement w , the prover Prove outputs a non-interactive proof π for the statement $x \in L_C$.

[0158] Verify (vk, x, π) $\Rightarrow \{0, 1\}$. On input vk , an input x , and a proof π , the verifier Verify outputs 1 if $x \in L_C$.

[0159] Implementations of zk-SNARK systems are available. Generally these systems require the KeyGen operation to be run by a trusted party; work on Scalable Computational Integrity and Privacy (SCIP) systems can avoid this initial step.

Seal Operation

[0160] The role of the Seal operation is to (1) have replicas be physically independent copies by having provers store a pseudo-random permutation of D unique to their public key, such that committing to store n replicas results in dedicating disk space for n independent replicas (hence n times the storage size of a replica) and (2) to have the generation of the replica during PoRep: Setup take substantially longer than the time expected for responding to a challenge.

[0161] The above operation can be realized with an appropriate seal technique such as SealTM_{AES-256}. SealTM_{AES-256} takes 10-100x longer than the honest challenge-prove-verify sequence. Note one can choose T such that running SealTM_{BC} is distinguishably more expensive than running Prove with random access to R .

Practical PoRep Construction

[0162] This section describes the construction of the PoRep protocol and includes a simplified protocol sketch in FIG. 4.

[0163] Creating a Replica. The Setup algorithm generates a replica via the Seal operation and a proof that it was correctly generated. The prover generates the replica and sends the outputs (excluding R) to the verifier.

Setup

[0164] inputs:

[0165] prover key pair (pkp ; skp)

[0166] prover SEAL key pk_{SEAL}

[0167] data D

[0168] outputs: replica R , Merkle root rt of R , proof SEAL

[0169] Proving Storage. The Prove algorithm generates a proof of storage for the replica. The prover receives a random challenge, c , from the verifier, which determines a specific leaf R_c in the Merkle tree of R with root rt ; the prover generates a proof of knowledge about R_c and its Merkle path leading up to rt .

Prove

[0170] inputs:

[0171] prover Proof-of-Storage key pk_{pos}

[0172] replica R

[0173] random challenge c

[0174] outputs: a proof TIPOS

[0175] Verifying the Proofs. The Verify operation checks the validity of the proofs of storage given the Merkle root of the replica and the hash of the original data. Proofs are publicly verifiable: nodes in the distributed system maintaining the ledger and clients interested in particular data can verify these proofs.

Verify

[0176] inputs:

[0177] prover public key, pkp

[0178] verifier SEAL and POS keys vk_{SEAL}, vk_{POS}

[0179] hash of data D , hp

[0180] Merkle root of replica R , rt

[0181] random challenge, c

[0182] tuple of proofs, (π_{SEAL}, π_{POS})

[0183] outputs: bit b , equals 1 if proofs are valid

Practical PoSt construction

[0184] This section describes the construction of the PoSt protocol. A simplified protocol sketch is shown in FIG. 4. In one embodiment, the Setup and Verify operations are equivalent to those in the PoRep construction.

[0185] Proving space and time. The Prove algorithm generates a Proof-of-Spacetime for the replica. The prover receives a random challenge from the verifier and generates Proofs-of-Replication in sequence, using the output of a proof as an input of the challenge generation engine for a specified amount of iterations t (see FIG. 3).

Prove

[0186] inputs:

[0187] prover PoSt key pk_{POST}

[0188] replica R

[0189] random challenge c

[0190] time parameter t

[0191] outputs: a proof π_{POST}

Usage in Filecoin

[0192] The Filecoin protocol can employ Proof-of-Space-time to audit the storage offered by miners. To use PoSt in Filecoin, the scheme can be modified to be non-interactive. In a standard setting there is a prover and a verifier and the verifier sends challenges to the prover. In the filecoin setting, the prover can take challenges from the blockchain, so in such an embodiment the prover does not need to interact with the verifier (a non-interactive approach). Challenges can be generated using randomness from the blockchain.

[0193] FIG. 4 illustrates examples of proof-of-replication and proof-of-spacetime protocols. Here CRH denotes a collision-resistant hash, x is the NP-statement to be proven, and w is the witness.

Filecoin: A DSN Construction

[0194] The Filecoin DSN is a decentralized storage network that is auditable, publicly verifiable and designed on incentives. Clients pay a network of miners for data storage and retrieval; miners offer disk space and bandwidth in exchange for awards. Miners receive their awards only if the network can audit that their service was correctly provided.

[0195] In this section, the specification describes the Filecoin DSN construction, based on the DSN definition and Proof-of-Spacetime.

Setting

Participants

[0196] Any user can participate as a Client, a Storage Miner, and/or a Retrieval Miner in the Filecoin DSN.

[0197] Clients pay to store data and to retrieve data in the DSN, via Put and Get requests. Storage Miners provide data storage to the network. Storage Miners participate in Filecoin by offering their disk space and serving Put requests. To become Storage Miners, users pledge their storage by depositing collateral proportional to it.

[0198] Storage Miners respond to Put requests by committing to store the client's data for a specified time. Storage Miners generate Proofs-of-Spacetime and submit them to the blockchain to prove to the Network that they are storing the data through time. In case of invalid or missing proofs, Storage Miners can be penalized and lose part of their collateral. Storage Miners are also eligible to mine new blocks, and in doing so they receive a mining reward for creating a block and transaction fees for the transactions included in the block.

[0199] Retrieval Miners provide data retrieval to the Network. Retrieval Miners participate in Filecoin by serving data that users request via a Get request. Unlike Storage Miners, in one embodiment they are not required to pledge, commit to store data, or provide proofs of storage. It is natural for Storage Miners to also participate as Retrieval Miners. Retrieval Miners can obtain pieces directly from clients, or from the Retrieval Market.

The Network, N

[0200] In one embodiment, the system can personify all the users that run Filecoin full nodes as one single abstract entity: The Network. The Network acts as an intermediary that runs the Manage protocol; informally, at every new block in the Filecoin blockchain, full nodes manage the available storage, validate pledges, audit the storage proofs, and repair possible faults.

The Ledger

[0201] The Filecoin protocol is applied on top of a ledger-based token system, e.g., the Ledger, L. At any given time t (referred to as epoch), all users have access to L_t , the ledger at epoch t , which is a sequence of transactions. The ledger is append-only. The Filecoin DSN protocol can be implemented on ledgers that allows for the verification of Filecoin's proofs. In one embodiment, the protocol implements the ledger based on useful work.

The Markets

[0202] Demand for and supply of storage and retrieval meet at the two Filecoin markets/exchanges: Storage Market and Retrieval Market. The markets are two decentralized exchanges. Clients and miners set the prices for the services they request or provide by submitting orders to the respective markets. The exchanges provide a way for clients and miners to see matching orders and initiate deals. By running the Manage protocol, the Network provides rewards to miners and charges to clients if the service requested is successfully provided.

[0203] FIG. 5 illustrates an example of data structures in a decentralized storage network (DSN) scheme.

Data Structure

[0204] Pieces. A piece is some part of data that a client is storing in the DSN. For example, data can be deliberately divided into many pieces and each piece can be stored by a different set of Storage Miners.

[0205] Sectors. A sector is some disk space that a Storage Miner provides to the network. Miners store pieces from clients in their sectors and earn tokens for their services. In order to store pieces, Storage Miners must pledge their sectors to the network.

[0206] AllocationTable. The AllocTable is a data structure that keeps track of pieces and their assigned sectors. The AllocTable is updated at every block in the ledger and its Merkle root is stored in the latest block. In practice, the table is used to keep the state of the DSN, allowing for quick look-ups during proof verification. FIG. 5 illustrates an example of data structures in a decentralized storage network (DSN) scheme.

[0207] Orders. An order is a statement of intent to request or offer a service. Clients submit bid orders to the markets to request a service (respectively the Storage Market for storing data and the Retrieval Market for retrieving data) and Miners submit ask orders to offer a service. Examples of order data structures are shown in FIG. 10.

[0208] Orderbook. Orderbooks are sets of orders.

[0209] Pledge. A pledge is a commitment to offer storage (specifically a sector) to the network. A Storage Miner submits her pledge to the ledger in order to start accepting orders in the Storage Market. A pledge consists of the size of the pledged sector and the collateral deposited by the Storage Miner.

[0210] FIG. 6 illustrates an example execution of a Filecoin DSN, grouped by party and sorted chronologically by row.

Protocol

[0211] The clients, the Network, and the miners of the Filecoin DSN perform various operations. For example, the operations include the methods of the Get and the Put protocol an example of which is shown in FIG. 7 and the Manage protocol an example of which is shown in FIG. 8. An example protocol execution is shown in FIG. 6. An example of an overall Filecoin Protocol is shown in FIG. 1B.

Client Cycle

[0212] In one embodiment, the client can perform operations including storing data in Filecoin DSN and retrieving data from the Filecoin DSN.

[0213] 1. Put: Client stores data in Filecoin.

[0214] Clients can store their data by paying Storage Miners in Filecoin tokens. The Put protocol is described below.

[0215] A client initiates the Put protocol by submitting a bid order to the Storage Market orderbook (by submitting its order to the blockchain). When a matching ask order from miners is found, the client sends the piece of data to the miner. Both parties sign a deal order and submit it to the Storage Market orderbook.

[0216] Clients should be able to decide the amount of physical replicas of their pieces either by submitting multiple orders (or specifying a replication factor in the order). Higher redundancy results in a higher tolerance of storage faults.

[0217] 2. Get: Client retrieves data from Filecoin.

[0218] Clients can retrieve any data stored in the DSN by awarding Retrieval Miners in Filecoin tokens. The Get protocol is described below.

[0219] A client initiates the Get protocol by submitting a bid order to the Retrieval Market orderbook (by gossiping their order to the network). When a matching ask order from miners is found, the client receives the piece from the retrieval miner. When received, both parties sign a deal order and submit it to the blockchain to confirm that the exchange succeeded.

Mining Cycle (for Storage Miners)

[0220] 1. Pledge: Storage Miners pledge to provide storage to the Network.

[0221] Storage Miners pledge their storage to the network by depositing collateral via a pledge transaction in the blockchain, via Manage: PledgeSector. Storage miners deposit collateral for the time intended to provide the service, and the collateral is returned if the miner generates proofs of storage for the data the miner commit to store. If some proofs of storage fail, a proportional amount of collateral is lost.

[0222] Once the pledge transaction appears in the blockchain, miners can offer their storage in the Storage Market: a storage miner can set its price and add an ask order to the market's orderbook.

Manage: PledgeSector

[0223] inputs:

[0224] current allocation table allocTable

[0225] pledge request pledge

[0226] outputs: allocTable'

[0227] 2. Receive Orders: Storage Miners get storage requests from the Storage Market.

[0228] Once the pledge transaction appears in the blockchain (e.g., in the AllocTable), miners can offer their storage in the Storage Market: a storage miner can set its price and add an ask order to the market's orderbook via Put: AddOrders.

[0229] Put: AddOrders

[0230] inputs: list of orders $O^1 \dots O^n$

[0231] outputs: bit b, equals 1 if successful

[0232] Check if ask orders are matched with a corresponding bid order from a client, via Put: MatchOrders.

Put: MatchOrders

[0233] inputs:

[0234] the current Storage Market OrderBook

[0235] query order to match O^q

[0236] outputs: matching orders $O^1 \dots O^n$

[0237] Once orders are matched, clients send their data to the Storage Miners. When receiving the piece, miners run Put: ReceivePiece. When the data is received, both the miner and the client sign a deal order and submit it to the blockchain.

Put: ReceivePiece

[0238] inputs:

[0239] signing key for M_j ,

[0240] current orderbook OrderBook

[0241] ask order O_{ask}

[0242] bid order O_{bid}

[0243] piece p

[0244] outputs: deal order O_{deal} signed by C_i and M_j

[0245] 3. Seal: Storage Miners prepare the pieces for future proofs.

[0246] Storage Miners' storage is divided in sectors, each sector contains pieces assigned to the miner. The Network keeps track of each Storage Miners' sector via the allocation table. When a Storage Miner sector is filled, the sector is sealed. Sealing is a slow, sequential operation that transforms the data in a sector into a replica, a unique physical copy of the data that is associated to the public key of the Storage Miner. In one implementation sealing is an operation used for Proof-of-Replication.

Manage: SealSector

[0247] inputs:

[0248] miner public/private key pair M

[0249] sector index j

[0250] allocation table allocTable

[0251] outputs: a proof π_{SEAL} , a root hash rt

[0252] 4. Prove: Storage Miners prove they are storing the committed pieces.

[0253] When Storage Miners are assigned data, they must repeatedly generate proofs of replication to guarantee they are storing the data. Proofs are posted on the blockchain and the Network verifies them.

Manage: ProveSector

[0254] inputs:

[0255] miner public/private key pair M

[0256] sector index j

[0257] challenge c

[0258] outputs: a proof π_{POS}

Mining Cycle (for Retrieval Miners)

[0259] In one embodiment, the retrieval miners can perform operations including receiving orders and sending pieces to clients.

[0260] 1. Receive Orders: Retrieval Miners get data requests from the Retrieval Market.

[0261] Retrieval Miners announce their pieces by gossiping their ask orders to the network: A Retrieval Miner set its price and adds an ask order to the market's orderbook.

Get: AddOrders

[0262] inputs: list of orders $O_1 \dots O_n$

[0263] outputs: none

[0264] Then, the Retrieval Miner checks if its orders are matched with a corresponding bid order from a client.

Get: MatchOrders

[0265] inputs:

[0266] the current Retrieval Market OrderBook

[0267] query order to match O^q

[0268] outputs: matching orders $O^1 \dots O^n$

[0269] 2. Send: Retrieval Miners send pieces to the client.

[0270] Once orders are matched, Retrieval Miners send the requested piece to the client. When the client receives the piece, both the miner and the client sign a deal order for the requested piece of data and submit the signed deal order to the blockchain.

Put: SendPiece

[0271] inputs:

[0272] an ask order O_{ask}

[0273] a bid order O_{bid}

[0274] a piece p

[0275] outputs: a deal order "Odeal: signed by M_i

Network Cycle

[0276] In one embodiment, the network can perform operations including assigning clients' pieces to storage miners and repairing network faults.

[0277] 1. Assign: The Network assigns clients' pieces to Storage Miners' sectors. Clients initiate the Put protocol by submitting a bid order in the Storage Market. When ask and bid orders match, the involved parties jointly commit to the exchange and submit a deal order in the market. At this point, the Network assigns the data to the miner and makes a note of it in the allocation table.

Manage: AssignOrders

[0278] inputs:

[0279] deal orders $O^1 \text{deal} \dots O^n \text{deal}$

[0280] allocation table allocTable

[0281] outputs: updated allocation table allocTable'

[0282] 2. Repair: The Network finds faults and attempt to repair them. All the storage allocations are public to participants in the network. At a block, the Network checks if the required proofs for each assignment are present, checks that they are valid, and acts accordingly:

[0283] if any proof is missing or invalid, the network penalizes the Storage Miners by taking part of their collateral,

[0284] if a large amount of proofs are missing or invalid (defined by a system parameter fault), the network considers the Storage Miner faulty, settles the order as failed and reintroduces a new order for the same piece into the market,

[0285] if every Storage Miner storing this piece is faulty, then the piece is lost and the client gets refunded.

Manage: RepairOrders

[0286] inputs:

[0287] current time t

[0288] current ledger L

[0289] table of storage allocations allocTable

[0290] outputs: orders to repair $O^1 \text{deal} \dots O^n \text{deal}$, updated allocation table allocTable

Beneficial Characteristics of Implementations of the DSN

[0291] The following are the intuitions on how Filecoin DSNs achieve integrity, retrievability, public verifiability and incentive-compatibility.

[0292] Achieving Integrity: Pieces can be named after their cryptographic hash. After a Put request, clients only need to store this hash to retrieve the data via Get and to verify the integrity of the content received.

[0293] Achieving Retrievability: In a Put request, clients specify the replication factor and the type of erasure coding desired, specifying the storage as (f, m) -tolerant. The assumption is that given m Storage Miners storing the data, a maximum of f faults are tolerated. By storing data in more than one Storage Miner, a client can increase the chances of recovery, in case Storage Miners go offline or disappear.

[0294] Achieving Public Verifiability and Auditability: Storage Miners submit their proofs of storage (π_{SEAL}, π_{POST}) to the blockchain. Any user in the network can verify the validity of these proofs, without having access to the outsourced data. Since the proofs are stored on the blockchain, they are a trace of operation that can be audited at any time.

[0295] Achieving Incentive Compatibility: Informally, miners are rewarded for the storage they are providing. When miners commit to store some data, then they can generate proofs. Miners that skip proofs are penalized (by losing part of their collateral) and not rewarded for their storage.

[0296] Achieving Confidentiality: Clients that desire for their data to be stored privately, can encrypt their data before submitting them to the network.

[0297] FIG. 7 illustrates examples of the put and get protocols in a Filecoin DSN.

[0298] FIG. 8 illustrates example of the manage protocol in a Filecoin DSN.

[0299] FIG. 9 illustrates an example generic protocol for a verifiable market/exchange.

Filecoin Storage and Retrieval Markets

[0300] Filecoin has two markets: the Storage Market and the Retrieval Market. The two markets have the same structure but different design. The Storage Market allows Clients to reward Storage Miners to store data. The Retrieval Market allows Clients to retrieve data by rewarding Retrieval Miners to deliver the data. In both cases, clients and miners can set their offer and demand prices or accept current offers. The exchanges are run by the Network, the network of full nodes in Filecoin. The network provides for miners to be rewarded by a client when providing a service to the client.

Verifiable Markets

[0301] Exchange Markets are protocols that facilitate exchange of a specific good or service. They do this by enabling buyers and sellers to conduct transactions. In the Filecoin DSN, exchanges are verifiable: a decentralized network of participants are able to verify the exchange between buyers and sellers.

[0302] In one embodiment, in verifiable markets, where no single entity governs an exchange, transactions are transparent, and anybody can participate, e.g., pseudonymously. Verifiable Market protocols operate the exchange of goods/services in a decentralized fashion: consistency of the orderbooks, orders settlements and correct execution of services are independently verified via the participants, i.e., miners and full nodes in the case of Filecoin. For example, verifiable markets can have the following construction:

[0303] A verifiable market can include a protocol with two phases: order matching and settlement. Orders are statements of intent to buy or sell a security, good or service and the orderbook is the list of all the available orders.

Storage Market

[0304] The Storage Market is a verifiable market that allows clients (i.e. buyers) to request storage for their data and Storage Miners (i.e. sellers) to offer their storage. In one embodiment, the Storage Market protocol includes the following requirements:

[0305] In-chain orderbook: (1) Storage Miners orders are public, so that the lowest price is known to the network and clients can make informed decision on their orders, (2) client orders are submitted to the orderbook, even when a storage miner accepts the lowest price offered by a client.

[0306] Participants committing their resources: both parties commit to their resources as a way to avoid disservice, to avoid a Storage Miner not providing a service it offers, and to avoid clients not having funds available to award a Storage Miner upon the Storage Miner's completion of the promised service. In order to participate to the Storage Market, a Storage Miner pledges collateral, depositing an amount of collateral proportional to its amount of proposed storage in the DSN. In this way, the Network can incentivize a Storage Miner to provide proofs of storage for the pieces it committed to store. Similarly, clients must deposit the awards specified in the order, ensuring availability of funds during settlement.

[0307] Self-organization to handle faults: In one embodiment, orders are settled if Storage Miners have proved that they have stored the pieces for the duration of the agreed-upon time period. The Network can verify the existence and the correctness of the proof(s). The Network can also act according to the rules outlined in the Repair description provided above.

[0308] FIG. 10 illustrates an example of order data structures for retrieval and storage markets/exchanges.

Data Structures

[0309] Put Orders. There are three types of orders: bid orders, ask orders and deal orders. Storage Miners create ask orders to add storage, clients create bid orders to request storage, when both parties agree on a price, they jointly create a deal order. The data structures of the orders are shown in detail in FIG. 10, and the parameters of the orders are explicitly defined. Put Orderbook. The Orderbook in the Storage Market is the set of currently valid and open ask, bid and deal orders. Users can interact with the orderbook via the methods defined in the Put protocol: AddOrders, MatchOrders as described in FIG. 7.

[0310] The orderbook is public and honest users have the same view of the orderbook. At every epoch, new orders are added to the orderbook if new order transactions (txorder) appear in new blockchain blocks; orders are removed if they are cancelled, expired or settled. Orders are added in blockchain blocks, hence in the orderbook, if they are valid.

[0311] One embodiment defines the validity of bid, ask, deal orders as follows: (Valid bid order): A bid order from client C_i , O_{bid} : $=\langle \text{size, funds [price, time, coll, coding]} \rangle_{C_i}$ is valid if:

[0312] C_i has at least the amount of funds available in their account.

[0313] time is not set in the past

[0314] The order guarantees at least a minimum amount of epochs of storage. (Valid ask order): An ask order from Storage Miner M_i , O_{ask} : $=\langle \text{space, price} \rangle_{M_i}$ is valid if:

[0315] M_i has pledged to be a miner and the pledge will not expire before time epochs.

[0316] space must be less than M_i 's available storage: M_i pledged storage minus the storage committed in the orderbook (in ask and deal orders).

(Valid deal order): A deal order O_{deal} : $=\langle \text{ask, bid, ts} \rangle_{C_i, M_j}$ is valid if:

[0317] ask references an order O_{ask} such that: it is in the Storage Market OrderBook, no other deal orders in the Storage Market OrderBook mention it, it is signed by C_i .

[0318] bid references an order O_{bid} such that: it is in the Storage Market OrderBook, no other deal orders in the Storage Market OrderBook mention it, it is signed by M_j .

[0319] ts is not set in the future or too far (e.g., more than 30 minutes to an hour) in the past. If a malicious client receives a signed deal from a Storage Miner, but never adds it to the orderbook, then the Storage Miner cannot re-use the storage committed in the deal. The field ts prevents this attack because, after ts, the order becomes invalid and cannot be submitted in the orderbook.

The Storage Market Protocol

[0320] With reference to FIG. 11, one embodiment of the Storage Market protocol is divided in two phases: order matching and settlement:

[0321] Order Matching: Clients and Storage Miners submit their orders to the orderbook by submitting a transaction to the blockchain (step 1). When orders are matched, the client sends the piece to the Storage Miner and both parties sign a deal order and submit it to the orderbook (step 2).

[0322] Settlement: Storage Miners seal their sectors (step 3a), generate proofs of storage for the sector containing the piece and submit them to the blockchain regularly (step 3b); meanwhile, the rest of the network must verify the proofs generated by the miners and repair possible faults (step 3c).

[0323] FIG. 12 illustrates an example of a detailed retrieval market/exchange protocol.

[0324] With reference to FIG. 12, the illustrated example of the Retrieval Market protocol is divided in two phases: order matching and settlement:

[0325] Order Matching: Clients and Retrieval Miners submit their orders to the orderbook by gossiping their

orders (step 1). When orders are matched, the client and the Retrieval Miners establish a micropayment channel (step 2).

[0326] Settlement: Retrieval Miners send small parts of the piece to the client and for each piece the client sends to the miner a signed receipt (step 3). The Retrieval Miner presents the delivery receipts to the blockchain to get its rewards (step 4).

Useful Work Consensus

[0327] FIG. 13 illustrates an example of a leader election process via an expected consensus protocol. More generally, the Filecoin DSN protocol can be implemented on top of any consensus protocol that allows for verification of the Filecoin's proofs. In one embodiment, a consensus protocol is implemented based on useful work. Instead of using a wasteful Proof-of-Work computation, miners can utilize useful Proof-of-Spacetime work to participate in a consensus protocol.

[0328] Useful Work. For example, work done by the miners in a consensus protocol is useful if the outcome of the computation is valuable to the network, beyond securing the blockchain.

Motivation

[0329] While securing the blockchain is a significant consideration, Proof-of-Work schemes often require solving puzzles whose solutions are not reusable or require a substantial amount of wasteful computation to find.

[0330] Non-reusable Work: Most permissionless blockchains require miners to solve a hard computational puzzle, such as inverting a hash function. Often the solutions to such a puzzle is useless and does not have any inherent value beyond securing the network.

[0331] Wasteful Work: Solving hard puzzles can be really expensive in terms of cost of machinery and energy consumed, especially if the puzzles involved rely heavily on computational power. When the mining algorithm is parallel, then the prevalent factor to solve the puzzle is computational power.

[0332] Attempts to reduce waste: Ideally, the majority of a network's resources should be spent on useful work. Some efforts require miners to use more energy-efficient solutions. Some efforts replace hard to solve puzzles with a traditional byzantine agreement based on Proof-of-Stake, where stakeholders vote on the next block proportional to their share of tokens in the system.

[0333] One embodiment of a consensus protocol utilizes useful work based on storing client data.

Filecoin Consensus

[0334] In one embodiment of a useful work consensus protocol, the probability that the network elects a miner to create a new block (e.g., the voting power of the miner) is proportional to its storage currently in use in relation to the rest of the network. The design of the Filecoin protocol is such that miners would rather invest in storage than in computing power to parallelize the mining computation. Miners offer storage and re-use the proof that data is being stored to participate in the consensus.

Modeling Mining Power

[0335] Power Fault Tolerance. Power Fault Tolerance is an abstraction that re-frames byzantine faults in terms of participants' influence over the outcome of the protocol. Every participant controls some power of which n is the total power in the network, and f is the fraction of power controlled by faulty or adversarial participants.

[0336] Power in Filecoin. In one embodiment of Filecoin, the power p_i^t of miner M_i at time t is the sum of the M_i 's storage assignments. The influence I_i^t of M_i is the fraction of M_i 's power over the total power in the network.

[0337] In one embodiment of Filecoin, power has the following properties:

[0338] Public: The total amount of storage currently in use in the network is public. By reading the blockchain, anyone can calculate the storage assignments of each miner-hence anyone can calculate the power of each miner and the total amount of power at any point in time.

[0339] Publicly verifiable: For each storage assignment, miners generate Proofs-of-Spacetime, proving that the service is being provided. By reading the blockchain, anyone can verify if the power claimed by a miner is correct.

[0340] Variable: At any point in time, miners can add new storage in the network by pledging with a new sector and filling the sector. In this way, a miner can change the amount of power it has through time.

Accounting for Power with Proof-of-Spacetime

[0341] In one embodiment, every Δ_{proof} blocks, the protocol asks miners to submit Proofs-of-Spacetime to the network, which are only successfully added to the blockchain if the majority of power in the network considers them valid. At every block, every full node updates the AllocTable, adding new storage assignments, removing expiring ones and marking missing proofs.

[0342] In one embodiment, the power of a miner M_i can be calculated and verified by summing the entries in the AllocTable, which can be done in at least two ways:

[0343] Full Node Verification: If a node has the full blockchain log, run the NetworkProtocol from the genesis block to the current block and read the AllocTable for miner M_i . This process verifies every Proof-of-Spacetime for the storage currently assigned to M_i .

[0344] Simple Storage Verification: Assume a light client has access to a trusted source that broadcasts the latest block. A light client is a client that doesn't have access to the entire blockchain, instead it only has access to the hash of the latest block from a trusted source and a way to request block headers from the network. A light client can request from nodes in the network: (1) the current AllocTable entry for miner M_i , (2) a Merkle path that proves that the entry was included in the state tree of the last block, (3) the headers from the genesis block until the current block. In this way, the light client can delegate the verification of the Proof-of-Spacetime to the network.

[0345] The security of the power calculation comes from the security of Proof-of-Spacetime. In this setting, PoSt ensures that the miner cannot lie about the amount of assigned storage they have. Indeed, a miner cannot claim to store more than the data it is storing, since this would require spending time fetching and running the slow PoSt: Setup,

and such a miner cannot generate proofs faster by parallelizing the computation, since PoSt: Prove is a sequential computation.

Using Power to Achieve Consensus

[0346] In one embodiment, the Filecoin consensus extends Proof-of-Stake consensus protocols, where stake is replaced with assigned storage. For example, the Filecoin consensus can be implemented using Expected Consensus. In one embodiment, Expected Consensus elects, at every round, one (or more) miners where the probability of a miner winning an election is proportional to that miner's assigned storage.

[0347] Expected Consensus. The basic intuition of Expected Consensus (EC) is to deterministically, unpredictably, and secretly elect a small set of leaders (e.g., 1-4) at each epoch. On expectation, the number of elected leaders per epoch is 1, but some epochs may have zero or many leaders. Leaders extend the chain by creating a block and propagating it to the network. At each epoch, the chain is extended with one or multiple blocks. In one embodiment, each leader can add one block. In case of a leaderless epoch, an empty block is added to the chain. Although the blocks in chain can be linearly ordered, the block chain's data structure can be a direct acyclic graph. EC is a probabilistic consensus, where each epoch introduces more certainty over previous blocks, eventually reaching enough certainty that the likelihood of a different history is sufficiently small. A block is committed if the majority of the participants add their weight on the chain where the block belongs to, by extending the chain or by signing blocks.

[0348] Electing Miners. At every epoch, each miner checks if they are elected leader, this is done similarly to existing consensus protocols:

[0349] (EC Election in Filecoin) A miner M_i is a leader at time t if the following condition is met:

$$\mathcal{H}(\langle t \parallel \text{rand}(t) \rangle_{M_i}) / 2^L \leq \frac{p_i^t}{\sum_j p_j^t}$$

[0350] Where $\text{rand}(t)$ is a function that extracts randomness from the blockchain at epoch t , p_i^t is the power of M_i . Consider the size of $\mathcal{H}(m)$ to be L for any m , \mathcal{H} to be a secure cryptographic hash function and $\langle m \rangle_{M_i}$ to be a message m signed by M_i , such that:

$$\langle m \rangle_{M_i} = ((m), \text{SIG}_{M_i}, (\mathcal{H}(m)))$$

[0351] FIG. 13 further shows the protocol between a miner (ProveElect) and a network node (VerifyElect).

[0352] This election scheme provides three properties: fairness, secrecy and public verifiability.

[0353] Fairness: each participant has only one trial for each election, since signatures are deterministic and t and $\text{rand}(t)$ are fixed. Assuming \mathcal{H} is a secure cryptographic hash function, then $\mathcal{H}(\langle t \parallel \text{rand}(t) \rangle_{M_i}) / 2^L$

must be a real number uniformly chosen from $(0, 1)$. Hence, the probability for the equation to be true must be $p_i^t / \sum_j p_j^t$, which is equal to the miner's portion of power within the network. Because this probability is linear in power, this likelihood is preserved under

splitting or pooling power. Note that the random value $\text{rand}(t)$ is not known before time t .

[0354] Secret: an efficient adversary that does not own the secret key M_i can compute the signature with negligible probability, given the assumptions of digital signatures.

[0355] Public Verifiability: an elected leader $i \in L'$ can convince an efficient verifier by showing t , $\text{rand}(t)$,

$\mathcal{H}(\langle t \parallel \text{rand}(t) \rangle_{M_i}) / 2^L$ given the previous point, no efficient adversary can generate a proof without having a winning secret key.

[0356] FIG. 14 is a flowchart of an example process 1400 for a space-time proof. The process 1400 includes: receiving 1402 at a prover, an initial input challenge; producing 1403, e.g., at the prover, an output proof proving that the prover has access to specified data for a specified time, e.g., for at least a minimum period of time (the producing step 1403 includes: producing 1404, e.g., at a prover, an initial proof responsive to the initial input challenge, the initial proof proving that the prover is storing specified data; generating 1406, e.g., at the prover, a new input challenge based at least in part on the initial proof; producing 1408, e.g., at the prover, a new proof responsive to the new input challenge, the new proof proving that the prover is storing the specified data; and

repeating 1409, e.g., at the prover, the generating step and the producing a new proof responsive to the new input challenge step a number of times, the repeating step generating sequential proofs of storage to determine time of storage, wherein each generating step is based at least in part on a most recent new proof, the repeating step 1409 producing 1410 an output proof; forwarding 1412 the output proof to a blockchain; and receiving 1414 tokens at least in part in response to forwarding the output proof to the blockchain. The output proof can be labeled with a loop counter index. The method can further include checking, at a verifier, if the prover is storing the specified data for at least a minimum period of time.

[0358] FIG. 15 is a flowchart of an example process 1500 for determining a winning prover from a plurality of candidate provers. The process 1500 includes: determining 1502 the power of a plurality of miners; electing 1504 a miner from the plurality of miners; receiving 1506 a new block from the elected miner; and adding 1508 the new block to a blockchain-based file storage system.

[0359] FIG. 16 is a flowchart of an example process 1600 for proof of replication. The process 1600 includes: receiving 1602 at a prover a file to be stored and storing the file; encoding 1604 the stored file using an encoding function that takes at least a minimum encoding time; receiving 1606 at the prover a challenge; producing 1608 a proof at the prover in response to the challenge from the prover within a verify time period; and taking an action 1610 in response to the proof.

[0360] Embodiments of the subject matter described in this specification can be implemented as one or more computer programs, i.e., one or more modules of computer program instructions encoded on a tangible non-transitory storage medium for execution by, or to control the operation of, data processing apparatus. The computer storage medium can be a machine-readable storage device, a machine-readable storage substrate, a random or serial access memory device, or a combination of one or more of them. Alterna-

tively or in addition, the program instructions can be encoded on an artificially-generated propagated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus for execution by a data processing apparatus.

[0361] The term “data processing apparatus” refers to data processing hardware and encompasses all kinds of apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can also be, or further include, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit). The apparatus can optionally include, in addition to hardware, code that creates an execution environment for computer programs, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them.

[0362] A computer program, which may also be referred to or described as a program, software, a software application, an app, a module, a software module, a script, or code, can be written in any form of programming language, including compiled or interpreted languages, or declarative or procedural languages; and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data, e.g., one or more scripts stored in a markup language document, in a single file dedicated to the program in question, or in multiple coordinated files, e.g., files that store one or more modules, sub-programs, or portions of code. A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a data communication network.

[0363] The processes and logic flows described in this specification can be performed by one or more programmable computers executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by special purpose logic circuitry, e.g., an FPGA or an ASIC, or by a combination of special purpose logic circuitry and one or more programmed computers.

[0364] Computers suitable for the execution of a computer program can be based on general or special purpose micro-processors or both, or any other kind of central processing unit. Generally, a central processing unit will receive instructions and data from a read-only memory or a random access memory or both. The essential elements of a computer are a central processing unit for performing or executing instructions and one or more memory devices for storing instructions and data. The central processing unit and the memory can be supplemented by, or incorporated in, special purpose logic circuitry. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. However, a computer need not have such devices. Moreover, a computer can be embedded in another device, e.g., a mobile telephone, a personal digital assistant (PDA), a mobile audio or video player, a game console, a

Global Positioning System (GPS) receiver, or a portable storage device, e.g., a universal serial bus (USB) flash drive, to name just a few.

[0365] Computer-readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks.

[0366] To provide for interaction with a user, embodiments of the subject matter described in this specification can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input. In addition, a computer can interact with a user by sending documents to and receiving documents from a device that is used by the user; for example, by sending web pages to a web browser on a user's device in response to requests received from the web browser. Also, a computer can interact with a user by sending text messages or other forms of message to a personal device, e.g., a smartphone, running a messaging application, and receiving responsive messages from the user in return.

[0367] Embodiments of the subject matter described in this specification can be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface, a web browser, or an app through which a user can interact with an implementation of the subject matter described in this specification, or any combination of one or more such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network (LAN) and a wide area network (WAN), e.g., the Internet.

[0368] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other. In some embodiments, a server transmits data, e.g., an HTML page, to a user device, e.g., for purposes of displaying data to and receiving user input from a user interacting with the device, which acts as a client. Data generated at the user device, e.g., a result of the user interaction, can be received at the server from the device.

[0369] While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any invention or on the scope of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular

inventions. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially be claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

[0370] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system modules and components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

[0371] Particular embodiments of the subject matter have been described. Other embodiments are within the scope of the following claims. For example, the actions recited in the claims can be performed in a different order and still achieve desirable results. As one example, the processes depicted in the accompanying figures do not necessarily require the particular order shown, or sequential order, to achieve desirable results. In some cases, multitasking and parallel processing may be advantageous.

1. (canceled)
2. (canceled)
3. (canceled)
4. (canceled)
5. (canceled)
6. (canceled)
7. (canceled)
8. (canceled)
9. (canceled)
10. (canceled)
11. (canceled)
12. (canceled)
13. (canceled)
14. (canceled)
15. (canceled)
16. (canceled)
17. (canceled)
18. (canceled)
19. (canceled)
20. (canceled)

21. A method comprising:

receiving a retrieval bid order from a client, the retrieval order specifying retrieval of first stored data;

posting, by a retrieval miner, a retrieval ask order to the client, the retrieval ask order for retrieving the first stored data;

upon a match between the retrieval bid order and the retrieval ask order, cryptographically signing, by the

retrieval miner, a deal to retrieve the first stored data, the deal specifying a payment amount for retrieving the first stored data;

receiving, at the retrieval miner, the deal signed by the client;

retrieving and forwarding the first stored data to the client; and

receiving payment for retrieving and forwarding the first stored data to the client as specified in the deal.

22. The method claim 21 wherein the retrieval ask order of the miner is gossiped off to a client off chain.

23. The method of claim 21 wherein the retrieval miner sends the retrieved data to the client in parts and after each part is sent the retrieval miner receives a corresponding micro-award.

24. The method of claim 23 wherein before micro-awards, tokens are locked by the client for use as a micro-award for the retrieval miner.

25. The method of claim 24 wherein the retrieval miner is able to claim its award once the client has logged a transfer within the transactions section of a blockchain.

26. The method of claim 23 wherein the method further comprises:

for each of a plurality of parts of the retrieved data,

sending a corresponding part to the client,

receiving a delivery receipt from the client,

forwarding the receipt to the blockchain, and

receiving a corresponding reward.

27. The method of claim 21 wherein the retrieval bid order of the client is gossiped to the retrieval miner off chain.

28. The method of claim 21 wherein if there is a match between the retrieval bid order of the client and retrieval ask order of the miner, the retrieval minor and the client can sign the deal.

29. The method of claim 21 wherein the method further comprises establishing a micropayment channel with the client.

30. A system comprising:

one or more computers and one or more storage devices on which are stored instructions that are operable, when executed by the one or more computers, to cause the one or more computers to perform operations comprising:

receiving a retrieval bid order from a client, the retrieval order specifying retrieval of first stored data;

posting, by a retrieval miner, a retrieval ask order to the client, the retrieval ask order for retrieving the first stored data;

upon a match between the retrieval bid order and the retrieval ask order, cryptographically signing, by the retrieval miner, a deal to retrieve the first stored data, the deal specifying a payment amount for retrieving the first stored data;

receiving, at the retrieval miner, the deal signed by the client;

retrieving and forwarding the first stored data to the client;

receiving payment for retrieving and forwarding the first stored data to the client as specified in the deal

31. The system of claim 30 wherein the retrieval ask order of the miner is gossiped off to a client off chain.

32. The system of claim 30 wherein the retrieval miner sends the retrieved data to the client in parts and after each part is sent the retrieval miner receives a corresponding micro-award.

33. The system of claim 32 wherein before micro-awards, tokens are locked by the client for use as a micro-award for the retrieval miner.

34. The system of claim 33 wherein the retrieval miner is able to claim its award once the client has logged a transfer within the transactions section of a blockchain.

35. The system of claim 32 wherein the operations further comprise:

- for each of a plurality of parts of the retrieved data,
 - sending a corresponding part to the client,
 - receiving a delivery receipt from the client,
 - forwarding the receipt to the blockchain, and
 - receiving a corresponding reward.

36. The system of claim 30 wherein the retrieval bid order of the client is gossiped to the retrieval miner off chain.

37. The system of claim 30 wherein if there is a match between the retrieval bid order of the client and retrieval ask order of the miner, the retrieval minor and the client can sign the deal.

38. The system of claim 30 wherein the operations further comprise establishing a micropayment channel with the client.

* * * * *